47th International Symposium on Mathematical Foundations of Computer Science

MFCS 2022, August 22–26, 2022, Vienna, Austria

Edited by

Stefan Szeider
Robert Ganian
Alexandra Silva

LIPIcs

Editors

Stefan Szeider
TU Wien, Vienna, Austria
sz@ac.tuwien.ac.at

Robert Ganian
TU Wien, Vienna, Austria
rganian@ac.tuwien.ac.at

Alexandra Silva
Cornell University, Ithaca, NY, USA
alexandra.silva@cornell.edu

ACM Classification 2012
Theory of computation; Mathematics of computing

ISBN 978-3-95977-256-3

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/978-3-95977-256-3.

Publication date
August, 2022

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.

Digital Object Identifier: 10.4230/LIPIcs.MFCS.2022.0

ISBN 978-3-95977-256-3 **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.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Stefan Szeider, Robert Ganian, and Alexandra Silva 0:xi

Conference Organization
.... 0:xiii

Invited Talks

Long Cycles in Graphs: Extremal Combinatorics Meets Parameterized Algorithms
Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov 1:1–1:4

Modern Dynamic Data Structures
Monika Henzinger 2:1–2:2

An Updated Survey of Bidding Games on Graphs
Guy Avni and Thomas A. Henzinger 3:1–3:6

Probabilistic Model Checking for Strategic Equilibria-Based Decision Making: Advances and Challenges
Marta Kwiatkowska, Gethin Norman, David Parker, Gabriel Santos, and Rui Yan 4:1–4:22

Online Bipartite Matching and Adwords
Vijay V. Vazirani 5:1–5:11

Regular Papers

Parameterized Complexity of Non-Separating and Non-Disconnecting Paths and Sets
Ankit Abhinav, Susobhan Bandopadhyay, Aritra Banik, Yasuaki Kobayashi, Shunsuke Nagano, Yota Otachi, and Saket Saurabh 6:1–6:15

Comonadic semantics for hybrid logic
Samson Abramsky and Dan Marsden 7:1–7:14

The Complexity of Periodic Energy Minimisation
Duncan Adamson, Argyrios Deligkas, Vladimir V. Gusev, and Igor Potapov 8:1–8:15

Weighted Counting of Matchings in Unbounded-Treewidth Graph Families
Antoine Amarilli and Mikaël Monet 9:1–9:15

On Upward-Planar L-Drawings of Graphs
Patrizio Angelini, Steven Chaplick, Sabine Cornelsen, and Giordano Da Lozzo ... 10:1–10:15

RAC Drawings of Graphs with Low Degree
Patrizio Angelini, Michael A. Bekos, Julia Katheder, Michael Kaufmann, and Maximilian Pfister 11:1–11:15

Polynomial Time Algorithm for ARRIVAL on Tree-Like Multigraphs
David Auger, Pierre Coucheney, and Loric Duhazé 12:1–12:14

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Graph Realization of Distance Sets
Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz 13:1–13:14

On the Role of the High-Low Partition in Realizing a Degree Sequence by a Bipartite Graph
Amotz Bar-Noy, Toni Böhnlein, David Peleg, and Dror Rawitz 14:1–14:15

Towards a Model Theory of Ordered Logics: Expressivity and Interpolation
Bartosz Bednarczyk and Reijo Jaakkola 15:1–15:14

Algebraic Representations of Unique Bipartite Perfect Matching
Gal Beniamini 16:1–16:17

Fixed-Point Cycles and Approximate EFX Allocations
Benjamin Aram Berendsohn, Simona Boyadzhiyska, and László Kozma 17:1–17:13

Improved Lower Bound, and Proof Barrier, for Constant Depth Algebraic Circuits
C. S. Bhargav, Sagnik Dutta, and Nitin Saxena 18:1–18:16

Conflict-Free Coloring on Claw-Free Graphs and Interval Graphs
Sriram Bhyravarapu, Subrahmanyam Kalyanasundaram, and Rogers Mathew 19:1–19:14

Skolem Meets Schanuel
Yuri Bilu, Florian Luca, Joris Nieuwveld, Joël Ouaknine, David Purser, and James Worrell 20:1–20:15

Deepening the (Parameterized) Complexity Analysis of Incremental Stable Matching Problems
Niclas Boehmer, Klaus Heeger, and Rolf Niedermeier 21:1–21:15

Tree Exploration in Dual-Memory Model
Dominik Bojko, Karol Gotfryd, Dariusz R. Kowalski, and Dominik Pająk 22:1–22:16

On Vanishing Sums of Roots of Unity in Polynomial Calculus and Sum-Of-Squares
Ilario Bonacina, Nicola Galesi, and Massimo Lauria 23:1–23:15

Complete ZX-Calculi for the Stabiliser Fragment in Odd Prime Dimensions
Robert I. Booth and Titouan Carette 24:1–24:15

Extending Partial Representations of Circle Graphs in Near-Linear Time
Guido Brückner, Ignaz Rutter, and Peter Stumpf 25:1–25:14

Boundaries to Single-Agent Stability in Additively Separable Hedonic Games
Martin Bullinger 26:1–26:15

Bounded Degree Nonnegative Counting CSP
Jin-Yi Cai and Daniel P. Szabo 27:1–27:16

Continuous Rational Functions Are Deterministic Regular
Olivier Carton and Gaëtan Douéneau-Tabot 28:1–28:13

On Kernels for d-Path Vertex Cover
Radovan Červený, Pratibha Choudhary, and Ondřej Suchý 29:1–29:14

On Synthesizing Computable Skolem Functions for First Order Logic
Supratik Chakraborty and S. Akshay 30:1–30:15

Sample Compression Schemes for Balls in Graphs
Jérémie Chalopin, Victor Chepoi, Fionn Mc Inerney, Sébastien Ratel, and Yann Vaxès 31:1–31:14

On Algorithms Based on Finitely Many Homomorphism Counts
Yijia Chen, Jörg Flum, Mingjun Liu, and Zhiyang Xun 32:1–32:15

Higher-Order Quantified Boolean Satisfiability
Dmitry Chistikov, Christoph Haase, Zahra Hadizadeh, and Alessio Mansutti 33:1–33:15

On Dynamic $\alpha + 1$ Arboricity Decomposition and Out-Orientation
Aleksander B. G. Christiansen, Jacob Holm, Eva Rotenberg, and Carsten Thomassen 34:1–34:15

LO_v-Calculus: A Graphical Language for Linear Optical Quantum Circuits
Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron 35:1–35:16

Resource Optimisation of Coherently Controlled Quantum Computations with the PBS-Calculus
Alexandre Clément and Simon Perdrix 36:1–36:15

A Complexity Approach to Tree Algebras: the Polynomial Case
Thomas Colcombet and Arthur Jaquard 37:1–37:14

Enumeration Classes Defined by Circuits
Nadia Creignou, Arnaud Durand, and Heribert Vollmer 38:1–38:14

Bounding the Escape Time of a Linear Dynamical System over a Compact Semialgebraic Set
Julian D'Costa, Engel Lefaucheux, Eike Neumann, Joël Ouaknine, and James Worrell 39:1–39:14

The Pseudo-Reachability Problem for Diagonalisable Linear Dynamical Systems
Julian D'Costa, Toghrul Karimov, Rupak Majumdar, Joël Ouaknine, Mahmoud Salamati, and James Worrell 40:1–40:13

New Lower Bounds and Upper Bounds for Listing Avoidable Vertices
Mingyang Deng, Virginia Vassilevska Williams, and Ziqian Zhong 41:1–41:14

Constant-Factor Approximation Algorithm for Binary Search in Trees with Monotonic Query Times
Dariusz Dereniowski and Izajasz Wrosz 42:1–42:15

On the Identity Problem for Unitriangular Matrices of Dimension Four
Ruiwen Dong 43:1–43:14

Regular Monoidal Languages
Matthew Earnshaw and Paweł Sobociński 44:1–44:14

Oracle with $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$, but No Many-One Completeness in UP, DisjNP, and DisjCoNP
Anton Ehrmanntraut, Fabian Egidy, and Christian Glaßer 45:1–45:15

Exact Matching in Graphs of Bounded Independence Number
Nicolas El Maalouly and Raphael Steiner 46:1–46:14

CNF Encodings of Parity
Gregory Emdin, Alexander S. Kulikov, Ivan Mihajlin, and Nikita Slezkin 47:1–47:12

On the Number of Quantifiers as a Complexity Measure
Ronald Fagin, Jonathan Lenchner, Nikhil Vyas, and Ryan Williams 48:1–48:14

Non-Determinism in Lindenmayer Systems and Global Transformations
Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher 49:1–49:13

A Robust Class of Languages of 2-Nested Words
Séverine Fratani, Guillaume Maurras, and Pierre-Alain Reynier 50:1–50:15

Metric Dimension Parameterized by Feedback Vertex Set and Other Structural Parameters
Esther Galby, Liana Khazaliya, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale 51:1–51:15

Graph Similarity Based on Matrix Norms
Timo Gervens and Martin Grohe 52:1–52:15

Approximation Algorithms for Covering Vertices by Long Paths
Mingyang Gong, Jing Fan, Guohui Lin, and Eiji Miyano 53:1–53:14

The Hamilton Compression of Highly Symmetric Graphs
Petr Gregor, Arturo Merino, and Torsten Mütze 54:1–54:14

Dispersing Obnoxious Facilities on Graphs by Rounding Distances
Tim A. Hartmann and Stefan Lendl 55:1–55:14

On the Binary and Boolean Rank of Regular Matrices
Ishay Haviv and Michal Parnas 56:1–56:14

Gaps, Ambiguity, and Establishing Complexity-Class Containments via Iterative Constant-Setting
Lane A. Hemaspaandra, Mandar Juvekar, Arian Nadjimzadah, and Patrick A. Phillips 57:1–57:15

Independent Set Reconfiguration on Directed Graphs
Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, Masahiro Takahashi, and Kunihiro Wasa 58:1–58:15

Automating OBDD proofs is NP-hard
Dmitry Itsykson and Artur Riazanov 59:1–59:15

Not All Strangers Are the Same: The Impact of Tolerance in Schelling Games
Panagiotis Kanellopoulos, Maria Kyropoulou, and Alexandros A. Voudouris 60:1–60:14

On the Skolem Problem for Reversible Sequences
George Kenison 61:1–61:15

The Complexity of Computing Optimum Labelings for Temporal Connectivity
Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis 62:1–62:15

Beyond Value Iteration for Parity Games: Strategy Iteration with Universal Trees
Zhuan Khye Koh and Georg Loho 63:1–63:15

Countdown μ-Calculus
Jędrzej Kołodziejski and Bartek Klin 64:1–64:14

Rabbits Approximate, Cows Compute Exactly!
Balagopal Komarath, Anurag Pandey, and Nitin Saurabh 65:1–65:14

Finding 3-Swap-Optimal Independent Sets and Dominating Sets Is Hard
Christian Komusiewicz and Nils Morawietz 66:1–66:14

SAT-Based Circuit Local Improvement
Alexander S. Kulikov, Danila Pechenev, and Nikita Slezkin 67:1–67:15

Complexity of the Cluster Vertex Deletion Problem on H-Free Graphs
Hoang-Oanh Le and Van Bang Le 68:1–68:10

Reducing the Vertex Cover Number via Edge Contractions
Paloma T. Lima, Vinicius F. dos Santos, Ignasi Sau, Uéverton S. Souza, and Prafullkumar Tale 69:1–69:14

Generalized Bundled Fragments for First-Order Modal Logic
Mo Liu, Anantha Padmanabha, R. Ramanujam, and Yanjing Wang 70:1–70:14

Membership Problems in Finite Groups
Markus Lohrey, Andreas Rosowski, and Georg Zetzsche 71:1–71:16

Streaming Word Problems
Markus Lohrey and Lukas Lück 72:1–72:15

A Universal Skolem Set of Positive Lower Density
Florian Luca, Joël Ouaknine, and James Worrell 73:1–73:12

Learning Deterministic Visibly Pushdown Automata Under Accessible Stack
Jakub Michaliszyn and Jan Otop 74:1–74:16

Cohomology in Constraint Satisfaction and Structure Isomorphism
Adam Ó Conghaile 75:1–75:16

Deciding Emptiness for Constraint Automata on Strings with the Prefix and Suffix Order
Dominik Peteler and Karin Quaas 76:1–76:15

On Uniformization in the Full Binary Tree
Alexander Rabinovich 77:1–77:14

An Exact Algorithm for Knot-Free Vertex Deletion
M. S. Ramanujan, Abhishek Sahu, Saket Saurabh, and Shaily Verma 78:1–78:15

On Extended Boundary Sequences of Morphic and Sturmian Words
Michel Rigo, Manon Stipulanti, and Markus A. Whiteland 79:1–79:16

Higher-Order Causal Theories Are Models of BV-Logic
Will Simmons and Aleks Kissinger 80:1–80:14

Space-Bounded Unitary Quantum Computation with Postselection
Seiichiro Tani 81:1–81:15

Computing the Minimum Bottleneck Moving Spanning Tree
Haitao Wang and Yiming Zhao 82:1–82:15

Improved Approximation Algorithms for the Traveling Tournament Problem
Jingyang Zhao, Mingyu Xiao, and Chao Xu 83:1–83:15

Preface

The International Symposium on Mathematical Foundations of Computer Science (MFCS conference series) is a well-established venue for presenting research results in theoretical computer science. The broad scope of the conference encourages interactions between researchers who might not meet at more specialized venues. The first MFCS conference was organized in 1972 in Jabłonna (near Warsaw, Poland), and the conference has traditionally moved between the Czech Republic, Slovakia, and Poland. Since 2013, the conference has traveled around Europe. In 2022, MFCS celebrates its 50th anniversary in Vienna, Austria, on August 22–26.

We have an exciting program which includes 5 invited and 78 contributed talks. The invited speakers are Fedor V. Fomin (University of Bergen), Monika Henzinger (University of Vienna), Thomas Henzinger (IST Austria), Marta Kwiatkowska (University of Oxford), and Vijay Vazirani (University of California) on topics that reflect the broad scope of the conference. The latter invited talk is shared with the co-located 6th Workshop on Matching Under Preferences (MATCH-UP 2022).

The program committee of MFCS 2022 selected 78 papers out of 221 submissions, with the authors of the submitted papers representing over 35 countries. We want to express our deep gratitude to all the members of the program committee and reviewers for their extensive reports and discussions on the submissions' merits.

MFCS proceedings have been published in the Dagstuhl/LIPIcs series since 2016. We want to thank Michael Wagner and the LIPIcs team for their kind help and support. We also thank Doris Brazda and Jan Dreier and our team of student volunteers for helping with the local organization of the conference.

Stefan Szeider
Robert Ganian
Alexandra Silva

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Committee

Christoph Berkholz	Humboldt-Universität zu Berlin
René van Bevern	Huawei Technologies
Olaf Beyersdorff	Friedrich Schiller University Jena
Filippo Bonchi	Computer Science Department, University of Pisa
Andrei Bulatov	Simon Fraser University
Ugo Dal Lago	Università di Bologna and INRIA Sophia Antipolis
Laure Daviaud	City, University of London
Anuj Dawar	University of Cambridge
Stefan Felsner	TU Berlin
Celina Figueiredo	UFRJ
Nathanaël Fijalkow	CNRS, LaBRI, University of Bordeaux
Marie Fortin	University of Liverpool
Robert Ganian	TU Wien, co-chair
Petr Golovach	Department of Informatics, Bergen University
Gregory Gutin	Royal Holloway, University of London
Sara Kalvala	The University of Warwick
Sandra Kiefer	RWTH Aachen University
Eun Jung Kim	CNRS - Paris Dauphine
Dušan Knop	Czech Technical University in Prague
Martin Koutecky	Charles University in Prague
Martin Lange	University of Kassel
Massimo Lauria	Sapienza University of Rome
Karoliina Lehtinen	University of Liverpool
Meena Mahajan	The Institute of Mathematical Sciences, HBNI, Chennai
Konstantinos Mamouras	Rice University
Barnaby Martin	Durham University
George Mertzios	Durham University
Stefan Milius	FAU Erlangen
Neeldhara Misra	Indian Institute of Technology, Gandhinagar
Fabrizio Montecchiani	University of Perugia
Sebastian Ordyniak	The University of Sheffield
Sang-il Oum	Institute for Basic Science (IBS) and KAIST
Daniel Paulusma	Durham University
Daniela Petrisan	Université de Paris, IRIF
Michał Pilipczuk	University of Warsaw
Damien Pous	CNRS - ENS Lyon
Simon Puglisi	University of Helsinki
Paweł Rzążewski	Warsaw University of Technology
Alexandra Silva	Cornell University, co-chair
Friedrich Slivovsky	TU Wien
Ana Sokolova	University of Salzburg
Stefan Szeider	TU Wien, chair
Hellis Tamm	Tallinn University of Technology
Florian Zuleger	TU Wien

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Vittorio Bilò
Amaury Pouly
Yusuke Kobayashi
Argyrios Deligkas
Berenger Bramas
Alexandre Goy
Petr Savicky
Pranabendu Misra
Nicholas Georgiou
Andrew Ryzhikov
Jérémy Ledent
Kim-Manuel Klein
Aurélie Lagoutte
Yi Tang
Marta Piecyk
Giacomo Ortali
Nai-Hui Chia
Christof Löding
Kostia Chardonnet
Laurent Bienvenu
Philipp Kindermann
Raul Lopes
Joanna Ochremiak
Oleg Verbitsky
Michał Skrzypczak
Pawel Idziak
Argyrios Deligkas
Archontia Giannopoulou
Luc Spachmann
Rachmad Vidya Wicaksana Putra
Sujoy Bhore
Elvira Mayordomo
Jefferson Elbert Simões
Simon Perdrix
Jacob Focke
Alessandro Di Giorgio
Piotr Faliszewski
Lionel Pournin
Jérémy Ledent
Danny Hermelin
Niel De Beaudrap
Nina Klobas
Lukasz Kowalik
Hendrik Molter
Alexander Irribarra
Théo Matricon
Yongjie Yang
Vesa Halava
Lehilton L. C. Pedrosa
Sushmita Gupta
Henning Urbat
Corto Mascle
Guillaume Theyssier
Krzysztof Sornat
Ziyuan Gao
Nathan Lhote
K. S. Thejaswini
Rosiane de Freitas
Prahladh Harsha
Chia-Wei Lee
Suguru Tamaki
Anand Natarajan
Dorian Rudolph
Augustin Vanrietvelde
Andrea Turrini
Agnes Schleitzer
Martin Gronemann
Akanksha Agrawal
Victor Campos
Daniel Neuen
Michael Kompatscher
Miki Hermann
Rafael Schouery
Gaëtan Staquet
Jordi Levy
Naoyuki Kamiyama
Rodrigo Silveira
Verónica Becher
Yassine Hamoudi
Artur Jeż
Tuukka Korhonen
Ilario Bonacina
Andrzej Kaczmarczyk
Joe Sawada
Etienne Lozes
Rene Sitters
John van de Wetering
Tim Hoffmann
Kathrin Hanauer
Marek Sokołowski
Tuukka Korhonen
Giacomo Paesani
Pierre Ohlmann
S. Akshay
Frank Sommer
Quentin Bramas
Valeria de Paiva
Igor Potapov
Silvio Capobianco
Bhaskar Ray Chaudhury
Dmitriy Traytel
Florian Bruse
Alexandra Lassota
Mika Göös
Huck Bennett
Josefran Bastos
Argyrios Deligkas
Atul Singh Arora
Dominique Perrin
Cole Comfort
Marius Zimand
Garth Isaak
Eduard Eiben
Jean-Guy Mailly
Pierre Pradic
Hugo Nobrega
Catarina Carvalho
Benny Kimelfeld
O-Joung Kwon
Hendrik Maarand
Florent Bréhard
Rohit Gurjar
Trent Marbach
Alban Goupil
Luis Kowada
Norbert Hundeshagen
Pawel Gawrychowski
Chandan Saha
Sanjukta Roy
Martin Zimmermann
Gaurav Rattan
Arne Meier
Kesha Hietala
Tomáš Peitl
Montserrat Hermo
Luca Castelli Aleardi
Hung Hoang
Lorenzo Balzotti

Kirill Simonov
Pierre Bergé
Spyros Angelopoulos
Nello Blaser
Olivier Laurent
Bartek Klin
Agnes Schleitzer
Linus Boes
Brian Cloteaux
Wiktor Zuba
Bora Ucar
Sylvain Perifel
Ferdinand Ihringer
Jesper Nederlof
Hubie Chen
Tim A. Hartmann
Oxana Tsidulko
Petra Wolf
Stefan Funke
Zachary Remscrim
Niclas Boehmer
Yann Disser
Julian Mestre
Marcus Michelen
Will Perkins
Jeffrey Shallit
Sankardeep Chakraborty
Christian Komusiewicz
Johannes Zink
Benjamin Aram Berendsohn
Rudini Sampaio
Peter Kutas
Spyros Kontogiannis
Thorsten Wißmann
Keisuke Nakano
Paolo Milazzo
Colin Riba
Andrea Corradini
Paolo Penna
K. S. Thejaswini
Richard Blute
Soichiro Fujii
Noleen Köhler
Yota Otachi
Sudeshna Kolay
Henning Urbat
Joel Ouaknine
Jesse Beisegel
Massimo Bartoletti
Konrad K. Dabrowski
Kim S. Larsen
Guido Sciavicco
Federico Olimpieri
Susanna F. de Rezende
Yogesh Dahiya
Jörg Rothe
Daniel Posner
Václav Blažej
David Peleg
Reinhard Diestel
Sourav Chakraborty
Klaus Kriegel
Matthias Bentert
Achim Blumensath
Andrea Marino
Antonio Casares
Thomas Bläsius
Abuzer Yakaryilmaz
George Skretas
Arnaud Labourel
Jan Matyáš Křišťan
Zongchen Chen
Arseny Shur
Miriam Backens
Erkan Narmanli
Pascal Lenzner
Giacomo Ortali
Manuel Cáceres
Ondrej Suchy
Judith Clymo
Claudson Bornstein
Marino Miculan
Aloïs Rosset
Davide Trotta
Thekla Hamm
Ian McQuillan
Stefano Leucci
Laszlo Kozma
Lutz Straßburger
Richard Garner
Siani Smith
Tomasz Krawczyk
Joao Paixao
Tom Hirschowitz
Benjamin Böhm
Adam Polak
Luca Grilli
Shahin Kamali
Léo Exibard
Francesco Antonio Genco
Vincenzo Ciancia
Ian Mertz
Hartmut Klauck
Palash Dey
Joe Sawada
Labbe Sebastien
Alessia Antelmi
Solveig Klepper
Karl Wimmer
Arnaud Durand
Laurent Gourves
Warut Suksompong
Nike Dattani
Maurizio Murgia
Dominik Engel
Daniel Neuen
Yuichi Sudo
Matthias Koeppe
Serge Gaspers
Mark Jerrum
Florian Bruse
Niel De Beaudrap
Jari J.H. de Kroon
Brendan Lucier
Felix Schröder
Tomáš Valla
Jurij Volcic
Tomáš Masařík
Rishiraj Bhattacharyya
Takeshi Tsukada
Francesco Gavazzo
Lutz Schröder
Sourabh Palande
Paolo Franciosa
John Fearnley
Bernard Lidicky
Chad Nester
Stanislav Živný
Alessandra Tappini
Dirk Sudholt
Maximilian Pfister
Konstanty Junosza-Szaniawski
Kirill Simonov
Dan Marsden

Cole Comfort
Pavel Smirnov
Matthias Bentert
Jan Obdrzalek
Kenjiro Takazawa
Alexander Knop
Ritam Raha
Yann Strozecki
Szymon Toruńczyk
Igor Potapov
Brink Van Der Merwe
Patrick Landwehr
Leon Bohn
Jannik Peters
Alexandre Vigny
Yixin Cao
Jocelyn Thiebaut
Carolin Rehs
Svante Janson
Théo Matricon
Hassan Ashtiani
Xiao Mao
Sergio Cabello
Yuval Filmus
Marin Bougeret
Nina Klobas
Aniket Murhekar
Franz Baader
Fernando Hugo Cunha Dias
Shaohua Li
Supanat Kamtue
Nils Vortmeier
Alessandro Simon
Kazuhiro Kurita
Sayan Bandyapadhyay
Robin Piedeleu
Eduard Eiben
Emilio Cruciani
Darren Strash
Manuel Aprile
Mateusz Skomra
Rui Soares Barbosa
Heribert Vollmer
Stéphane Demri
Martin Dyer
Stefan Mengel
Hajo Broersma
C Ramya

Jamie Vicary
Minati De
Rémi Morvan
Andrea Jiménez
Günter Rote
Ben Lee Volk
Michaël Cadilhac
Jane Lange
Abdul Ghani
Suguru Tamaki
Marco Sälzer
Emilie Charlier
Marco Sälzer
Alina Ostafe
Jacobo Torán
Eric Alsmann
Felicia Lucke
Šimon Schierreich
Niclas Boehmer
Maurice Herwig
Idan Attias
Diptapriyo Majumdar
Dominik Kaaser
Marc Vinyals
Huib Donkers
Jérémy Ledent
Igor Potapov
Manfred Schmidt-Schauß
André Nichterlein
Guillaume Ducoffe
Corentin Barloy
Anatole Dahan
Bo Liang
Benjamin Gras
Anaëlle Wilczynski
Peter Winkler
Lars Jaffke
Michał Dębski
Marc Roth
Shenwei Huang
Blaise Genest
Matteo Riondato
Florian Lehner
Sriram Sankaranarayanan
David Richerby
Giulio Malavolta
Jean-Guillaume Dumas
Michal Garlik

Arthur Jaquard
Nicole Wein
Yijia Chen
Pascal Kunz
N. V. Vinodchandran
Prerona Chatterjee
Lê Thành Dũng Nguyễn
Édouard Bonnet
Olivier Bournez
Ramprasad Saptharishi
Denis Kuperberg
Radek Hušek
Martin Bullinger
Amer Mouawad
Tim A. Hartmann
Benedikt Bollig
Nina Klobas
Arne Winterhof
Andrzej Kaczmarczyk
Guillaume Ducoffe
Santiago Escobar
Bin Sheng
Arnaud Casteigts
Huib Donkers
Ondrej Suchy
Jeroen Zuiddam
Evgeniy Zorin
Michael Lampis
Nidhi Purohit
Tomohiro Koana
Benjamin Steinberg
Bruno Guillon
John Lapinskas
Till Fluschnik
Dario Stein
Robert Bredereck
Nikhil Kumar
David Kutner
Sandro Roch
Olivier Bournez
Rémi Morvan
Yinzhan Xu
Navid Talebanfard
Gabriela Jeronimo
Libor Barto
Brink Van Der Merwe
Marvin Künnemann
Sébastien Tavenas

Mingyu Xiao
Georg Struth
Nathaniel Harms
Uéverton Souza
Paul Brunet
Akbar Rafiey
Ondrej Suchy
Andrea Corradini
Jungho Ahn
Louis Esperet
Bruno Courcelle
Florin Manea
David Cerna
Zacharias Heinrich
Alexandre Goy
Loris Marchal
David Kutner
Noleen Köhler
Édouard Bonnet
Dina Sokol
Andreas Emil Feldmann
Feodor Dragan
Maaike Zwart

Long Cycles in Graphs: Extremal Combinatorics Meets Parameterized Algorithms

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway

Petr A. Golovach
Department of Informatics, University of Bergen, Norway

Danil Sagunov
St. Petersburg Department of V.A. Steklov Institute of Mathematics, Russia

Kirill Simonov
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

We discuss recent algorithmic extensions of two classic results of extremal combinatorics about long paths in graphs. First, the theorem of Dirac from 1952 asserts that a 2-connected graph G with the minimum vertex degree $d > 1$, is either Hamiltonian or contains a cycle of length at least $2d$. Second, the theorem of Erdős-Gallai from 1959, states that a graph G with the average vertex degree $D > 1$, contains a cycle of length at least D. The proofs of these theorems are constructive, they provide polynomial-time algorithms constructing cycles of lengths $2d$ and D. We extend these algorithmic results by showing that each of the problems, to decide whether a 2-connected graph contains a cycle of length at least $2d + k$ or of a cycle of length at least $D + k$, is fixed-parameter tractable parameterized by k.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Longest path, longest cycle, fixed-parameter tractability, above guarantee parameterization, average degree, dense graph, Dirac theorem, Erdős-Gallai theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.1

Category Invited Talk

Funding *Fedor V. Fomin*: The research received funding from the Research Council of Norway via the project BWCA (grant no. 314528).
Danil Sagunov: This project was supported by Leonhard Euler International Mathematical Institute in Saint Petersburg (agreement no. 075–15–2019–1620).

1 Introduction

The two fundamental theories from graph theory guarantee the existence of long cycles in dense graphs. The first theorem is Dirac's theorem from 1952.

▶ **Theorem 1** (Dirac [2, Theorem 4]). *Every n-vertex 2-connected undirected graph G with minimum vertex degree $\delta(G) \geq 2$, contains a cycle with at least $\min\{2\delta(G), n\}$ vertices.*

The second theorem from 1959 is due to Erdős and Gallai [3].

▶ **Theorem 2** (Erdős and Gallai [3]). *Every undirected graph with n vertices and more than $\frac{1}{2}(n-1)\ell$ edges ($\ell \geq 2$) contains a cycle of length at least $\ell + 1$.*

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 1; pp. 1:1–1:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The proofs of both theorems are constructive, in the sense that they provide polynomial-time algorithms constructing cycles of lengths $\min\{2\delta(G), n\}$ and $\ell + 1$. This brings us to a natural and "innocent" question: is it possible to extend the algorithms provided by Theorems 1 and 2 by a "tiny" bit? For example, for an integer $k \geq 1$, *is there a polynomial time algorithm comuting a cycle of length at least $2\delta(G) + k$?* Or, *is it possible to identify in polynomial time whether a graph with $\frac{1}{2}(n-1)\ell$ edges contains a cycle of length at least $\ell + k$?*

The methods developed in the extremal Hamiltonian graph theory do not answer such questions. The combinatorial bounds in Theorems 1 and 2 are known to be sharp; that is, there exist graphs that have no cycles of length at least $\min\{2\delta(G) + 1, n\}$ or $\ell + 2$. Since the extremal graph theory studies the existence of a cycle under certain conditions, such type of questions are beyond its applicability. On the other hand, the existing methods of parameterized complexity, see e.g. [1], do not seem to be much of use here either. Such algorithms compute a cycle of length at least k in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, which in our case is $2^{\mathcal{O}(\delta(G))} \cdot n^{\mathcal{O}(1)}$. Hence when $\delta(G)$ is, for example, at least $n^{1/100}$, these algorithms do not run in polynomial time.

We answer both questions affirmatively and in a much more general way. Our first theorem, this theorem appears in [4], implies that in polynomial time one can decide whether G contains a cycle of length at least $2\delta(G - B) + k$ for $B \subseteq V(G)$ and $k \geq 0$ as long as $k + |B| \in \mathcal{O}(\log n)$. (We denote by $G - B$ the induced subgraph of G obtained by removing vertices of B.) To state our result more precisely, we define the following problem.

Long Dirac Cycle parameterized by $k + |B|$

Input: Graph G with vertex set $B \subseteq V(G)$ and integer $k \geq 0$.
Task: Decide whether G contains a cycle of length at least $\min\{2\delta(G - B), |V(G)| - |B|\} + k$.

In the definition of Long Dirac Cycle we use the minimum of two values for the following reason. The question whether an n-vertex graph G contains a cycle of length at least $2\delta(G - B) + k$ is meaningful only for $\delta(G - B) \leq n/2$. Indeed, for $\delta(G - B) > n/2$, G does not contain a cycle of length at least $2\delta(G - B) + k > n$. However, even when $\delta(G - B) > n/2$, deciding whether G is Hamiltonian, is still very intriguing. By taking the minimum of the two values, we capture both interesting situations.

▶ **Theorem 3.** *On an n-vertex 2-connected graph G, Long Dirac Cycle is solvable in time $2^{\mathcal{O}(k+|B|)} \cdot n^{\mathcal{O}(1)}$.*

In other words, Long Dirac Cycle is fixed-parameter tractable parameterized by $k + |B|$ and the dependence on the parameters is single-exponential. This dependence is asymptotically optimal up to the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [6]. Solving Long Dirac Cycle in time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ even with $B = \emptyset$ yields recognizing in time $2^{o(n)}$ whether a graph is Hamiltonian. A subexponential algorithm deciding Hamiltonicity would fail ETH. We show that solving Long Dirac Cycle in time $2^{o(|B|)} \cdot n^{\mathcal{O}(1)}$ even for $k = 1$ would contradict ETH as well. It is also NP-complete to decide whether a 2-connected graph G has a cycle of length at least $(2 + \varepsilon)\delta(G)$ for any $\varepsilon > 0$.

The 2-connectivity requirement in the statement of the theorem is important – without it Long Dirac Cycle is already NP-complete for $k = |B| = 0$. Indeed, for an n-vertex graph G construct a graph H by attaching to each vertex of G a clique of size $n/2$. Then H has a cycle of length at least $2\delta(H) \geq n$ if and only if G is Hamiltonian.

Our second theorem, that appears in [5], provides an algorithmic extension of the Erdős-Gallai theorem: A fixed-parameter tractable (FPT) algorithm with parameter k, that decides whether the circumference (the length of the longest cycle) of a graph is at least $\ell + k$. To state our result formally, we need a few definitions. For an undirected graph G with n vertices and m edges, we define $\ell_{EG}(G) = \frac{2m}{n-1}$. Then by the Erdős-Gallai theorem, G always has a cycle of length at least $\ell_{EG}(G)$ if $\ell_{EG}(G) > 2$. The parameter $\ell_{EG}(G)$ is closely related to the *average* degree of G, $\mathsf{ad}(G) = \frac{2m}{n}$. It is easy to see that for every graph G with at least two vertices, $\ell_{EG}(G) - 1 \leq \mathsf{ad}(G) < \ell_{EG}(G)$.

The *maximum average degree* $\mathsf{mad}(G)$ is the maximum value of $\mathsf{ad}(H)$ taken over all induced subgraphs H of G. Note that $\mathsf{ad}(G) \leq \mathsf{mad}(G)$ and $\mathsf{mad}(G) - \mathsf{ad}(G)$ may be arbitrary large. By Theorem 2, we have that if $\mathsf{ad}(G) \geq 2$, then G has a cycle of length at least $\mathsf{ad}(G)$ and, furthermore, if $\mathsf{mad}(G) \geq 2$, then there is a cycle of length at least $\mathsf{mad}(G)$. Based on this guarantee, we define the following problem.

Longest Cycle Above MAD

Input: A graph G on n vertices and an integer $k \geq 0$.
Task: Decide whether G contains a cycle of length at least $\mathsf{mad}(G) + k$.

Our main result is that this problem is FPT parameterized by k. More precisely, we show the following.

▶ **Theorem 4.** *Longest Cycle Above MAD can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ on 2-connected graphs.*

While Theorems 1 and 2 concern decision problems, their proofs may be adapted to produce desired cycles, if they exist. We underline this because the standard construction of a long cycle that for every $e \in E(G)$ invokes the decision algorithm on $G - e$, does not work in our case, as edge deletions decrease the average degree of a graph.

We also briefly discuss the ideas behind the proofs of both theorems that are based on an interplay between extremal combinatorics and parameterized algorithms. We develop a new graph decomposition that we call *Dirac decomposition* and then show how to use this decomposition algorithmically. Dirac decomposition is defined for a cycle C in a 2-connected graph G. Let C be a cycle of length less than $2\delta(G) + k$. Informally, the components of Dirac decomposition are connected components in $G - V(C)$. Since G is 2-connected, we can reach C by a path starting in such a component in G. One of the essential properties of Dirac decomposition is a limited number of vertices in $V(C)$ that have neighbors outside of C. In fact, we can choose two short paths P_1 and P_2 in C (and short means that their total length is of order k) such that all connections between connected components of $G - V(C)$ and C go through $V(P_1) \cup V(P_2)$. The second important property is that each connected component of $G - (V(P_1) \cup V(P_2))$ is connected with P_i in G in a very restricted way: The maximum matching size between its vertex set and the vertex set of P_i is at most one. Dirac decomposition appears to be very useful for algorithmic purposes. For a cycle C, given a Dirac decomposition for C, in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ we either solve the problem or succeed in enlarging C.

To apply Dirac decomposition, we also design a polynomial time that (except some "extremal" cases) we can either (a) enlarge the cycle C, or (b) compute a vertex cover of G of size at most $\delta(G) + 2k$, or (c) compute a Dirac decomposition. In cases (a) and (c), we can proceed iteratively. For the case (b) we need another algorithm that solves the problem in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

References

1 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

2 Gabriel Andrew Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc. (3)*, 2:69–81, 1952.

3 Paul Erdős and Tibor Gallai. On maximal paths and circuits of graphs. *Acta Math. Acad. Sci. Hungar*, 10:337–356 (unbound insert), 1959.

4 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Algorithmic extensions of Dirac's theorem. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2022)*, pages 931–950. SIAM, 2022. doi:10.1137/1.9781611977073.20.

5 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Longest cycle above Erdős-Gallai bound. *CoRR*, abs/2202.03061, 2022. arXiv:2202.03061.

6 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.

Modern Dynamic Data Structures

Monika Henzinger
University of Vienna, Department of Computer Science, Vienna, Austria

Abstract

We give an overview of differentially private dynamic data structure, aka differentially private algorithms under continual release.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Differential privacy, data structures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.2

Category Invited Talk

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024

1 Extended Abstract

In the past the main goal when designing data structures was to achieve optimal time per operation and optimal space. However, in recent years new applications have lead to new requirements for data structures, such as differential privacy or fairness.

In this talk I will limit myself to the first topic, namely differential privacy. Since its invention in 2006 by Dwork, McSherry, Nissim, and Smith [3] differentially private algorithms and (static) data structures have been designed for many combinatorial problems (see e.g. [5] for a book on the topic). However, very little work has been done for *dynamic* data structures. A *dynamic data structure* is a data structure that supports not only *query* operations to the stored data, but also *update* operations, such as insertions and/or deletions. In the differentially privacy research community such data structures are frequently called data structures *in the continual release* (or *continual observation*) *model.*

The problem of *binary counting* the number of 1s in a binary sequence is equivalent to a dynamic data structure that supports the *AppendBit* operation and that outputs the number of 1s in the current sequence after each *AppendBit* operation. This problem has been well-studied in the differentially private setting [4, 1, 8, 2, 7], including a version that outputs a weighted average of the bits in the sequence so far [7]. Another extension of this problem leads to the *MaxSum* and the *SumSelect* problem: Assume the input is a sequence of d-dimensional binary vectors such that the t-th vector is denoted by b_t. The goal of the *SumSelect* problem is to output the value of the coordinate i such that $i = \text{argmax}_i \sum_t b_t[i]$, the goal of *MaxSum* is to output $\max_i \sum_t b_t[i]$. These problems were studied in the continual release model in [9]: Differentially private partially *dynamic graph algorithms* were also analyzed in the continual release setting [10, 6].

We will describe the algorithm of [7] in detail and explain why it is superior to the previous solutions for binary counting under continual observation.

References

1 T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26:1–26:24, 2011. doi:10.1145/2043621.2043626.

2 Sergey Denisov, Brendan McMahan, Keith Rush, Adam Smith, and Abhradeep Thakurta. Improved differential privacy for sgd via optimal private linear operators on adaptive streams. *arXiv preprint*, 2022. arXiv:2202.08312.

3 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*, pages 265–284, 2006.

4 Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proc. of the Forty-Second ACM Symp. on Theory of Computing (STOC'10)*, pages 715–724, 2010.

5 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi:10.1561/0400000042.

6 Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost. Differentially private algorithms for graphs under continual observation. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, 2021.

7 Hendrik Fichtenberger, Monika Henzinger, and Jalaj Upadhyay. Constant matters: Fine-grained complexity of differentially private continual observation using completely bounded norms. *arXiv preprint*, 2022. arXiv:2202.11205.

8 James Honaker. Efficient use of differentially private binary trees. *Theory and Practice of Differential Privacy (TPDP 2015), London, UK*, 2015.

9 Palak Jain, Sofya Raskhodnikova, Satchit Sivakumar, and Adam Smith. The price of differential privacy under continual observation. *arXiv preprint*, 2021. arXiv:2112.00828.

10 Shuang Song, Susan Little, Sanjay Mehta, Staal Vinterbo, and Kamalika Chaudhuri. Differentially private continual release of graph statistics. *arXiv preprint*, 2018. arXiv:1809.02575.

An Updated Survey of Bidding Games on Graphs

Guy Avni
University of Haifa, Israel

Thomas A. Henzinger
IST Austria, Austria

Abstract

A graph game is a two-player zero-sum game in which the players move a token throughout a graph to produce an infinite path, which determines the winner or payoff of the game. In *bidding games*, both players have budgets, and in each turn, we hold an "auction" (bidding) to determine which player moves the token. In this survey, we consider several bidding mechanisms and their effect on the properties of the game. Specifically, bidding games, and in particular bidding games of infinite duration, have an intriguing equivalence with *random-turn* games in which in each turn, the player who moves is chosen randomly. We summarize how minor changes in the bidding mechanism lead to unexpected differences in the equivalence with random-turn games.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory; Theory of computation → Formal languages and automata theory

Keywords and phrases Bidding games, Richman bidding, poorman bidding, mean-payoff, parity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.3

Category Invited Talk

Related Version This paper is a short, updated version of [3]. In particular, results from [7] have been added.
Full Version: https://doi.org/10.4230/LIPIcs.CONCUR.2020.2

Funding *Guy Avni*: Work partially supported by the Israel Science Foundation, ISF grant agreement no 1679/21.
Thomas A. Henzinger: This work was supported in part by the ERC-2020-AdG 101020093.

Acknowledgements We would like to thank all our collaborators Milad Aghajohari, Ventsislav Chonev, Rasmus Ibsen-Jensen, Ismäel Jecker, Petr Novotný, Josef Tkadlec, and Đorđe Žikelić; we hope the collaboration was as fun and meaningful for you as it was for us.

1 Introduction

Games on graphs are a central class of games in formal verification [2] and have deep connections to foundations of logic [17]. They have numerous applications including reactive synthesis [16], verification [10], and reasoning about multi-agent systems [1]. Theoretically, graph games give rise to interesting and challenging problems. For example, solving parity games is a rare problem that is in NP and coNP [11], not known to be in P, and for which a quasi-polynomial algorithm was only recently discovered [8].

A graph game proceeds as follows. We place a token on one of the vertices of a graph and allow the players to move it to produce an infinite path that determines the winner or payoff of the game. Several *modes of moving* the token have been studied [2], and the most popular is *turn-based* graph games in which the players alternate turns in moving the token. We study the *bidding* mode of moving [13, 12]: players have budgets and in each turn, an "auction" (bidding) determines which player moves the token. Bidding games are a class of concurrent graph games [1]. They combine graph games with auctions, a central topic of research in algorithmic game theory (e.g., [14]).

Objectives. We stress that bidding is the mode of moving the token and it is orthogonal to the players' objectives. We consider the qualitative objectives reachability and parity, and the quantitative objective mean-payoff.

2 Bidding Mechanisms

In all the bidding mechanisms that we consider, in each turn, both players simultaneously submit bids that do not exceed their available budget, and the higher bidder "wins" the bidding and moves the token. The bidding mechanisms differ in two orthogonal properties:

1. Who pays: in *first-price* bidding only the higher bidder pays; in *all-pay* bidding both players pay their bids.
2. Where are the bids paid: in *Richman* bidding (named after David Richman), bids are paid to the opponent; in *poorman* bidding bids are paid to the "bank", thus the budget is lost.

▶ Remark 1. A well-known auction mechanism is *second-price bidding* in which the highest bidder pays the second highest bid. We point out that bidding games under first- and second-price bidding coincide, since the players in second-price bidding can follow the same optimal strategies they use in first-price bidding to guarantee the same values.

The central quantity in bidding games regards the ratio between the two players' budget, formally defined as follows.

▶ **Definition 2** (Budget ratio). *Assuming Player i's budget is B_i, for $i \in \{0, 1\}$, then Player 1's ratio is $B_1/(B_1 + B_2)$.*

Random-turn games. A *random-turn game* [15] is similar to a bidding game only that instead of bidding for moving, in each turn, we decide which player moves according to a (possibly biased) coin toss. For a bidding game $\mathcal{G}$ and $p \in [0, 1]$, we denote by $\mathsf{RT}(\mathcal{G}, p)$ the random-turn game that is obtained from $\mathcal{G}$ using a coin with bias p. Formally, random-turn games are a subclass of stochastic games [9]. To obtain a random-turn game from $\mathcal{G}$, we proceed as follows. For every vertex v in $\mathcal{G}$, we add two vertices v_{Max} and v_{Min} owned by the respective players. To simulate the coin toss at v, we add probabilistic edges from v to v_{Max} with probability p and to v_{Min} with probability $1 - p$. To simulate the choice of the player who wins the bidding, we add edges from both v_{Max} and v_{Min} to every u that is a neighbor of v in $\mathcal{G}$. The objective of $\mathsf{RT}(\mathcal{G}, p)$ coincides with the objective of $\mathcal{G}$.

3 Qualitative bidding games

The main question considered in qualitative bidding games regards the threshold budgets, which intuitively represent a necessary and sufficient initial budget ratio that suffices for winning the game. Formally,

▶ **Definition 3** (Threshold ratio). *Consider a qualitative bidding game $\mathcal{G}$ and a vertex v in $\mathcal{G}$. The* threshold ratio *in v, denoted $\mathit{Th}(v)$, is such that:*

- *If Player 1's initial ratio is strictly greater than $\mathit{Th}(v)$, he has a winning strategy from v.*
- *If Player 1's initial ratio is strictly less than $\mathit{Th}(v)$, Player 2 has a winning strategy from v.*

3.1 Reachability first-price bidding games

The focus in [13, 12] was on first-price reachability bidding games. An intriguing equivalence between reachability games with first-price Richman bidding with random-turn games, and, interestingly, only for this bidding mechanism. We formally state the result below and illustrate it in Fig. 1.

▶ **Theorem 4** ([13, 12]). *Consider a reachability bidding game with target states t_1 and t_2. Threshold ratios exist. Moreover, $\mathit{Th}(t_1) = 0$ and $\mathit{Th}(t_2) = 1$, and for any other vertex v, let v^- and v^+ be the neighbors of v such that $\mathit{Th}(v^-) \leq \mathit{Th}(v') \leq \mathit{Th}(v^+)$, for every neighbor v' of v. Then:*

- *Richman bidding: $\mathit{Th}(v) = \frac{1}{2}(\mathit{Th}(v^+) + \mathit{Th}(v^-))$ and $\mathit{Th}(v)$ coincides with the value of the vertex v in the random-turn game $RT(\mathcal{G}, 0.5)$.*
- *Poorman bidding: $\mathit{Th}(v) = \frac{\mathit{Th}(v^+)}{1-\mathit{Th}(v^-)+\mathit{Th}(v^+)}$.*

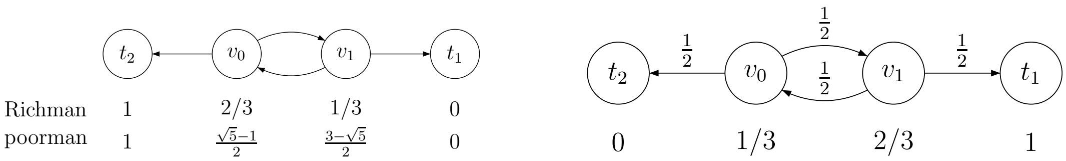


Figure 1 Left: A reachability bidding game $\mathcal{G}$ with threshold ratios, under first-price Richman and poorman bidding.
Right: The (simplified) unbiased random-turn game $\mathsf{RT}(\mathcal{G}, 0.5)$ is a Markov chain. The value of a vertex is the probability of reaching t_1. Note that under Richman bidding, for every vertex v, we have $\mathtt{Th}(\mathcal{G}, v) = 1 - val\big(\mathsf{RT}(\mathcal{G}, 0.5), v\big)$. Moreover, under poorman bidding, ratios are irrational thus such an equivalence is unlikely to exist.

3.2 Reachability all-pay bidding games

In [6], reachability games under all-pay bidding are shown to be technically much more challenging than under first-price bidding. Some positive results are shown; namely, an approximation algorithm based on discretization in games played on DAGs and results on the threshold for surely winning. Most results, however, are negative and fundamental problems, including proving that the value of the game always exists, remain open.

▶ **Theorem 5** ([6]). *Optimal strategies in reachability all-pay poorman bidding are sometimes mixed and draw bids from infinite-support distributions.*

3.3 Parity bidding games

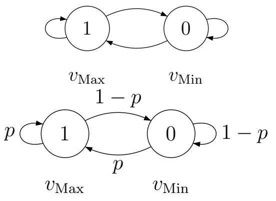
We state a key property of parity bidding games played on strongly-connected graphs.

▶ **Theorem 6** ([4, 5, 7]). *Consider a parity game $\mathcal{G}$ played on a strongly-connected graph in which the highest parity index is odd. Under first-price Richman and poorman bidding, the threshold ratios are 0 in all the vertices; namely, Player 1 can win with any positive initial budget. Under all-pay Richman and poorman bidding, with any positive initial ratio, Player 1 has a mixed strategy that guarantees satisfying the parity objective with probability 1.*

For first-price bidding games, Theorem 6 gives rise to the following simple reduction from parity to reachability bidding games. Given a parity game $\mathcal{G}$, first reason about the bottom strongly-connected components and classify them into those that are "winning" for Player 1 and those that are “winning” for Player 2. Then, construct a reachability bidding game in which each player’s goal is to force the game to a winning bottom strongly-connected component. A similar reduction applies to all-pay bidding, however reachability games under those bidding mechanisms are not yet understood.

4 Mean-Payoff Bidding Games

In this section we consider mean-payoff games played on strongly-connected graphs. We show intricate equivalences between mean-payoff bidding games and random-turn games.



	Richman		poorman	
First-price	$\mathsf{RT}(\mathcal{G}, \frac{1}{2})$ [4]		$\mathsf{RT}(\mathcal{G}, r)$ [5]	
All-pay	Pure	Mixed	Pure	Mixed
[7]	$\mathsf{RT}(\mathcal{G}, 0)$	$\mathsf{RT}(\mathcal{G}, \frac{1}{2})$	$\mathsf{RT}(\mathcal{G}, \frac{2r-1}{r})$	$\mathsf{RT}(\mathcal{G}, \frac{3r-1}{r})$

Figure 2 Left: On top, the mean-payoff bidding game $\mathcal{G}_{\bowtie}$. The payoff of a player in $\mathcal{G}_{\bowtie}$ is the long-run ratio of the biddings won. On bottom, for $p \in [0, 1]$, the (simplified) random-turn game $\mathsf{RT}(\mathcal{G}_{\bowtie}, p)$ is a weighted Markov chain. The expected payoff in $\mathsf{RT}(\mathcal{G}_{\bowtie}, p)$ is p; we expect that a random walk stays ratio p of the time in v_{Max}.
Right: The equivalence relates the optimal payoff in a strongly-connected mean-payoff game with the expected payoff in a random-turn game, where for all-pay poorman bidding we omit the cases of $r \leq 0.5$.

Mean-payoff value. Each play of a mean-payoff game has a payoff, which is Player 1’s (Max) reward and Player 2’s (Min) cost. We illustrate the definition of the mean-payoff objective. Consider the game $\mathcal{G}_{\bowtie}$ that is depicted in the top left of Fig. 2. It models the following setting. Max and Min represent two advertisers. In each day, an auctioneer holds an auction to determine which ad shows that day. Max’s goal is to maximize the payoff, which coincides with the number of days that his ad appears in a very long time (say, a year). Alternatively, the payoff in $\mathcal{G}_{\bowtie}$ can be seen as the ratio of the biddings that Max wins in the long run. Formally, the payoff of an infinite sequence of weights $w_1, w_2, \ldots$ is $\liminf_{n \to \infty} \frac{1}{n} \sum_{1 \leq i \leq n} w_i$.

▶ **Definition 7** (Mean-payoff value in bidding games). *Consider a strongly-connected mean-payoff bidding game $\mathcal{G}$ and a budget ratio $r \in (0, 1)$. The* mean-payoff value *of $\mathcal{G}$ w.r.t. r, denoted $\mathsf{MP}(\mathcal{G}, r)$, is $c \in \mathbb{R}$ if independent of the initial vertex,*

- *when Max’s initial ratio exceeds r, he has a strategy that guarantees a payoff of $c - \varepsilon$, for every $\varepsilon > 0$, and*
- *Max cannot do better: with a ratio that exceeds $1 - r$, Min can guarantee a payoff of at most $c + \varepsilon$, for every $\varepsilon > 0$.*

Similarly, we use asMP to denote the almost-sure value, which is defined as the payoff that Max can guarantee with a mixed strategy with probability 1.

Consider a strongly-connected mean-payoff game $\mathcal{G}$ and $p \in [0, 1]$. Recall that $\mathsf{RT}(\mathcal{G}, p)$ denotes the random-turn game that is constructed from $\mathcal{G}$ in which in each turn, Max moves the token with probability p. We denote by $\mathsf{MP}(\mathsf{RT}(\mathcal{G}, p))$ the *mean-payoff value* of $\mathsf{RT}(\mathcal{G}, p)$, which is defined as the expected payoff when both players play optimally, and it is well-known to exist in stochastic games.

▶ **Theorem 8.** *Consider a strongly-connected mean-payoff game $\mathcal{G}$ and a ratio $r \in (0,1)$.*

- *First-price Richman bidding [4]: For all $r \in (0,1)$, we have $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, 0.5)\big)$.*
- *First-price poorman bidding [5]: $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, r)\big)$.*
- *All-pay Richman bidding [7]:*
  - *Under pure strategies, $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, 0)\big)$.*
  - *Under mixed strategies, $\mathit{asMP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, 0.5)\big)$.*
- *All-pay poorman bidding [7]:*
  - *Under pure strategies, if $r > 0.5$, then $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, \frac{2r-1}{r})\big)$, and if $r \leq 0.5$, then $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, 0)\big)$.*
  - *Under mixed strategies, if $r > 0.5$, then $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, \frac{3r-1}{r})\big)$, and if $r \leq 0.5$, then $\mathit{MP}(\mathcal{G}, r) = \mathit{MP}\big(\mathit{RT}(\mathcal{G}, \frac{1-r}{r})\big)$.*

In the following example, we illustrate the results stated in Theorem 8 on $\mathcal{G}_{\bowtie}$ that is depicted in Fig. 2.

▶ **Example 9.** First, let $p \in (0,1)$. The mean-payoff value of $\mathsf{RT}(\mathcal{G}_{\bowtie}, p)$ is p since intuitively, a random walk is expected to stay portion p in vertex v_{Max}.

We start with first-price bidding. Here, optimal strategies are pure (deterministic). Under Richman bidding, the initial ratio does not matter and the optimal payoff is 0.5, matching the mean-payoff value of $\mathsf{RT}(\mathcal{G}_{\bowtie}, 0.5)$. That is, for every $\varepsilon > 0$, Max can guarantee a payoff of at least 0.5 and he cannot do better even when his ratio is $1 - \varepsilon$.

The equivalence for mean-payoff Richman-bidding games can be seen as an extension of the equivalence of reachability Richman-bidding games. Since no equivalence is known for reachability poorman-bidding games, we find the equivalence for mean-payoff poorman-bidding particularly surprising. The optimal payoff Max can guarantee with a ratio of $r \in (0,1)$ in $\mathcal{G}_{\bowtie}$ is r. For example, when the initial budgets are $\langle 3, 1 \rangle$, Max's ratio is $\frac{3}{4}$, and he can guarantee a payoff arbitrarily close to $\frac{3}{4}$ (in a similar manner to Richman bidding above). This means that in the long-run, Max can win 3 times more biddings than Min. Thus, given the option to choose between first-price Richman and poorman bidding, Max prefers using first-price poorman bidding when his budget exceeds Min's budget.

We turn to illustrate the results for all-pay bidding. Again, since reachability all-pay bidding games are technically involved, we find the equivalences in mean-payoff games to be particularly good news. First, under all-pay Richman, pure (deterministic) strategies are "useless". Using mixed strategies, first-price and all-pay coincide. Specifically, using a pure strategy, Max cannot guarantee a payoff greater than 0 in $\mathcal{G}_{\bowtie}$, and he has a mixed strategy that guarantees an almost-sure payoff of 0.5. Thus, given a choice between all-pay and first-price Richman, Max would not have a preference between the two bidding mechanisms.

Surprisingly, the properties of all-pay poorman bidding are quite different. First, in contrast to all-pay Richman bidding, pure strategies are "useful" under all-pay poorman when $r > \frac{1}{2}$. For example, when $r = \frac{3}{4}$, Max can guarantee a payoff of $\frac{2}{3}$ with a pure strategy. Not too far from $\frac{3}{4}$, the optimal payoff under first-price poorman bidding. Second, when allowing mixed strategies, given the choice between all-pay and first-price poorman bidding, when $r > \frac{1}{2}$, Max prefers all-pay poorman! With a ratio of $r = \frac{3}{4}$, Max has a mixed strategy that can guarantee an almost-sure payoff of $\frac{5}{6}$; higher than the optimal payoff under first-price poorman.

References

1 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

2 K.R. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists.* Cambridge University Press, 2011.

3 G. Avni and T. A. Henzinger. A survey of bidding games on graphs. In *Proc. 31st CONCUR*, volume 171 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

4 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. *J. ACM*, 66(4):31:1–31:29, 2019.

5 G. Avni, T. A. Henzinger, and R. Ibsen-Jensen. Infinite-duration poorman-bidding games. In *Proc. 14th WINE*, volume 11316 of *LNCS*, pages 21–36. Springer, 2018.

6 G. Avni, R. Ibsen-Jensen, and J. Tkadlec. All-pay bidding games on graphs. In *Proc. 34th AAAI*, pages 1798–1805. AAAI Press, 2020.

7 G. Avni, I. Jecker, and Đ. Žikelić. Infinite-duration all-pay bidding games. In *Proc. 32nd SODA*, pages 617–636, 2021.

8 C. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th STOC*, 2017.

9 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

10 A. E. Emerson, C. S. Jutla, and P. A. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th CAV*, pages 385–396, 1993.

11 M. Jurdzinski. Deciding the winner in parity games is in up ∩ co-up. *Information Processing Letters*, 68(3):119–124, 1998.

12 A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial games under auction play. *Games and Economic Behavior*, 27(2):229–264, 1999.

13 A. J. Lazarus, D. E. Loeb, J. G. Propp, and D. Ullman. Richman games. *Games of No Chance*, 29:439–449, 1996.

14 N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.

15 Y. Peres, O. Schramm, S. Sheffield, and D. B. Wilson. Tug-of-war and the infinity laplacian. *J. Amer. Math. Soc.*, 22:167–210, 2009.

16 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

17 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

Probabilistic Model Checking for Strategic Equilibria-Based Decision Making: Advances and Challenges

Marta Kwiatkowska
University of Oxford, Oxford, UK

Gethin Norman
University of Glasgow, Glasgow, UK
University of Oxford, Oxford, UK

David Parker
University of Birmingham, Birmingham, UK

Gabriel Santos
University of Oxford, Oxford, UK

Rui Yan
University of Oxford, Oxford, UK

Abstract

Game-theoretic concepts have been extensively studied in economics to provide insight into competitive behaviour and strategic decision making. As computing systems increasingly involve concurrently acting autonomous agents, game-theoretic approaches are becoming widespread in computer science as a faithful modelling abstraction. These techniques can be used to reason about the competitive or collaborative behaviour of multiple rational agents with distinct goals or objectives. This paper provides an overview of recent advances in developing a modelling, verification and strategy synthesis framework for concurrent stochastic games implemented in the probabilistic model checker PRISM-games. This is based on a temporal logic that supports finite- and infinite-horizon temporal properties in both a zero-sum and nonzero-sum setting, the latter using Nash and correlated equilibria with respect to two optimality criteria, social welfare and social fairness. We summarise the key concepts, logics and algorithms and the currently available tool support. Future challenges and recent progress in adapting the framework and algorithmic solutions to continuous environments and neural networks are also outlined.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Probabilistic model checking, stochastic games, equilibria

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.4

Category Invited Talk

Funding This project was funded by the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115).

1 Introduction

Game-theoretic techniques have long been a source of fundamental insights into strategic decision making for multi-agent systems. They have been widely studied in areas such as economics [27], control [43] and robotics [36]. Concurrent stochastic multi-player games (CSGs), in particular, provide a natural framework for modelling a set of interactive, rational agents operating concurrently within an uncertain or stochastic environment. They can be viewed as a collection of players (agents) with strategies for determining their actions based on the execution so far, and where the resulting evolution of the system is probabilistic.

Game-theoretic analysis is versatile, in that it can support both zero-sum and nonzero-sum (equilibria) analysis. Zero-sum properties focus on scenarios in which one player (or a coalition of players) aims to optimise some objective, while the remaining players form a coalition with the directly opposing goal. On the other hand, nonzero-sum (equilibria) properties correspond to situations where two or more players (or coalitions of players) in a CSG have distinct objectives to be maximised or minimised. In nonzero-sum properties the goals of the players (or coalitions) are not necessarily directly opposing, and therefore it may be beneficial for players to collaborate. Competitive scenarios occur in many applications, e.g., attackers and defenders in the context of computer security. Similarly, collaborative behaviour can be essential, e.g., to effectively control a multi-robot system, or for users to send data efficiently through a shared medium in a communication protocol.

Probabilistic model checking is a powerful approach to the formal analysis of systems with stochastic behaviour. It relies on the construction and analysis of a probabilistic model, guided by a formal specification of its desired behaviour in temporal logic. It is of particular benefit in the context of models, such as stochastic games, which combine nondeterministic and probabilistic behaviour. This is because the interplay between these aspects of the model can be subtle and lead to unexpected results if not carefully modelled and analysed. This is exacerbated when the system comprises multiple agents with differing objectives.

Until recently, practical applications of probabilistic model checking based on stochastic games had focused primarily on *turn-based* models [15], in which simultaneous decision making by agents is forbidden. Alternatively, model checking of *non-stochastic* games has been extensively studied, and tool support developed [5, 39]. CSGs provide a more powerful and realistic modelling formalism, but also bring considerable challenges, in terms of the higher computational complexity or undecidability for some key problems.

There has nonetheless been significant amounts of work on tackling verification problems for CSGs. A number of algorithms have been proposed for solving CSGs against formally specified zero-sum properties, e.g. [17, 18, 11]. In the case of nonzero-sum properties, [14, 25] study the existence of and the complexity of finding equilibria for stochastic games. Complexity results for finding equilibria are also considered in [9] and [23] for quantitative reachability properties and temporal logic properties, respectively. Other work concerns finding equilibria for discounted properties; we mention [49], which formulates a learning-based algorithm, and [40], which presents iterative algorithms. However these advances are mostly lacking in implementations, tool support or case studies. Tools exist for solving turn-based stochastic games [13, 16] and non-stochastic concurrent games [16, 8, 10, 55, 24, 45], with the latter class including support for computing equilibria.

At the same time, there is an increasing trend to incorporate data-driven decision making, which necessitates the incorporation on of machine learning components within autonomous systems, which are built largely using conventional, symbolic methods. Examples of such *neuro-symbolic* systems are self-driving cars whose vision function is provided via a neural network image classifier, or an aircraft controller whose collision avoidance system uses a neural network for decision support. Design automation support for such systems is lacking, yet automatic computation of equilibria aids in ensuring stable solutions.

This paper provides an overview of recent advances in developing a modelling, verification and strategy synthesis framework for concurrent stochastic games, as implemented in the PRISM-games probabilistic model checker [33]. The framework uses a temporal logic that supports a wide range of finite- and infinite-horizon properties, relating to the probability of an event's occurrence or the expected amount of reward or cost accumulated. The logic allows specification of both *zero-sum* and *nonzero-sum* properties, with the latter expressed using either *Nash equilibria* or *correlated equilibria*. For both types of equilibria, strategies are synthesised in which it is not beneficial for any player to unilaterally alter their chosen actions, but correlated equilibria also allow players to coordinate through *public signals*. Since several, varied such equilibria may exist, we also support distinct optimality criteria to select between them; we consider *social welfare*, which maximises the sum of the players utilities, and *social fairness*, which minimises the difference between the utilities.

We summarise the key concepts, logics and algorithms that underlie this framework and discuss the tool support provided by PRISM-games, including an illustrative case study of formally modelling and analysing a multi-agent communication protocols using CSGs. Future challenges and recent progress in extending the framework and algorithmic solutions to modelling of neuro-symbolic CSGs are also outlined. In contrast to the majority of prior research, the focus of this strand of work is on software tool development, applications and case studies.

2 Normal form games

We introduce the main concepts used in this paper by means of simple one-shot games known as *normal form games* (NFGs), where players make their choices at the same time. We consider both zero-sum NFGs and nonzero-sum NFGs, then define equilibria concepts for these games and summarise existing algorithms for equilibria computation.

We first require the following notation. Let $\mathit{Dist}(X)$ denote the set of probability distributions over set X. For any vector $v \in \mathbb{R}^n$, we use $v(i)$ to refer to the ith entry of the vector. For any tuple $x = (x_1, \ldots, x_n) \in X^n$, element $x' \in X$ and $i \leqslant n$, we define the tuples $x_{-i} \overset{\text{def}}{=} (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ and $x_{-i}[x'] \overset{\text{def}}{=} (x_1, \ldots, x_{i-1}, x', x_{i+1}, \ldots, x_n)$.

▶ **Definition 1** (Normal form game). *A (finite, n-player)* normal form game *(NFG) is a tuple* $\mathsf{N} = (N, A, u)$ *where:*

- $N = \{1, \ldots, n\}$ *is a finite set of* players*;*
- $A = A_1 \times \cdots \times A_n$ *and* A_i *is a finite set of* actions *available to player* $i \in N$*;*
- $u = (u_1, \ldots, u_n)$ *and* $u_i \colon A \to \mathbb{R}$ *is a* utility function *for player* $i \in N$.

In a normal form game N, the players choose actions simultaneously, with player $i \in N$ choosing an action from the set A_i and, assuming that each player $i \in N$ selects action a_i, player j receives the utility $u_j(a_1, \ldots, a_n)$. The *objective* of each player is to maximise their utility and their choices are governed by *strategies*, which we now define. We will also distinguish *strategy profiles*, which comprise a strategy for each player, and *correlated profiles*, which correspond to choices of the players when they are allowed to coordinate through a (probabilistic) *public signal*.

▶ **Definition 2** (Strategies, profiles and correlated profiles). *For an NFG* N*:*

- *a* strategy σ_i *for player* i *in an NFG* N *is a probability distribution over the set of actions* A_i *and we let* Σ^i_{N} *denote the set of all strategies for player* i*;*
- *a* strategy profile *(or* profile*)* $\sigma = (\sigma_1, \ldots, \sigma_n)$ *is a tuple of strategies for each player;*
- *a* correlated profile *is a tuple* (τ, ς) *comprising* $\tau \in \mathit{Dist}(D_1 \times \cdots \times D_n)$*, where* D_i *is a finite set of* signals *for player* i*, and* $\varsigma = (\varsigma_1, \ldots, \varsigma_n)$*, where* $\varsigma_i \colon D_i \to A_i$ *is a function from the signals of player* i *to the actions of player* i.

For a correlated profile (τ, ς) of N, the public signal τ is a joint distribution over signals D_i for each player i such that, if player i receives the signal $d_i \in D_i$, then it chooses action $\varsigma_i(d_i)$. We can consider any correlated profile (τ, ς) as a *joint strategy*, i.e., a distribution over $A_1 \times \cdots \times A_n$ where:

$$(\tau, \varsigma)(a_1, \ldots, a_n) = \textstyle\sum\{\tau(d_1, \ldots, d_n) \mid d_i \in D_i \wedge \varsigma(d_i) = a_i \text{ for all } i \in N\}\,.$$

Conversely, any joint strategy $\tau \in \mathit{Dist}(A_1 \times \cdots \times A_n)$ of N can be considered as a correlated profile (τ, ς), where $D_i = A_i$ and ς_i is the identity function for $i \in N$. Any profile σ of an NFG N can be mapped to an equivalent correlated profile (in which τ is the joint distribution $\sigma_1 \times \cdots \times \sigma_n$ and ς_i is the identity function). On the other hand, there are correlated profiles with no equivalent strategy profile.

Under profile σ or correlated profile (τ, ς) the expected utilities of player i are:

$$\begin{array}{rcl} u_i(\sigma) & \overset{\text{def}}{=} & \sum_{(a_1,\dots,a_n)\in A} u_i(a_1,\dots,a_n) \cdot \left(\prod_{j=1}^n \sigma_j(a_j)\right) \\ u_i(\tau,\varsigma) & \overset{\text{def}}{=} & \sum_{(d_1,\dots,d_n)\in D} \tau(d_1,\dots,d_n) \cdot u_i(\varsigma_1(d_1),\dots,\varsigma_n(d_n))\,. \end{array}$$

▶ **Example 3.** Consider the two-player NFG with available action sets $A_i = \{\mathit{heads}_i, \mathit{tails}_i\}$ for $1 \leqslant i \leqslant 2$ and a correlated profile corresponding to the joint distribution $\tau \in \mathit{Dist}(A_1 \times A_2)$, where $\tau(\mathit{heads}_1, \mathit{heads}_2) = \tau(\mathit{tails}_1, \mathit{tails}_2) = 0.5$. Under this correlated profile, the players share a fair coin and choose their action based on the outcome of the coin toss. There is no equivalent strategy profile.

2.1 Zero-sum NFGs

A *zero-sum NFG* is a two-player NFG N such that $u_1(\alpha)+u_2(\alpha) = 0$ for all $\alpha \in A$, meaning that the objectives of the players are directly opposing. Such an NFG is often called a *matrix game*, as it can be represented by a single matrix $\mathsf{Z} \in \mathbb{Q}^{l \times m}$, where $A_1 = \{a_1, \dots, a_l\}$, $A_2 = \{b_1, \dots, b_m\}$ and $z_{ij} = u_1(a_i, b_j) = -u_2(a_i, b_j)$.

We next introduce the notion of the *value* of a zero-sum NFG and recall classical results about the existence of optimal strategies.

▶ **Theorem 4** (Minimax theorem [56, 57]). *For any zero-sum NFG $\mathsf{N} = (N, A, u)$ and corresponding matrix game Z, there exists $v^\star \in \mathbb{Q}$, called the* value *of the game and denoted $\mathit{val}(\mathsf{Z})$, such that:*

- *there is a strategy $\sigma_1^\star$ for player 1, called an optimal strategy of player 1, such that under this strategy the player's expected utility is at least $v^\star$ regardless of the strategy of player 2, i.e., $\inf_{\sigma_2 \in \Sigma_{\mathsf{N}}^2} u_1(\sigma_1^\star, \sigma_2) \geqslant v^\star$;*
- *there is a strategy $\sigma_2^\star$ for player 2, called an optimal strategy of player 2, such that under this strategy the player's expected utility is at least $-v^\star$ regardless of the strategy of player 1, i.e., $\inf_{\sigma_1 \in \Sigma_{\mathsf{N}}^1} u_2(\sigma_1, \sigma_2^\star) \geqslant -v^\star$.*

The value of a matrix game $\mathsf{Z} \in \mathbb{Q}^{l \times m}$ can be found by solving a linear programming (LP) problem [56, 57].

▶ **Example 5.** Table 1 shows a classic example of a two-player zero-sum game known as *matching pennies*. Columns α and u_i represent the collective choice (profile) and player i's utility, respectively. In this example, each player has a coin for which they may choose the value to be heads or tails, i.e., $A_i = \{\mathit{heads}_i, \mathit{tails}_i\}$. If the coins match, player 1 wins the round, which is indicated by being awarded a utility of 1, while player 2 receives utility -1. If the coins do not match, then the players' utilities are negated.

Table 1 Matching pennies game in normal form.

α	$u_1(\alpha)$	$u_2(\alpha)$	α	$u_1(\alpha)$	$u_2(\alpha)$
$(\mathit{heads}_1, \mathit{heads}_2)$	1	-1	$(\mathit{tails}_1, \mathit{heads}_2)$	-1	1
$(\mathit{heads}_1, \mathit{tails}_2)$	-1	1	$(\mathit{tails}_1, \mathit{tails}_2)$	1	-1

The value for the corresponding matrix game is the solution to the following LP problem: Maximise v subject to:

$$x_1 - x_2 \geqslant v, \; x_2 - x_1 \geqslant v, \; x_1 + x_2 = 1$$

which yields the value $v^\star = 0$ with optimal strategy $\sigma_1^\star = (\frac{1}{2}, \frac{1}{2})$ for player 1 (the optimal strategy for player 2 is the same).

2.2 Nonzero-sum NFGs

The requirement for players to have directly opposing objectives is often too limiting, and it is necessary to allow distinct objectives, which cannot be modelled in a zero-sum fashion. These scenarios can be captured using the notion of *equilibria*, defined by a separate, independent objective for each agent. We now define the concepts of *Nash equilibrium* [57] and *correlated equilibrium* [6] for NFGs, which ensure stability against deviations by individual agents, improving the overall game outcomes. Since many equilibria may exist, we also introduce optimality criteria for these equilibria: *social welfare*, which is standard [46], and *social fairness*, which was first defined in [35].

Before giving the formal definitions, we first extend our notation as follows: for any profile σ and strategy $\sigma_i^\star$, the strategy tuple σ_{-i} corresponds to σ with the strategy of player i removed and $\sigma_{-i}[\sigma_i^\star]$ to the profile σ after replacing player i's strategy with $\sigma_i^\star$.

▶ **Definition 6** (Best response). *For any nonzero-sum NFG* N *and profile* σ *or correlated profile* (τ, ς) *of* N*, the* best response *moves for player* i *to* σ_{-i} *and* (τ, ς_{-i}) *are, respectively:*

- *a strategy* $\sigma_i^\star$ *for player* i *such that* $u_i(\sigma_{-i}[\sigma_i^\star]) \geqslant u_i(\sigma_{-i}[\sigma_i])$ *for all* $\sigma_i \in \Sigma^i_{\mathsf{N}}$*;*
- *a function* $\varsigma_i^\star \colon D_i \to A_i$ *for player* i *such that* $u_i(\tau, \varsigma_{-i}[\varsigma_i^\star]) \geqslant u_i(\tau, \varsigma_{-i}[\varsigma_i])$ *for all functions* $\varsigma_i \colon D_i \to A_i$*.*

▶ **Definition 7** (NE and CE). *For any nonzero-sum NFG* N*, a strategy profile* $\sigma^\star$ *is a* Nash equilibrium *(NE) and a correlated profile* $(\tau, \varsigma^\star)$ *of* N *is a* correlated equilibrium *(CE) if:*

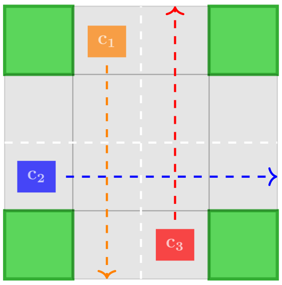
- $\sigma_i^\star$ *is a best response to* $\sigma_{-i}^\star$ *for all* $i \in N$*;*
- $\varsigma_i^\star$ *is a best response to* $(\tau, \varsigma_{-i}^\star)$ *for all* $i \in N$*;*

respectively.

Any NE of N is also a CE, while there exist CEs that cannot be represented by a strategy profile, and therefore are not NEs. For each class of equilibria, NE and CE, we introduce two optimality criteria, the first maximising *social welfare* (SW), defined as the *sum* of the utilities, and the second maximising *social fairness* (SF), which minimises the *difference* between the players' utilities. Other variants of fairness have been considered for NEs, such as in [38], where the authors seek to maximise the lowest utility among the players.

▶ **Definition 8** (SW and SF). *An equilibrium* $\sigma^\star$ *is a* social welfare *(SW) equilibrium if the sum of the utilities of the players under* $\sigma^\star$ *is maximal over all equilibria, while* $\sigma^\star$ *is a* social fair *(SF) equilibrium if the difference between the player's utilities under* $\sigma^\star$ *is minimised over all equilibria.*

We can also define the dual concept of *social cost* (SC) equilibria [34], where players try to minimise, rather than maximise, their expected utilities by considering equilibria of the game $\mathsf{N}^- = (N, A, -u)$ in which the utilities of N are negated. We remark that SC equilibria strategies are not a subset of classically defined NE or CE strategies of N.



α	$u_1(\alpha)$	$u_2(\alpha)$	$u_3(\alpha)$
(pro_1, pro_2, pro_3)	-1000	-1000	-100
(pro_1, pro_2, yld_3)	-1000	-100	-5
(pro_1, yld_2, pro_3)	5	-5	5
(pro_1, yld_2, yld_3)	5	-5	-5
(yld_1, pro_2, pro_3)	-5	-1000	-100
(yld_1, pro_2, yld_3)	-5	5	-5
(yld_1, yld_2, pro_3)	-5	-5	5
(yld_1, yld_2, yld_3)	-10	-10	-10

Figure 1 Example from [35]: Cars at an intersection and the corresponding NFG.

▶ **Example 9.** Consider the scenario from [35], based on an example from [50], where three cars meet at an intersection and want to proceed as indicated by the arrows in Figure 1. Each car can either *proceed* or *yield*. If two cars with intersecting paths proceed, then there is an accident. If an accident occurs, the car having the right of way, i.e., the other car is to its left, has a utility of −100 and the car that should yield has a utility of −1000. If a car proceeds without causing an accident, then its utility is 5 and the cars that yield have a utility of −5. If all cars yield, then, since this delays all cars, all have utility −10. The 3-player NFG is given in Figure 1. The different optimal equilibria of the NFG are:

- the SWNE and SWCE are the same: for c_2 to yield and c_1 and c_3 to proceed, with the expected utilities of the players $(5, -5, 5)$;
- the SFNE is for c_1 to yield with probability 1, c_2 to yield with probability 0.863636 and c_3 to yield with probability 0.985148, with the expected utilities of the players $(-9.254050, -9.925742, -9.318182)$;
- the SFCE gives a joint distribution where the probability of c_2 yielding and of c_1 and c_3 yielding are both 0.5 with the expected utilities of the players $(0, 0, 0)$.

Modifying u_2 such that $u_2(pro_1, pro_2, pro_3) = -4.5$ to, e.g., represent a reckless driver, the SWNE becomes for c_1 and c_3 to yield and c_2 to proceed with the expected utilities of the players $(-5, 5, -5)$, while the SWCE is still for c_2 to yield and c_1 and c_3 to proceed. The SFNE and SFCE also do not change.

Algorithms for computing equilibria in NFGs. Finding NEs in two-player NFGs is in the class of *linear complementarity* problems (LCPs). Established algorithms include the Lemke-Howson algorithm [37], which is based on the method of labelled polytopes [46], support enumeration [48] and regret minimisation [52]. In [34] a method for NE computation is developed, which reduces the problem to SMT via labelled polytopes [46] by considering the regions of the strategy profile space. This method iteratively reduces the search space of profiles as positive probability assignments are found and added as constraints on the profiles. This approach can also be used for finding both an SWNE and SFNE by computing all NEs and then selecting an optimal one.

In the case of NFGs with more than two players, the computation of NEs is more complex since, for a given support (i.e., a sub-region of the strategy profile space which fixes the set of actions chosen with nonzero probability by each player), finding NEs cannot be reduced to an LP problem. A method for such NFGs is presented in [32], based on support enumeration [48], which exhaustively examines all supports one at a time, checking whether that sub-region contains NEs. For each support, finding an SWNE can be reduced to a *nonlinear programming problem* [32]. This nonlinear programming problem can be modified to find an SFNE in each support [35].

In the case of CEs, the approach introduced in [35] is to first find a joint strategy for the players, i.e., a distribution over the action tuples, which can then be mapped to a correlated profile. For SWCEs, [35] reduces the computation to solving a LP problem which has $|A|$ variables, one for each action tuple, and $\sum_{i \in N}(|A_i|^2 - |A_i|) + |A| + 1$ constraints. For SFCEs, on the other hand, the method of [35] involves solving an optimisation problem with an additional has $|N| + 2$ variables and $3 \cdot |N|$ constraints compared to the LP problem for finding SWCEs.

3 Concurrent Stochastic Games

This section introduces *concurrent stochastic games* (CSGs) [54], in which players repeatedly make simultaneous choices over actions and the action choices cause a probabilistic update of the game state. CSGs thus provide a natural framework for modelling a set of interactive, rational agents operating concurrently within an uncertain or probabilistic environment. Compared to normal form games, they are classified as *multi-stage*, which is more convenient for specifying repeated or sequential interactions among agents. The introduction of stochasticity facilitates modelling of a wide range of important phenomena, for example uncertain behaviour due to noisy sensors or unreliable hardware in a multi-robot system, or the use of randomisation for coordination in a distributed security or networking protocol.

▶ **Definition 10** (Concurrent stochastic game). *A concurrent stochastic multi-player game (CSG) is a tuple $\mathsf{G} = (N, S, \bar{s}, A, \Delta, \delta)$ where:*

- *$N = \{1, \dots, n\}$ is a finite set of players;*
- *S is a finite set of states and $\bar{s} \in S$ is an initial state;*
- *$A = (A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\})$ where A_i is a finite set of actions available to player $i \in N$ and $\bot$ is an idle action disjoint from the set $\cup_{i=1}^n A_i$;*
- *$\Delta \colon S \to 2^{\cup_{i=1}^n A_i}$ is an action assignment function;*
- *$\delta \colon S \times A \to \mathit{Dist}(S)$ is a probabilistic transition function.*

Given a CSG G, the set of actions available to player $i \in N$ in state $s \in S$ is given by $A_i(s) \overset{\text{def}}{=} \Delta(s) \cap A_i$. The CSG G starts in the initial state $\bar{s}$ and, if G is in the game state s, then each player $i \in N$ selects an action from its available actions in state s if this set is non-empty, and from $\{\bot\}$ otherwise. Next, supposing each player $i \in N$ chooses action a_i, the game state is updated according to the distribution $\delta(s, (a_1, \dots, a_n))$. We allow sets of players $C \subseteq N$ to form *coalitions*, and will consider the induced CSG, called the *coalition game*, with coalitions as players.

A *path* π of a CSG G is a sequence $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots$, where $s_i \in S$, $\alpha_i \in A$ and $\delta(s_i, \alpha_i)(s_{i+1}) > 0$ for all $i \geqslant 0$. We denote by $\mathit{FPaths}_{\mathsf{G},s}$ and $\mathit{IPaths}_{\mathsf{G},s}$ the sets of finite and infinite paths starting in state s of G, respectively, and drop the subscript s when considering all finite and infinite paths of G. As for NFGs, we can define *strategies* of G that resolve the choices of the players. Here, a strategy for player i is a function $\sigma_i \colon \mathit{FPaths}_{\mathsf{G}} \to \mathit{Dist}(A_i \cup \{\bot\})$ mapping finite paths to distributions over available actions, such that, if $\sigma_i(\pi)(a_i) > 0$, then $a_i \in A_i(\mathit{last}(\pi))$ where $\mathit{last}(\pi)$ is the final state of π. Furthermore, we can define strategy profiles, correlated profiles and joint strategies analogously to Section 2.

A *labelled* CSG is a tuple (G, AP, L), where G is a CSG, as in Definition 10, AP is a set of atomic propositions and $L \colon S \to 2^{AP}$ is a labelling function, specifying which atomic propositions are true in each state. We also associate CSGs with *reward structures*, which annotate states and transitions with real values. More precisely a reward structure is a pair $r = (r_A, r_S)$ consisting of an action reward function $r_A \colon S \times A \to \mathbb{R}$ and state reward function $r_S \colon S \to \mathbb{R}$. We use atomic propositions and rewards as the building blocks to specify players' utilities in a CSG, which will be described in Section 4.

Formally, the utility function or *objective* of player i in a CSG is given by a random variable $X_i \colon \mathit{IPaths}_{\mathsf{G}} \to \mathbb{R}$ over infinite paths. For a profile σ and state s, using standard techniques [31], we can construct a probability measure $\mathit{Prob}^{\sigma}_{\mathsf{G},s}$ over the paths that start in state s corresponding to σ, denoted $\mathit{IPaths}^{\sigma}_{\mathsf{G},s}$, and define the expected value $\mathbb{E}^{\sigma}_{\mathsf{G},s}(X_i)$ of player i's utility from s under σ. Similarly, we can also define such a probability measure and expected value given a correlated profile or joint strategy of G.

3.1 Zero-sum CSGs

Similarly to NFGs (see Section 2.1), *zero-sum* CSGs are two-player games that have a single utility function X for player 1, with the utility function of player 2 given by $-X$, and both players aiming to maximise the expected value of their utility. Equivalently, we can suppose that player 1 tries to maximise the expected value of X, while player 2 tries to minimise it. As for NFGs (see Theorem 4), a CSG has a *value* with respect to X if it is determined, i.e., if the maximum value that player 1 can ensure equals the minimum value that player 2 can ensure when starting from any state of the CSG. Since the CSGs we discuss in this paper are finite-state and finitely-branching, it follows that they are determined for all of the objectives that we consider [44].

Given a multi-player CSG and objective X, we can divide the players into two coalitions, $C \subseteq N$ and $N \backslash C$, and then construct a *two-player* zero-sum coalition game, in which each coalition acts as a single player, with one coalition trying to maximise the value of X and the other trying to minimise that value.

3.2 Nonzero-sum CSGs

We define *nonzero-sum* CSGs similarly to NFGs: we assume that there is a distinct and independent objective X_i for each player i (or coalitions of players). We can then define NE and CE for CSGs (see Definition 7), as well as the restricted classes of SW and SF equilibria, similarly to those for NFGs (see Definition 8). Following [34, 32], we focus on *subgame-perfect* equilibria [47], which are equilibria in *every state* of G. Furthermore, because we include infinite-horizon objectives, where the existence of NE is an open problem [7], we will in some cases use ε-NE, which do exist for any $\varepsilon > 0$ for all the infinite-horizon objectives we consider.

▶ **Definition 11** (Subgame-perfect ε-NE)**.** *For CSG G and $\varepsilon > 0$, a strategy profile $\sigma^\star$ is a* subgame-perfect ε-Nash equilibrium *for objectives $\langle X_i \rangle_{i \in N}$ if and only if $\mathbb{E}^{\sigma^\star}_{\mathsf{G},s}(X_i) \geqslant \sup_{\sigma_i \in \Sigma_i} \mathbb{E}^{\sigma^\star_{-i}[\sigma_i]}_{\mathsf{G},s}(X_i) - \varepsilon$ for all $i \in N$ and $s \in S$.*

▶ **Example 12.** As an example scenario that can be modelled as a CSG, consider a number of users trying to send packets using the slotted ALOHA protocol studied in [32, 34, 35]. If there is a collision or if sending a packet fails, a user waits for some number of slots before resending, with the wait set according to an exponential backoff scheme.

If we model this scenario as a CSG then, when a player has a packet to send, the actions available to the player correspond to either sending their packet or waiting to send the packet at some future time step. In the case when one coalition of players has an objective related to sending their packets efficiently, e.g., minimising the expected time to send their packets, and the remaining players form a second coalition and have the dual objective, we can model this scenario as a zero-sum CSG. In such a zero-sum CSG, the optimal strategy for the first coalition is to try and choose times to send that avoid collisions, while the second coalition will do the opposite and instead try and cause collisions. On the other hand, when there are more coalitions and each coalition's goal corresponds to sending their own packets efficiently, we can model this as a nonzero-sum CSG. Here we would be looking for equilibria, i.e., profiles such that no coalition could improve its objective by changing its strategy, which are also optimal, e.g., the sum of the expected times is minimal or the difference between the expected time to send for each coalition is minimal.

4 Property specifications and model checking for CSGs

Probabilistic model checking is a technique for systematically constructing a stochastic model and analysing it against a quantitative property formally specified in temporal logic. This approach can be used either to *verify* that a specification is always satisfied or to perform *strategy synthesis*, i.e., to construct a witness to the satisfaction of a property. In the context of CSGs, the latter means synthesising strategies for one or more players (or coalitions) such that the resulting behaviour of the game satisfies the specification.

To specify properties of labelled CSGs, we use the property specification language of the PRISM-games model checker [33], which is based on the logic PCTL (probabilistic computation tree logic) [26], extended with operators to specify expected reward properties [21] and the *coalition* operator $\langle\langle C\rangle\rangle$ from alternating temporal logic (ATL) [4]. The variant of this logic that just considers *zero-sum* formulae is referred to as rPATL (probabilistic alternating-time temporal logic with rewards) in [15], but here we use a further extended version that also supports *nonzero-sum* properties, using the notion of equilibria [34, 35].

▶ **Definition 13** (PRISM-games logic [34, 35]). *The syntax of the PRISM-games logic is given by the grammar:*

$$
\begin{array}{lcl}
\phi & \coloneqq & \mathtt{true} \mid \mathsf{a} \mid \neg\phi \mid \phi\wedge\phi \mid \langle\langle C\rangle\rangle\mathtt{P}_{\sim q}[\,\psi\,] \mid \langle\langle C\rangle\rangle\mathtt{R}^{r}_{\sim x}[\,\rho\,] \mid \langle\langle \mathbb{C}\rangle\rangle(\star_1,\star_2)_{\mathrm{opt}\sim x}(\theta) \\
\psi & \coloneqq & \mathtt{X}\,\phi \mid \phi\ \mathtt{U}^{\leqslant k}\ \phi \mid \phi\ \mathtt{U}\ \phi \\
\rho & \coloneqq & \mathtt{I}^{=k} \mid \mathtt{C}^{\leqslant k} \mid \mathtt{F}\ \phi \\
\theta & \coloneqq & \mathtt{P}[\,\psi\,]{+}\cdots{+}\mathtt{P}[\,\psi\,] \mid \mathtt{R}^{r}[\,\rho\,]{+}\cdots{+}\mathtt{R}^{r}[\,\rho\,]
\end{array}
$$

where a *is an atomic proposition,* $\mathbb{C} = C_1{:}\cdots{:}C_m$*,* C *and* $C_1,\dots,C_m$ *are coalitions of players such that* $C' = N\backslash C$*,* $C_i\cap C_j = \varnothing$ *for all* $1\leqslant i\neq j\leqslant m$*,* $(\star_1,\star_2)\in\{\mathrm{NE},\mathrm{CE}\}\times\{\mathrm{SW},\mathrm{SF}\}$*,* $\mathrm{opt}\in\{\min,\max\}$*,* $\sim\in\{<,\leqslant,\geqslant,>\}$*,* $q\in\mathbb{Q}\cap[0,1]$*,* $x\in\mathbb{Q}$*,* r *is a reward structure and* $k\in\mathbb{N}$*.*

The syntax distinguishes between state (ϕ), path (ψ) and reward (ρ) formulae. State formulae are evaluated over states of a CSG, while path and reward formulae are both evaluated over paths. Sums of formulae (θ) are used to specify multiple objectives for equilibria.

We omit the formal semantics, which can be found in [34, 35]. Path and reward formulae are used to express the utilities of the players, i.e., random variables over paths. For path formulae, we allow *next* ($\mathtt{X}\,\phi$), *bounded until* ($\phi\ \mathtt{U}^{\leqslant k}\ \phi$) and *unbounded until* ($\phi\ \mathtt{U}\ \phi$). We also allow the usual equivalences such as $\mathtt{F}\ \phi \equiv \mathtt{true}\ \mathtt{U}\ \phi$ (i.e., *probabilistic reachability*) and $\mathtt{F}^{\leqslant k}\ \phi \equiv \mathtt{true}\ \mathtt{U}^{\leqslant k}\ \phi$ (i.e., *bounded probabilistic reachability*). The random variable corresponding to the path formula ψ returns 1 for paths that satisfy ψ and zero otherwise. For reward formulae, we allow instantaneous (state) reward at the kth step (*instantaneous reward* $\mathtt{I}^{=k}$), reward accumulated over k steps (*bounded cumulative reward* $\mathtt{C}^{\leqslant k}$), and reward accumulated until a formula ϕ is satisfied (*expected reachability* $\mathtt{F}\ \phi$). The random variable corresponding to the reward formula ρ returns for a path the reward corresponding to ρ.

4.1 Zero-sum formulae

A state satisfies a formula $\langle\langle C\rangle\rangle\mathtt{P}_{\sim q}[\,\psi\,]$ if the coalition of players $C \subseteq N$ can ensure that the probability of the path formula ψ being satisfied is $\sim q$, regardless of the actions of the other players $(N \backslash C)$ in the game. A state satisfies a formula $\langle\langle C\rangle\rangle\mathtt{R}^{r}_{\sim x}[\,\rho\,]$ if the players in C can ensure that the expected value of the reward formula ρ for reward structure r is $\sim x$, whatever the other players do.

The model checking algorithms presented in [34] involve graph-based analysis followed by backward induction [53, 57] for exact computation of *finite-horizon properties* and value iteration [51, 12] for approximate computation of *infinite-horizon properties*. During both backward induction and value iteration for each state, at each iteration, an LP problem of size $|A|$ must be solved (corresponding to finding the value of a zero-sum one-shot game), which has complexity PTIME [30].

Strategy synthesis for the formulae $\langle\langle C\rangle\rangle\mathtt{P}_{\sim q}[\,\psi\,]$ and $\langle\langle C\rangle\rangle\mathtt{R}^{r}_{\sim x}[\,\rho\,]$ corresponds to finding optimal strategies for the players in coalition C when their objective, respectively, is maximising the probability of satisfying the formula ψ and maximising the expected value of the reward formula with respect to the reward structure r. All strategies synthesised are randomised and can be found during model checking by extracting not just the value of the zero-sum one-shot game solved in each state, but also an optimal (randomised) strategy. For infinite-horizon objectives, the synthesised strategies are memoryless, while for finite-horizon objectives, the synthesised strategies are finite-memory, with a separate distribution required for each state and each time step.

4.2 Nonzero-sum formulae

Nonzero-sum formulae allow us to reason about equilibria, for either of the types (NE or CE) and optimality criteria (SW or SF) considered here. A probabilistic formula $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle(\star_1,\star_2)_{\max\sim x}(\mathtt{P}[\,\psi_1\,]{+}\cdots{+}\mathtt{P}[\,\psi_m\,])$ is true in a state if, when the players form the coalitions $C_1,\dots,C_m$, there is a subgame-perfect equilibrium of type $\star_1$ meeting the optimality criterion $\star_2$ for which the *sum* of the values of the objectives $\mathtt{P}[\,\psi_1\,],\dots,\mathtt{P}[\,\psi_m\,]$ for the coalitions $C_1,\dots,C_m$ satisfies $\sim x$. The objective of coalition C_i is to maximise the probability of satisfying a path formula ψ_i.

For a reward formula $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle(\star_1,\star_2)_{\max\sim x}(\mathtt{R}^{r_1}[\,\rho_1\,]{+}\cdots{+}\mathtt{R}^{r_m}[\,\rho_m\,])$ the meaning is similar; however, here the objective of coalition C_i refers to a reward formula ρ_i with respect to reward structure r_i. Formulae of the form $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle(\star_1,\star_2)_{\min\sim x}(\theta)$ correspond to the dual notion of cost equilibria, which are also supported. We also allow *numerical* queries of the form $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle(\star_1,\star_2)_{\mathrm{opt}=?}(\theta)$, which return the sum of the subgame-perfect equilibrium's values of of type $\star_1$ meeting the optimality criterion $\star_2$.

Model checking algorithms, presented in [34, 32, 35], involve solving an m-player *coalition game* $\mathsf{G}^{\mathcal{C}}$, where $\mathcal{C} = \{C_1,\dots,C_m\}$ and the choices of each player i in $\mathsf{G}^{\mathcal{C}}$ correspond to the choices of the players in coalition C_i in G. If all the objectives in θ are finite-horizon, then *backward induction* [53, 57] can be applied to compute (precise) optimal equilibria values. On the other hand, if all the objectives are infinite-horizon, *value iteration* [12] can be used to approximate optimal equilibria values. When there is a combination of finite- and infinite-horizon objectives, the game under study is modified in a standard manner to make all objectives infinite-horizon.

Both backward induction and value iteration over the CSG $\mathsf{G}^{\mathcal{C}}$ work by iteratively computing new values for each state s of $\mathsf{G}^{\mathcal{C}}$. The values for each state, in each iteration, are found by computing optimal equilibria values, with respect to the criterion $\star_2$ and equilibrium type $\star_1$, of an NFG N whose utility function is derived from the outgoing transition probabilities from s in the CSG and the values computed for successor states of s in the previous iteration.

We can synthesise a strategy profile representing the appropriate type of equilibrium for the CSG by combining the optimal strategies for the equilibria generated in each individual state during solution. As for zero-sum formulae, randomisation is required and memory is needed both to keep track of both the step bound of finite-horizon objectives and the satisfaction of each player's objective.

▶ **Example 14.** We now return to the scenario from Example 12, where a number users are attempting to send packets using the slotted ALOHA protocol. The zero-sum formulae $\langle\langle \mathit{usr}_1, \ldots, \mathit{usr}_k \rangle\rangle \mathtt{R}^{\mathit{time}}_{\min=?}[\,\mathtt{F}\ \mathsf{sent}_{1\ldots k}\,]$ and $\langle\langle \mathit{usr}_1, \ldots, \mathit{usr}_k \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ \mathsf{sent}_{1\ldots k} \wedge t \leqslant D\,]$ represent the case where the first k users form a coalition and try to minimise the expected time to send their packets or maximise the probability they send their packets within a deadline $D \in \mathbb{N}$, respectively, while the remaining users form a second coalition and try and achieve the opposite objective, i.e., maximise the expected time or minimise the probability.

On the other hand, in the nonzero-sum case, if we suppose there are m users and the objective of each user is to minimise the expected time to send their packet, this can be expressed by the nonzero-sum formula $\langle\langle \mathit{usr}_1{:}\cdots{:}\mathit{usr}_m \rangle\rangle (\star_1, \star_2)_{\min=?}(\mathtt{R}^{\mathit{time}}[\,\mathtt{F}\ \mathsf{sent}_1\,]{+}\cdots{+}\mathtt{R}^{\mathit{time}}[\,\mathtt{F}\ \mathsf{sent}_m\,])$.

5 Tool support and case studies

Tool support for the modelling and automated verification of CSGs has been implemented in PRISM-games [33], which is available from [61]. A variety of case studies have been modelled and analysed as CSGs with the tool, using both zero-sum and nonzero-sum properties. These include: a robot coordination problem [34]; futures market investors [34, 35]; medium access control [32, 34]; power control [34, 35]; a public good game [32, 35] and secret sharing [32]. The results for these case studies demonstrate: the advantages of using CSGs for modelling (for example, with respect to simpler turn-based games); that using nonzero-sum properties can yield gains for the players (or coalitions); and that the use of correlated equilibria and social fairness results may be advantageous compared to Nash equilibria and social welfare. We give a brief description of the functionality and implementation of PRISM-games and then present a representative case study: the slotted ALOHA protocol.

5.1 PRISM-games

PRISM-games [33] is an extension of the PRISM model checker, which provides support for a variety of stochastic game models, including turn-based and concurrent multi-player stochastic games, and (turn-based) timed probabilistic games.

These are all described in the PRISM-games modelling language, a stochastic extension of the Reactive Modules formalism [3]. The language facilitates the specification of systems comprising multiple components, referred to as modules, that operate in parallel, both asynchronously and synchronously through action labels. Each module has a number of finite-valued variables and a state of the system specifies the values of the variables of all modules. The behaviour of each module is defined by probabilistic guarded commands, where the guard is a predicate over the variables of the modules and the command specifies a probabilistic update of the module's variables. In a CSG model, each player constitutes a set of modules, and these therefore execute concurrently.

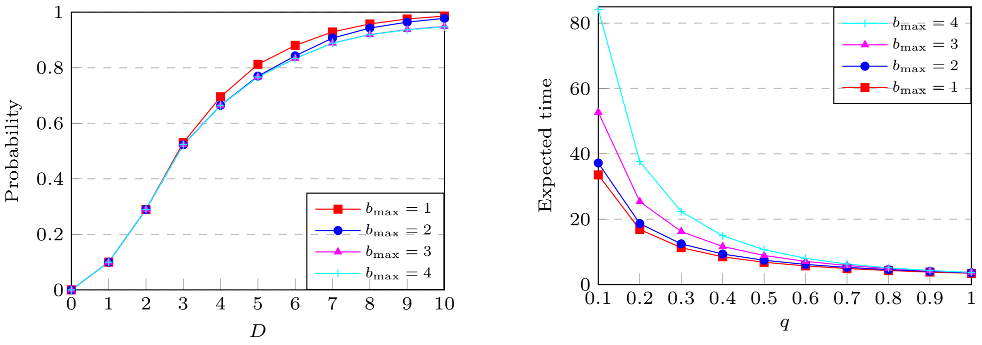


Figure 2 Results from a CSG model of the ALOHA protocol: one user maximising the probability of sending their packet before a deadline D (left); and minimising the expected time to send the packet, assuming a message transmission failure probability q (right).

PRISM-games provides a graphical user interface for designing and simulating stochastic games models, but its core functionality is to exhaustively construct a game and perform verification and strategy synthesis against a logical specification. For CSGs, the PRISM-games logic described in Definition 13 is supported, and the resulting strategies can be exported, simulated or further verified.

The implementation of CSG model checking is built within PRISM's "explicit" engine, which is based on sparse matrices and implemented in Java. Computing values (and optimal strategies) for zero-sum NFGs, needed for zero-sum formulae, is performed using the LPSolve library [41] via linear programming. The computation of SWNE or SFNE for nonzero-sum NFG, required for nonzero-sum formulae, depends on the number of players. For two players [34], labelled polytopes are used to characterise and find NE values through a reduction to SMT in both Z3 [19] and Yices [20]. If there are more than two players, the implementation [32] is based on support enumeration and uses a combination of the SMT solver Z3 [19] and the nonlinear optimisation suite Ipopt [58]. In the case of SWCE for nonzero-sum NFGs, as the problem reduces to an LP problem [35], either Gurobi [22] or the SMT solver Z3 [19] is used. Finally, for SFCE, since the problem does not reduce directly to an LP problem, only Z3 can be used.

5.2 The ALOHA case study

We now return to the slotted ALOHA protocol discussed in Examples 12 and 14 to illustrate the benefits of game-theoretic analysis with CSGs. For further details of this case study, as well as several others, see [61]. Recall that, in the slotted ALOHA protocol, a number of users are attempting to send packets on a shared medium. We assume that, in any time slot, if a single user tries to send a packet then there is a probability (q) that the packet is sent and, as more users try and send, then the probability of success decreases. If sending a packet fails, the user waits for a number of slots before resending, defined according to an exponential backoff scheme. More precisely, each user maintains a backoff counter, which it increases each time there is a failure (up to $b_{\max}$) and, if the counter equals k, randomly chooses the slots to wait from $\{0, 1, \ldots, 2^k{-}1\}$.

Zero-sum properties. We first consider the zero-sum properties $\langle\langle usr_1 \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}^{\leqslant D}\mathtt{sent}_1\,]$ and $\langle\langle usr_1 \rangle\rangle \mathtt{R}^{time}_{\min=?}[\,\mathtt{F}\ \mathtt{sent}_1\,]$ from Example 14, which correspond to the first user trying to maximise the probability that their packet is sent before a deadline and trying to minimise

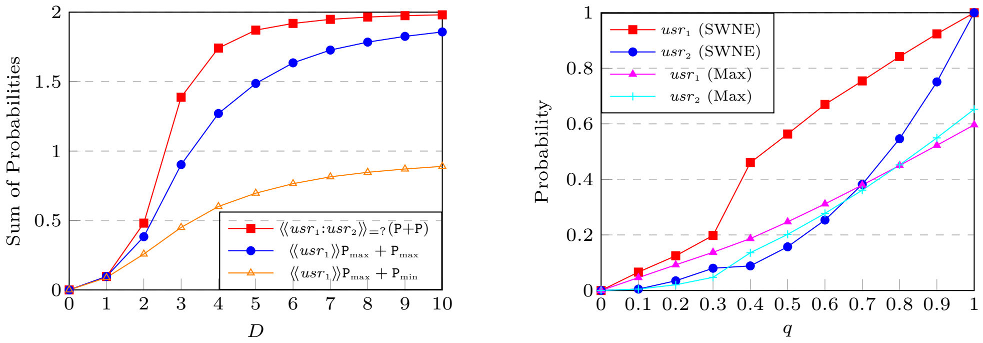


Figure 3 Results from CSG equilibria synthesis on the ALOHA protocol, maximising the probabilities of sending packets by deadline D for two coalitions (user 1, and users 2 and 3): probability sums (left) and individual probabilities (right).

the expected time to send their packet, respectively. The results for the first property when $q = 0.9$ as the deadline D varies, and for the second property as the probability q varies, are presented in Figure 2 for different values of $b_{\max}$. We see that the probability decreases and the expected time decreases as $b_{\max}$ increases; this is because, as $b_{\max}$ increases, the additional time the first user can spend in backoff outweighs the gains in reducing the chance of avoiding further collisions. By performing strategy synthesis we see that it is optimal for the first user to initially randomly decide as to when to send their packet in order to avoid collisions with the coalition of the second and third user. However, this changes to a deterministic strategy of just sending its packet when the other users have sent their packets or the deadline is getting close, and therefore waiting will mean the deadline is missed.

Benefits of equilibria. We next highlight the analysis from [34], which demonstrates the advantages of cooperation through nonzero-sum properties when using Nash equilibria (NE) and the social welfare (SW) optimality criterion, as opposed to adopting a strategy that assumes antagonistic behaviour. The first non-zero sum property we consider corresponds to the case when each user is trying to maximise the probability of sending their packet before a deadline D, with users 2 and 3 forming a coalition, represented by the formula $\langle\langle usr_1{:}usr_2{,}usr_3\rangle\rangle(\text{NE}, \text{SW})_{\max=?}(\mathtt{P}[\,\mathtt{F}\ (\mathsf{sent}_1 \wedge t{\leqslant}D)\,] + \mathtt{P}[\,\mathtt{F}\ (\mathsf{sent}_2 \wedge \mathsf{sent}_3 \wedge t{\leqslant}D)\,])$.

Figure 3 presents total values (the sum of the probabilities for user 1 and the coalition of user 2 and 3) as D varies (left) and individual values as q varies (right). By performing strategy synthesis, the analysis found that the collaboration is dependent on both D and q. In particular, if the users have more time there is a greater chance for the users to collaborate by sending in different slots, whereas, when q is large, it is unlikely users need to repeatedly send, so again can send in different slots. As Figure 3 (right) demonstrates, since the coalition has more packets to send, their probabilities are lower.

Equilibria types and optimality criteria. Finally, we report on the experiments of [35], which investigate the benefits of using different types of equilibria, i.e., *correlated* (CE) over *Nash* equilibria, and optimality criteria, i.e., *social fairness* (SF) over *social welfare* (SW). The experiments varied the number of users and considered the case when the objective of each individual user is to minimise the expected time to send their packet, which is represented by the nonzero-sum formula $\langle\langle usr_1{:}\cdots{:}usr_m\rangle\rangle(\star_1, \star_2)_{\min=?}(\mathtt{R}^{time}[\,\mathtt{F}\ \mathsf{sent}_1\,]{+}\cdots{+}\mathtt{R}^{time}[\,\mathtt{F}\ \mathsf{sent}_m\,])$.

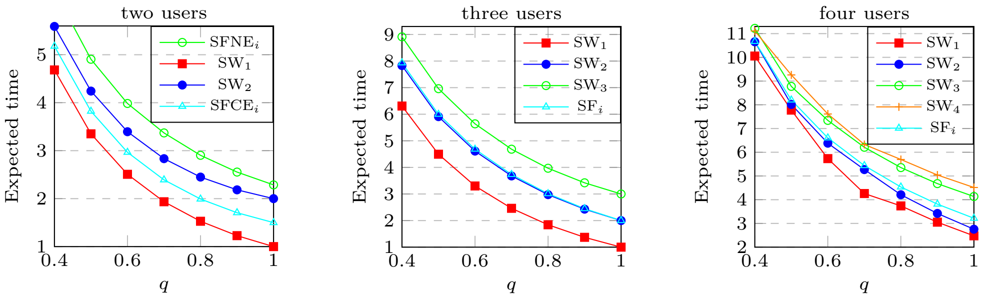


Figure 4 Results from different types of equilibria (correlated vs. Nash) and optimality criteria (social fairness vs. social welfare) for minimising the expected times for users to send packets in the ALOHA protocol., for varying numbers of users.

Synthesising optimal strategies for this specification, it was found that the cases for SWNE and SWCE coincide (although SWCE returns a joint strategy for the users, this joint strategy can be separated to form a strategy profile). This profile required one user to try and send first, and then for the remaining users to take turns to try and send afterwards. If a user fails to send, then they enter backoff and allow all remaining users to try and send before trying to send again. The reason for this is that there is no gain in a user trying to send at the same time as another user, as this will increase the probability of a collision and thus their packets not being sent, and therefore the users having to spend time in backoff.

For SFNE, which has only been implemented for the two-player case, the two users followed identical strategies, which involve randomly deciding whether to wait or transmit, unless they are the only user that has not transmitted, and then they always try to send when not in backoff. In the case of SFCE, users employed a shared probabilistic signal to coordinate which user sends next. Initially, this was a uniform choice over the users, but as time progresses the signal favoured the users with lower backoff counters as these users had fewer opportunities to send their packet previously.

Figure 4 plots the optimal values for the users, where SW$_i$ corresponds to the optimal values (expected times to send their packets) for user i for both SWNE and SWCE for the cases of two, three and four users. We see that the optimal values for the different users under SFNE and SFCE coincide, while under SWNE and SWCE they are different for each user (with the user sending first having the lowest and the user sending last the highest). Comparing the sum of the SWNE (and SWCE) values and that of the SFCE values, we see a small decrease in the sum of less than 2% of the total, whereas for SFNE there is a greater difference as the users cannot coordinate, and hence try and send at the same time.

6 Recent Developments: Neuro-symbolic CSGs

The recent encouraging advances of AI, and particularly deep learning, have resulted in computing architectures that integrate components that are synthesized from data (e.g., implemented as neural networks) with conventional, symbolic modules (e.g., controllers). Design automation support for such *neuro-symbolic* systems is, however, lacking. To this end, we have developed the model of *neuro-symbolic concurrent stochastic games* (NS-CSGs) [60, 59], which is targeted at AI-based autonomous systems, e.g., autonomous driving or aircraft controllers. NS-CSGs are a variant of (continuous-space) CSGs, in which each player is a neuro-symbolic *agent* and the agents act concurrently in a shared, continuous-state environment. As for the players of CSGs, each agent has a finite set of available actions and agents choose their actions simultaneously; however, in NS-CSGs the action choices cause the agents' local states to be updated probabilistically and the agents are endowed with a perception mechanism implemented as a neural network, through which they can observe the local states of the other agents and that of the environment and encode these observations as locally stored *percepts*. The global states of NS-CSGs comprise the state of the environment together with the local state and percept of each agent, and are therefore infinite-state, in contrast to the CSGs discussed in the rest of this paper.

▶ **Definition 15** (Neuro-symbolic concurrent stochastic game [60, 59]). *A* neuro-symbolic concurrent stochastic multi-player game *(NS-CSG)* NSC *comprises players* $(\mathsf{Ag}_i)_{i\in N}$*, for* $N = \{1,\dots,n\}$*, and an environment* E *where:*

$$\mathsf{Ag}_i = (S_i, A_i, \Delta_i, obs_i, \delta_i) \text{ for } i \in N, \quad E = (S_E, \delta_E)$$

and we have:

- $S_i = Loc_i \times Per_i$ *is a set of states for* Ag_i*, where* $Loc_i \subseteq \mathbb{R}^{b_i}$ *and* $Per_i \subseteq \mathbb{R}^{d_i}$ *are finite sets of local states and percepts, respectively;*
- $S_E \subseteq \mathbb{R}^e$ *is a closed infinite set of environment states;*
- A_i *is a nonempty finite set of actions for* Ag_i *and* $A := (A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\})$ *is the set of* joint actions*, where* $\bot$ *is an idle action disjoint from* $\cup_{i=1}^n A_i$*;*
- $\Delta_i : S_i \to 2^{A_i}$ *is an available action function, defining the actions* Ag_i *can take in each state;*
- $obs_i : (Loc_1 \times \cdots \times Loc_n \times S_E) \to Per_i$ *is a perception function for* Ag_i*, mapping the local states of all agents and the environment to a percept of the agent, implemented via a neural network (NN) classifier;*
- $\delta_i : S_i \times A \to \mathbb{P}(Loc_i)$ *is a partial probabilistic transition function for* Ag_i *determining the distribution over the agent's local states given its current state and joint action;*
- $\delta_E : S_E \times A \to S_E$ *is a partial deterministic environment transition function determining the environment's next state given its current state and joint action.*

A (global) state for an NS-CSG NSC comprises a state $s_i = (loc_i, per_i)$ for each agent Ag_i (a pair of a local state and percept) and an environment state s_E. If an NS-CSG is in a state $s = (s_1,\dots,s_n,s_E)$, then each Ag_i simultaneously chooses one of the actions available in its state s_i (if no action is available, i.e., $\Delta_i(s_i) = \varnothing$, it picks the idle action $\bot$) yielding a joint action $\alpha = (a_1,\dots,a_n) \in A$. Next, each Ag_i updates its local state to $loc_i' \in Loc_i$, according to its probabilistic local transition function δ_i, applied to its current state (loc_i, per_i) and the joint action α. The environment updates its state to $s_E' \in S_E$ according to its local deterministic transition function δ_E, based on its state s_E and on α. Finally, each agent, based on its new local state loc_i', observes the new local states of the other agents and environment through its perception function obs_i to generate a new percept $per_i' = obs_i(loc_1',\dots,loc_n',s_E')$. Thus, the game reaches the state $s' = (s_1',\dots,s_n',s_E')$, where $s_i' = (loc_i', per_i')$ for $i \in N$. We assume that each perception function obs_i is implemented via an NN $f_i : \mathbb{R}^{b+e} \to \mathbb{P}(Per_i)$ yielding a normalised score over different percept values, where $b = \sum_{i=1}^n b_i$; however, it can be any function including other types of machine learning models. A rule is then applied that selects the percept value with the maximum score.

Formally, the semantics of an NS-CSG NSC is given by an infinite-state CSG $[\![\mathsf{NSC}]\!]$ over the product of the states of the agents and environment, which assumes a particular structure of the transition function that distinguishes between agent and environment states and uses the NN-based perception function to define which states have the same characteristics.

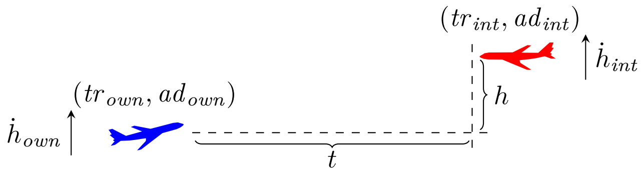


Figure 5 Geometry with trust levels and advisories for the agents of the VCAS[2] case study.

▶ **Definition 16** (Semantics of an NS-CSG). *Given an NS-CSG* NSC *consisting of n players and an environment, the semantics of* NSC *is* $[\![\mathsf{NSC}]\!] = (N, S, A, \Delta, \delta)$ *where:*

- $S = S_1 \times \cdots \times S_n \times S_E$ *is the set of (global) states, which contain both discrete and continuous elements;*
- $A = (A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\})$*;*
- $\Delta(s_1, \ldots, s_n, s_E) = \cup_{i=1}^{n} \Delta_i(s_i)$*;*
- $\delta : (S \times ((A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\}))) \to \mathbb{P}(S)$ *is the partial probabilistic transition function, where for states* $s = (s_1, \ldots, s_n, s_E), s' = (s'_1, \ldots, s'_n, s'_E) \in S$ *and joint action* $\alpha = (a_1, \ldots, a_n) \in A$*, if* $a_i \in \Delta_i(s_i)$ *when* $\Delta_i(s_i) \neq \varnothing$ *and* $a_i = \bot$ *otherwise, then* $\delta(s, \alpha)$ *is defined and if* $s'_i = (loc'_i, per'_i)$*,* $per'_i = obs_i(loc'_1, \ldots, loc'_n, s'_E)$ *for all* $i \in N$ *and* $s'_E = \delta_E(s_E, \alpha)$*, then*

  $$\delta(s, \alpha)(s') = \left(\textstyle\prod_{i=1}^{n} \delta_i(s_i, \alpha)(loc'_i)\right)$$

  and otherwise $\delta(s, \alpha)(s') = 0$.

To illustrate NS-CSGs, we present the VerticalCAS Collision Avoidance Scenario (VCAS[2]) [28, 29], modelled in [59], which is a variant of the one studied in [2], where the agents' trust level is modelled probabilistically to account for possible uncertainty.

▶ **Example 17.** The geometry of the VCAS[2] case study is shown in Figure 5. There are two aircraft (ownship and intruder), constituting the agents of the NS-CSG, both equipped with an NN-controlled collision avoidance system called VCAS. At each time unit (i.e., every second), VCAS issues an advisory ($ad \in \{1, \ldots, 9\}$) from which, together with the current trust level ($tr \in \{1, \ldots, 4\}$) from the previous advisory, the pilot needs to make a decision about the rate of acceleration, aimed at avoiding a near mid-air collision (NMAC) [1].

The input to the VCAS system is the tuple $(h, \dot{h}_{own}, \dot{h}_{int}, t) \in \mathbb{R}^4$ including the relative altitude h of the aircraft, the climb rate $\dot{h}_{own}$ of ownship, the climb rate $\dot{h}_{int}$ of intruder, and the time t until loss of horizontal separation between the aircraft. VCAS is implemented via nine feed-forward NNs $F = \{f_i : \mathbb{R}^4 \to \mathbb{R}^9 \mid 1 \leqslant i \leqslant 9\}$, each of which corresponds to an advisory and outputs the scores of the nine possible advisories. Each advisory will provide a set of accelerations for the agent to select from and the trust level increases probabilistically if the current advisory is compliant with the executed accelerations, and decreases otherwise. We formulate the NS-CSG with agents Ag_i for $i \in \{own, int\}$ as follows:

- the set of states for Ag_i is given by $S_i = \{1, \ldots, 4\} \times \{1, \ldots, 9\}$, where the agent state $s_i = (tr_i, ad_i) \in S_i$ has local state (trust level) tr_i and percept (advisory) ad_i;
- the set of environment states is given by $S_E = \mathbb{R}^4$, where $s_E = (h, \dot{h}_{own}, \dot{h}_{int}, t) \in S_E$ represents the relative altitude, the climb rate of the ownship, the climb rate of the intruder, and the time until loss of their horizontal separation;
- the set of actions of Ag_i is given by $A_i = \{0, \pm 3.0, \pm 7.33, \pm 9.33, \pm 9.7, \pm 11.7\}$ representing the acceleration options of Ag_i;

Table 2 Actions available for the agents of VCAS[2] for each advisory [2].

Label (ad_i)	Advisory	Description	Vertical Range (Min, Max) ft/min	Actions ft/s^2
1	COC	Clear of Conflict	$(-\infty, +\infty)$	-3, $+3$
2	DNC	Do Not Climb	$(-\infty, 0]$	-9.33, -7.33
3	DND	Do Not Descend	$[0, +\infty)$	7.33, $+9.33$
4	DES1500	Descend at least 1500 ft/min	$(-\infty, -1500]$	-9.33, -7.33
5	CL1500	Climb at least 1500 ft/min	$[+1500, +\infty)$	$+7.33$, $+9.33$
6	SDES1500	Strengthen Descend to at least 1500 ft/min	$(-\infty, -1500]$	-11.7, -9.7
7	SCL1500	Strengthen Climb to at least 1500 ft/min	$[+1500, +\infty)$	$+9.7$, $+11.7$
8	SDES2500	Strengthen Descend to at least 2500 ft/min	$(-\infty, -2500]$	-11.7, -9.7
9	SCL2500	Strengthen Climb to at least 2500 ft/min	$[+2500, +\infty)$	$+9.7$, $+11.7$

- the available action function $\Delta_i : Per_i \to A_i$ of Ag_i is independent of the local state of the agent and returns the set consisting two non-zero acceleration actions from Table 2 for a given percept and the zero acceleration action;
- the perception function $obs_i : Per_i \times S_E \to \{1, \dots, 9\}$ of Ag_i is independent of the local state of the agent and is given by the feed forward NNs F of VCAS;
- the local transition function δ_i of Ag_i updates the agent's trust level probabilistically according to its current trust level, its current advisory and its executed action;
- the environment transition function δ_E is given by $\delta_E((h, \dot{h}_{own}, \dot{h}_{int}, t), (\ddot{h}_{own}, \ddot{h}_{int})) = (h', \dot{h}'_{own}, \dot{h}'_{int}, t')$ where for time step $\Delta t = 1$:
  - $h' = h - \Delta t(\dot{h}_{own} - \dot{h}_{int}) - 0.5\Delta t^2(\ddot{h}_{own} - \ddot{h}_{int})$;
  - $\dot{h}'_{own} = \dot{h}_{own} + \ddot{h}_{own}\Delta t$;
  - $\dot{h}'_{int} = \dot{h}_{int} + \ddot{h}_{int}\Delta t$;
  - $t' = t - \Delta t$.

6.1 Zero-sum NS-CSGs

In view of the uncountable state spaces, [60] presents an approach for zero-sum *discounted infinite-horizon* cumulative rewards under the assumption of full state observability for NS-CSGs, which exploits Borel state space decomposition and identifies model restrictions to ensure determinacy, and therefore existence of a value that corresponds to a unique fixed point. Value iteration and policy iteration algorithms to compute values and synthesise optimal strategies are also derived based on formulating piecewise linear or constant representations of the value functions and strategies for NS-CSGs.

6.2 Nonzero-sum NS-CSGs

In the case of nonzero-sum NS-CSGs, [59] studies the *undiscounted, finite-horizon* equilibria synthesis problem. The use of finite-horizon objectives simplifies the analysis (note that the existence of infinite-horizon NE for CSGs is an open problem [7], and the verification of non-probabilistic infinite-horizon reachability properties for neuro-symbolic games is undecidable [2]). Both NE and CE using the SW optimality criteria are considered. The algorithms, based on backward induction and non-linear programming, compute *globally optimal* equilibria which, from a fixed initial state, are optimal over the chosen time horizon, in contrast to the local optimality of equilibria for finite-state CSGs [34, 35].

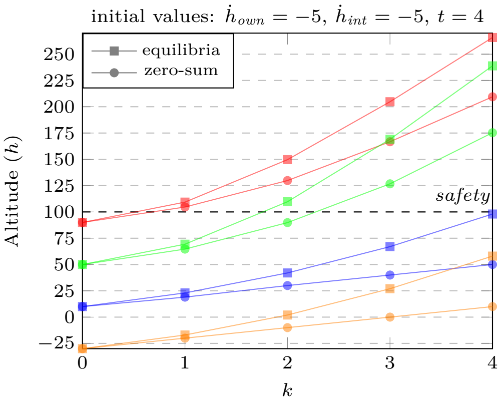


Figure 6 Relative altitude h of the two aircraft at time instants k for equilibria and zero-sum strategies for the VCAS[2] case study.

▶ **Example 18.** The NS-CSG model of the VCAS[2] system described in Example 17 is studied in [59], comparing equilibria strategies to the zero-sum strategies analysed in [2]. Figure 6 plots the relative altitude h of the two aircraft for equilibria and zero-sum strategies when maximising this value for a given time instant k, plotted for several different initial values of h. It can be seen that, with respect to the safety criterion established in [28, 2], i.e., avoiding an NMAC when two aircraft are separated by less than 100 ft vertically (dotted line) and 500 ft horizontally, equilibria strategies allow the two aircraft to reach a safe configuration within a shorter horizon, which would be missed through a zero-sum analysis.

The analysis of [59] also considers a reward structure that incorporates both the trust level and fuel consumption. Figure 7 shows the resulting equilibria strategies when both safety and trust are prioritised, using two different time horizons. When $t = 3$ initially (left), it is optimal to follow the advisories and the trust levels tr_{own} and tr_{int} of the two aircraft never decrease from their initial values of 4. However, when $t = 3$ initially (right), the optimal strategy shows a deviation from the advisory, denoted by action $a_{own} = 0$, in state s_2, resulting in tr_{own} dropping to 3 in s_3 with probability 0.9.

The experiments from [59], summarised above, and from [60] are developed using prototype tools that build upon parts of PRISM-games, but full tool support for NS-CSGs

7 Conclusions and Future Challenges

This paper has provided an overview of modelling, verification and strategy synthesis techniques that have been developed and implemented for concurrent stochastic games in the PRISM-games model checker. Through case studies, we have demonstrated the uses and advantages of zero-sum and equilibria-based reasoning in strategic decision making, highlighting Nash and correlated equlilibria in conjunction with two optimality criteria, social welfare and social fairness. We have also discussed recent trends in autonomous systems towards neural-symbolic architectures, and summarised the first steps towards developing a modelling framework to support the development of such AI-based systems. Despite some progress, many problems remain open in this area, in particular, the development of (efficient) approximate algorithms for (undiscounted infinite-horizon) temporal logic specifications where the underlying problem is undecidable even in the finite-state case [42].

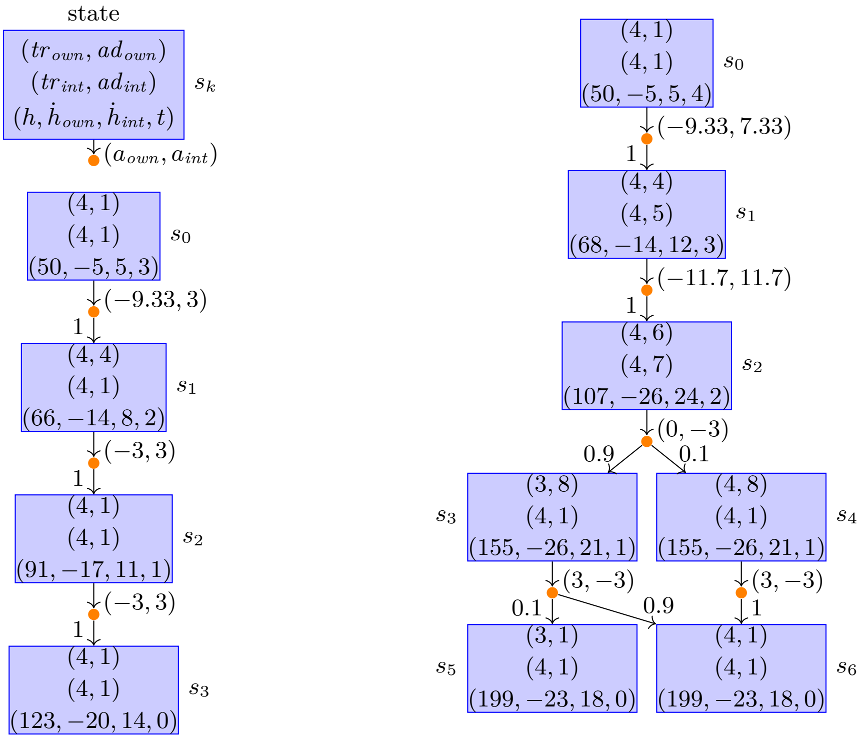

state
(tr_{own}, ad_{own})
(tr_{int}, ad_{int})
$(h, \dot{h}_{own}, \dot{h}_{int}, t)$
s_k
(a_{own}, a_{int})
$(4,1)$ $(4,1)$ $(50,-5,5,3)$ s_0
$(-9.33, 3)$ 1
$(4,4)$ $(4,1)$ $(66,-14,8,2)$ s_1
$(-3,3)$ 1
$(4,1)$ $(4,1)$ $(91,-17,11,1)$ s_2
$(-3,3)$ 1
$(4,1)$ $(4,1)$ $(123,-20,14,0)$ s_3
$(4,1)$ $(4,1)$ $(50,-5,5,4)$ s_0
$(-9.33, 7.33)$ 1
$(4,4)$ $(4,5)$ $(68,-14,12,3)$ s_1
$(-11.7, 11.7)$ 1
$(4,6)$ $(4,7)$ $(107,-26,24,2)$ s_2
$(0,-3)$ 0.9 0.1
s_3 $(3,8)$ $(4,1)$ $(155,-26,21,1)$
$(4,8)$ $(4,1)$ $(155,-26,21,1)$ s_4
$(3,-3)$ 0.1 0.9
$(3,-3)$ 1
s_5 $(3,1)$ $(4,1)$ $(199,-23,18,0)$
$(4,1)$ $(4,1)$ $(199,-23,18,0)$ s_6

Figure 7 Optimal strategies for the VCAS[2] case study over two different time horizons, using initial t values of 3 (left) and 4 (right).

References

1 M. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio. Formal verification of neural agents in non-deterministic environments. In *Proc. AAMAS'20*, pages 25–33. Springer, 2020.

2 M. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio. Verifying Strategic Abilities of Neural-symbolic Multi-agent Systems. In *Proc. KR'20*, pages 22–32. IJCAI Organization, September 2020.

3 R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

4 R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

5 R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 521–525, Vancouver, 1998. Springer.

6 R. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

7 P. Bouyer, N. Markey, and D. Stan. Mixed Nash equilibria in concurrent games. In *Proc. FSTTCS'14*, volume 29 of *LIPICS*, pages 351–363, 2014.

8 R. Brenguier. PRALINE: A tool for computing Nash equilibria in concurrent games. In *Proc. CAV'13*, volume 8044 of *LNCS*, pages 890–895. Springer, 2013. `http://www.lsv.fr/Software/praline/`.

9 T. Brihaye, V. Bruyère, A. Goeminne, J.-F. Raskin, and M. van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. In *Proc. CONCUR'19*, volume 140 of *LIPIcs*, pages 13:1–13:16. Leibniz-Zentrum für Informatik, 2019.

10 P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Proc. CAV'14*, volume 8559 of *LNCS*, pages 525–532. Springer, 2014.

11 K. Chatterjee, L. de Alfaro, and T. Henzinger. Strategy improvement for concurrent reachability and turn-based stochastic safety games. *Journal of Computer and System Sciences*, 79(5):640–657, 2013.

12 K. Chatterjee and T. Henzinger. Value iteration. In *25 Years of Model Checking*, volume 5000 of *LNCS*, pages 107–138. Springer, 2008.

13 K. Chatterjee, T. Henzinger, B. Jobstmann, and A. Radhakrishna. Gist: A solver for probabilistic games. In *Proc. CAV'10*, volume 6174 of *LNCS*, pages 665–669. Springer, 2010. http://pub.ist.ac.at/gist/.

14 K. Chatterjee, R. Majumdar, and M. Jurdziński. On Nash equilibria in stochastic games. In *Proc. CSL'04*, volume 3210 of *LNCS*, pages 26–40. Springer, 2004.

15 T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

16 C. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: An open platform for the research of algorithmic game solving. In *Proc. TACAS'11*, volume 6605 of *LNCS*, pages 258–261. Springer, 2011. https://sourceforge.net/projects/gavsplus/.

17 L. de Alfaro, T. Henzinger, and O. Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.

18 L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68(2):374–397, 2004.

19 L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008. https://github.com/Z3Prover/z3.

20 B. Dutertre. Yices 2.2. In *Proc. CAV'14*, volume 8559 of *LNCS*, pages 737–744. Springer, 2014. http://yices.csl.sri.com.

21 V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *SFM'11*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

22 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. https://www.gurobi.com.

23 J. Gutierrez, M. Najib, P. Giuseppe, and M. Wooldridge. Equilibrium design for concurrent games. In *Proc. CONCUR'19*, volume 140 of *LIPIcs*, pages 22:1–22:16. Leibniz-Zentrum für Informatik, 2019.

24 J. Gutierrez, M. Najib, G. Perelli, and M. Wooldridge. EVE: A tool for temporal equilibrium analysis. In *Proc. ATVA'18*, volume 11138 of *LNCS*, pages 551–557. Springer, 2018. https://github.com/eve-mas/eve-parity.

25 Julian Gutierrez, Lewis Hammond, Anthony W. Lin, Muhammad Najib, and Michael J. Wooldridge. Rational verification for probabilistic systems. In *Proc. 18th International Conference on Principles of Knowledge Representation and Reasoning (KR'21)*, pages 312–322, 2021.

26 H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *FAC*, 6(5):512–535, 1994.

27 L. Hurwicz and S. Reiter. *Designing Economic Mechanisms.* Cambridge University Press, 2006.

28 K. Julian and M. Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers. *CoRR*, abs/1903.00520, 2019. arXiv:1903.00520.

29 K. Julian, S. Sharma, J.-B. Jeannin, and M. Kochenderfer. Verifying aircraft collision avoidance neural networks through linear approximations of safe regions. *CoRR*, abs/1903.00762, 2019. arXiv:1903.00762.

30 N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

31 J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains.* Springer, 1976.

32 M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Multi-player equilibria verification for concurrent stochastic games. In *Proc. QEST'20*, volume 12289 of *LNCS*, pages 74–95. Springer, 2020.

33 M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *Proc. CAV'20*, volume 12225 of *LNCS*, pages 475–487. Springer, 2020.

34 M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Automatic verification of concurrent stochastic systems. *Formal Methods in System Design*, pages 1–63, 2021.

35 M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Correlated equilibria and fairness in concurrent stochastic games. In *Proc. TACAS'22*, volume 13244 of *LNCS*, pages 60–78. Springer, 2022.

36 S. LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26:430–465, 2000.

37 C. Lemke and Jr J. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

38 M. Littman, N. Ravi, A. Talwar, and M. Zinkevich. An efficient optimal-equilibrium algorithm for two-player game trees. In *Proc. UAI'06*, pages 298–305. AUAI Press, 2006.

39 A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. 21st Int. Conf. Computer Aided Verification (CAV'09)*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

40 D. Lozovanu and S. Pickl. Determining Nash equilibria for stochastic positional games with discounted payoffs. In *Proc. ADT'17*, volume 10576 of *LNAI*, pages 339–343. Springer, 2017.

41 LPSolve (version 5.5). `http://lpsolve.sourceforge.net/5.5/`.

42 O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1–2):5–34, 2003.

43 J. Marden and J. Shamma. Game theory and control. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):105–134, 2018.

44 D. Martin. The determinacy of Blackwell games. *J. Symbolic Logic*, 63(4):1565–1581, 1998.

45 R. McKelvey, A. McLennan, and T. Turocy. Gambit: Software tools for game theory. `http://www.gambit-project.org`.

46 N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

47 M. Osborne and A. Rubinstein. *An Introduction to Game Theory*. Oxford University Press, 2004.

48 R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. In *Proc. AAAI'04*, pages 664–669. AAAI Press, 2004.

49 H. Prasad, L. Prashanth, and S. Bhatnagar. Two-timescale algorithms for learning Nash equilibria in general-sum stochastic games. In *Proc. AAMAS'15*, pages 1371–1379. IFAAMAS, 2015.

50 E. Prisner. *Game Theory Through Examples*. Mathematical Association of America, 1 edition, 2014.

51 T. Raghavan and J. Filar. Algorithms for stochastic games – A survey. *Zeitschrift für Operations Research*, 35(6):437–472, November 1991.

52 T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proc. AAAI'05*, pages 495–501. AAAI Press, 2005.

53 U. Schwalbe and P. Walker. Zermelo and the early history of game theory. *Games and Economic Behavior*, 34(1):123–137, 2001.

54 L. Shapley. Stochastic games. *PNAS*, 39:1095–1100, 1953.

55 A. Toumi, J. Gutierrez, and M. Wooldridge. A tool for the automated verification of Nash equilibria in concurrent games. In *Proc. ICTAC'15*, volume 9399 of *LNCS*, pages 583–594. Springer, 2015.

56 J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

57 J. von Neumann, O. Morgenstern, H. Kuhn, and A. Rubinstein. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

58 A. Wächter. Short tutorial: Getting started with IPOPT in 90 minutes. In *Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings. Leibniz-Zentrum für Informatik, 2009. `https://github.com/coin-or/Ipopt`.

59 R. Yan, G. Santos, X. Duan, D. Parker, and M. Kwiatkowska. Finite-horizon equilibria for neuro-symbolic concurrent stochastic games. In *Proc. UAI'22*, 2022.

60 R. Yan, G. Santos, G. Norman, D. Parker, and M. Kwiatkowska. Strategy synthesis for zero-sum neuro-symbolic concurrent stochastic games, 2022. `arXiv:2202.06255`.

61 PRISM-games web site. `http://www.prismmodelchecker.org/games/`.

Online Bipartite Matching and Adwords

Vijay V. Vazirani ✉
University of California, Irvine, CA, USA

Abstract

The purpose of this paper is to give a "textbook quality" proof of the optimal algorithm, called Ranking, for the online bipartite matching problem (OBM) and to highlight its role in matching-based market design. In particular, we discuss a generalization of OBM, called the adwords problem, which has had a significant impact in the ad auctions marketplace.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases matching-based market design, online algorithms, ad auctions, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.5

Category Invited Talk

Funding NSF Grant CCF-2230414

1 Introduction

The online bipartite matching problem[1] (OBM) occupies a central place not only in online algorithms but also in matching-based market design, see details in Sections 1.1 and 1.2. The purpose of this paper is to give a "textbook quality" proof[2] of the optimal algorithm, called Ranking, for this problem and to highlight its role in matching-based market design. In particular, we discuss a generalization of OBM, called the adwords problem, which has had a significant impact in the ad auctions marketplace, see Section 1.2.

Ranking achieves a competitive ratio of $\left(1 - \frac{1}{e}\right)$ [17]. Its analysis, given in [17], was considered "difficult" and it also had an error. Over the years, several researchers contributed valuable ideas to simplifying its proof, see Section 1.1 for details. The proof given in this paper is based on these ideas. Additionally, we highlight a key property used in the proof, called the *No-Surpassing Property* and simplify further its proof. This property turns out to be the bottleneck to a substantial generalization which was attempted in [21], as described below.

The adwords problem, which is called General in this paper, is a generalization of OBM. It involves matching keyword queries, as they arrive online, to advertisers; the latter have daily budget limits and they make bids for the queries. The overall goal is to maximize the total revenue. This problem is notoriously difficult and has remained largely unsolved; see Section 1.1 for marginal progress made recently. Its special case, when bids are small compared to budgets, called Small, captures a key computational issue that arises in the context of ad auctions, for instance in Google's AdWords marketplace. An optimal algorithm for Small, achieving a competitive ratio of $\left(1 - \frac{1}{e}\right)$, was first given in [19]; for the impact of this result in the marketplace, see Section 1.2.

In Open Problem Number 20 in [18], Mehta asks for a *budget-oblivious online algorithm* for Small. Such an algorithm does not know the daily budgets of advertisers; however, in a run of the algorithm, it knows when the budget of an advertiser is exhausted. However, its revenue is still compared to the optimal revenue generated by an offline algorithm with full

[1] For formal statements of problems discussed in this paper, see Section 2.
[2] e.g., the proof given in the chapter [8] of the upcoming edited book on matching-based market design.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 5; pp. 5:1–5:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

knowledge of the budget. Its importance lies in its use in autobidding platforms [1, 6], which manage the ad campaigns of large advertisers; they dynamically adjust bids and budgets over multiple search engines to improve performance. The greedy algorithm, which matches an arriving query to the advertiser making the highest bid, is clearly budget-oblivious; its competitive ratio is 0.5. An improved algorithm, having a competitive ratio of 0.522, was recently obtained by Udwani [20], using the idea of an LP-free analysis, which involves writing appropriate linear inequalities to compare the online algorithm with the offline optimal algorithm.

Motivated by the recent simplification of the proof of (OBM), [21] attempted to extend Ranking all the way to Small. This attempt represents a more basic approach to Small than the one used in [19] (see Section 1.1) and the hope was that it would yield an algorithm with better properties, e.g., budget-obliviousness. [21] managed to extend Ranking to an intermediate problem, called Single-Valued, thereby giving an optimal, budget-oblivious algorithm; see Section 1.1 for competing results for this problem. Under Single-Valued, each advertiser can make bids of one value only, although the value may be different for different advertisers.

The analysis of Single-Valued given in [21] involved new ideas from two domains, namely probability theory and combinatorics, with the former playing a dominant role and the latter yielding a proof of the No-Surpassing Property for Single-Valued. Equipped with these new ideas, [21] next attempted an extension from Ranking to Small. Although the more difficult, probabilistic part, of the argument did extend, a counter-example was found to the combinatorial part, showing that the No-Surpassing Property does not hold for Small.

1.1 Related Works

We start by stating simplifications to the proof of OBM. At first, [11, 4], got the ball rolling, setting the stage for the substantial simplification given in [7], using a randomized primal-dual approach. [7] introduced the idea of splitting the contribution of each matched edge into primal and dual contributions and lower-bounding each part separately. Their method for defining prices p_j of goods, using randomization, was used by subsequent papers, including this one[3].

Interestingly enough, the next simplification involved removing the scaffolding of LP-duality and casting the proof in purely probabilistic terms[4], using notions from economics to split the contribution of each matched edge into the contributions of the buyer and the seller. This elegant analysis was given by [9]. A further simplification to the proof of the No-Surpassing Property for OBM is given in the current paper.

An important generalization of OBM is online b-matching. This problem is a special case of General in which the budget of each advertiser is \$$b$ and the bids are 0/1. [16] gave a simple optimal online algorithm, called BALANCE, for this problem. BALANCE awards the next query to the interested bidder who has been matched least number of times so far. [16] showed that as b tends to infinity, the competitive ratio of BALANCE tends to $\left(1 - \frac{1}{e}\right)$.

Observe that b-matching is a special case of Small, if b is large. Indeed, the first online algorithm [19] for Small was obtained by extending BALANCE as follows: [19] first gave a simpler proof of the competitive ratio of BALANCE using the notion of a *factor-revealing*

[3] For a succinct proof of optimality of the underlying function, e^{x-1}, see Section 2.1.1 in [12].

[4] Even though there is no overt use of LP-duality in the proof of [9], it is unclear if this proof could have been obtained directly, without going the LP-duality-route.

LP [15]. Then they gave the notion of a *tradeoff-revealing LP*, which yielded an algorithm achieving a competitive ratio of $\left(1 - \frac{1}{e}\right)$. [19] also proved that this is optimal for b-matching, and hence Small, by proving that no randomized algorithm can achieve a better ratio for online b-matching; previously, [16] had shown a similar result for deterministic algorithms.

The algorithm of [19] is simple and operates as follows. The effective bid of each bidder j for a query is its bid multiplied by $(1 - e^{L_j/B_j})$, where B_j and L_j are the total budget and the leftover budget of bidder j, respectively; the query is matched to the bidder whose effective bid is highest. As a result, the algorithm of [19] needs to know the total budget of each bidder. Following [19], a second optimal online algorithm for Small was given in [5], using a primal-dual approach.

Another relevant generalization of OBM is online vertex weighted matching, in which the offline vertices have weights and the objective is to maximize the weight of the matched vertices. [2] extended Ranking to obtain an optimal online algorithm for this problem. Clearly, Single-Valued is intermediate between General and online vertex weighted matching. [2] gave an optimal online algorithm for Single-Valued by reducing it to online vertex weighted matching. This involved creating k_j copies of each advertiser j. As a result, their algorithm needs to use $\sum_{j \in A} k_j$ random numbers, where A is the set of advertisers.

We note that independent of [21], Albers and Schubert [3] had also obtained an optimal, budget-oblivious algorithm for Single-Valued; however, their technique was different and involved formulating a configuration LP and conducting a primal-dual analysis. Another advantage of the algorithms of [3] and [21], in contrast to [2], was that they need to use only $|A|$ random numbers.

For General, the greedy algorithm, which matches each query to the highest bidder, achieves a competitive ratio of 1/2. Until recently, that was the best possible. In [13] a marginally improved algorithm, with a ratio of 0.5016, was given. It is important to point out that this 60-page paper was a tour-de-force, drawing on a diverse collection of ideas – a testament to the difficulty of this problem.

In the decade following the conference version (FOCS 2005) of [19], search engine companies generously invested in research on models derived from OBM and adwords. Their motivation was two-fold: the substantial impact of [19] and the emergence of a rich collection of digital ad tools. It will be impossible to do justice to this substantial body of work, involving both algorithmic and game-theoretic ideas; for a start, see the surveys [18, 12].

1.2 Significance and Practical Impact

Google's AdWords marketplace generates multi-billion dollar revenues annually and the current annual worldwide spending on digital advertising is almost half a trillion dollars. These revenues of Google and other Internet services companies enable them to offer crucial services, such as search, email, videos, news, apps, maps etc. for free – services that have virtually transformed our lives.

We note that Small is the most relevant case of adwords for the search ads marketplace e.g., see [6]. A remarkable feature of Google, and other search engines, is the speed with which they are able to show search results, often in milliseconds. In order to show ads at the same speed, together with search results, the solution for Small needed to be minimalistic in its use of computing power, memory and communication.

The online algorithm of [19] satisfied these criteria and therefore had a substantial impact in this marketplace. Furthermore, the idea underlying their algorithm was extracted into a simple heuristic, called *bid scaling*, which uses even less computation and is widely used by search engine companies today. As mentioned above, our Conditional Algorithm for Small is even more elementary and is budget-oblivious.

It will be useful to view the AdWords marketplace in the context of a bigger revolution, namely the advent of the Internet and mobile computing, and the consequent resurgence of the area of matching-based market design. The birth of this area goes back to the seminal 1962 paper of Gale and Shapley on stable matching [10]. Over the decades, this area became known for its highly successful applications, having economic as well as sociological impact. These included matching medical interns to hospitals, students to schools in large cities, and kidney exchange.

The resurgence led to a host of highly innovative and impactful applications. Besides the AdWords marketplace, which matches queries to advertisers, these include Uber, matching drivers to riders; Upwork, matching employers to workers; and Tinder, matching people to each other, see [14] for more details.

A successful launch of such markets calls for economic and game-theoretic insights, together with algorithmic ideas. The Gale-Shapley deferred acceptance algorithm and its follow-up works provided the algorithmic backbone for the "first life" of matching-based market design. The algorithm RANKING has become the paradigm-setting algorithmic idea in the "second life" of this area. Interestingly enough, this result was obtained in the pre-Internet days, over thirty years ago.

2 Preliminaries

Online Bipartite Matching. (OBM): Let B be a set of n buyers and S a set of n goods. A bipartite graph $G = (B, S, E)$ is specified on vertex sets B and S, and edge set E, where for $i \in B$, $j \in S$, $(i, j) \in E$ if and only if buyer i *likes* good j. G is assumed to have a perfect matching and therefore each buyer can be given a unique good she likes. Graph G is revealed in the following manner. The n goods are known up-front. On the other hand, the buyers arrive one at a time, and when buyer i arrives, the edges incident at i are revealed.

We are required to design an online algorithm $\mathcal{A}$ in the following sense. At the moment a buyer i arrives, the algorithm needs to match i to one of its unmatched neighbors, if any; if all of i's neighbors are matched, i remains unmatched. The difficulty is that the algorithm does not "know" the edges incident at buyers which will arrive in the future and yet the size of the matching produced by the algorithm will be compared to the best *off-line matching*; the latter of course is a perfect matching. The formal measure for the algorithm is defined in Section 2.1.

General Adwords Problem (GENERAL): Let A be a set of m *advertisers*, also called *bidders*, and Q be a set of n *queries*. A bipartite graph $G = (Q, A, E)$ is specified on vertex sets Q and A, and edge set E, where for $i \in Q$ and $j \in A$, $(i, j) \in E$ if and only if bidder j is *interested in* query i. Each query i needs to be matched[5] to at most one bidder who is interested in it. For each edge (i, j), bidder j *knows* his bid for i, denoted by $\text{bid}(i, j) \in \mathbb{Z}_+$. Each bidder also has a *budget* $B_j \in \mathbb{Z}_+$ which satisfies $B_j \geq \text{bid}(i, j)$, for each edge (i, j) incident at j.

Graph G is revealed in the following manner. The m bidders are known up-front and the queries arrive one at a time. When query i arrives, the edges incident at i are revealed, together with the bids associated with these edges. If i gets matched to j, then the matched edge (i, j) is assigned a weight of $\text{bid}(i, j)$. The constraint on j is that the total weight of matched edges incident at it be at most B_j. The objective is to maximize the total weight of all matched edges at all bidders.

[5] Clearly, this is not a matching in the usual sense, since a bidder may be matched to several queries.

Adwords under Single-Valued Bidders (Single-Valued): Single-Valued is a special case of General in which each bidder j will make bids of a single value, $b_j \in \mathbb{Z}_+$, for the queries he is interested in. If i accepts j's bid, then i will be matched to j and the weight of this matched edge will be b_j. Corresponding to each bidder j, we are also given $k_j \in \mathbb{Z}_+$, the maximum number of times j can be matched to queries. The objective is to maximize the total weight of matched edges. Observe that the matching M found in G is a b-matching with the b-value of each query i being 1 and of advertiser j being k_j.

Adwords under Small Bids (Small): Small is a special case of General in which for each bidder j, each bid of j is small compared to its budget. Formally, we will capture this condition by imposing the following constraint. For a valid instance I of Small, define

$$\mu(I) = \max_{j \in A} \left\{ \frac{\max_{(i,j)\in E} \{\text{bid}(i,j) - 1\}}{B_j} \right\}.$$

Then we require that

$$\lim_{n(I)\to\infty} \mu(I) = 0,$$

where $n(I)$ denotes the number of queries in instance I.

2.1 The competitive ratio of online algorithms

We will define the notion of competitive ratio of a randomized online algorithm in the context of OBM.

▶ **Definition 1.** *Let $G = (B, S, E)$ be a bipartite graph as specified above. The competitive ratio of a randomized algorithm $\mathcal{A}$ for OBM is defined to be:*

$$c(\mathcal{A}) = \min_{G=(B,S,E)} \min_{\rho(B)} \frac{\mathbb{E}[\mathcal{A}(G, \rho(B))]}{n},$$

where $\mathbb{E}[\mathcal{A}(G, \rho(B))]$ is the expected size of matching produced by $\mathcal{A}$; the expectation is over the random bits used by $\mathcal{A}$. We may assume that the worst case graph and the order of arrival of buyers, given by $\rho(B)$, are chosen by an adversary who knows the algorithm. It is important to note that the algorithm is provided random bits after *the adversary makes its choices.*

▶ Remark 2. For each problem studied in this paper, we will assume that the offline matching is complete. It is easy to extend the arguments, without changing the competitive ratio, in case the offline matching is not complete. As an example, this is done for OBM in Remark 14.

3 Ranking and its Analysis

Algorithm 1 presents an optimal algorithm for OBM. Note that this algorithm picks a random permutation of goods only once. Its competitive ratio is $(1 - \frac{1}{e})$, as shown in Theorem 13. Furthermore, as shown in [17], it is an optimal online bipartite matching algorithm: no randomized algorithm can do better, up to an $o(1)$ term.

We will analyze Algorithm 2 which is equivalent to Algorithm 1 and operates as follows. Before the execution of Step (1), the adversary determines the order in which buyers will arrive, say $\rho(B)$. In Step (1), each good j is assigned a *price* $p_j = e^{w_j - 1}$, where w_j, called the *rank* of j, is picked at random from $[0, 1]$; observe that $p_j \in [\frac{1}{e}, 1]$. In Step (2), buyers

Algorithm 1 Ranking.

```
1. Initialization: Pick a random permutation, $\pi$, of the goods in $S$.

2. Online buyer arrival: When a buyer, say $i$, arrives, match her to the first
   unmatched good she likes in the order $\pi$; if none, leave $i$ unmatched.

Output the matching, $M$, found.
```

will arrive in the order $\rho(B)$, picked by the adversary, and will be matched to the cheapest available good. With probability 1 all n prices are distinct and sorting the goods by increasing prices results in a random permutation. Furthermore, since Algorithm 2 uses this sorted order only and is oblivious of the actual prices, it is equivalent to Algorithm 1. As we will see, the random variables representing actual prices are crucially important as well – in the analysis. We remark that for the generalizations of OBM studied in this paper, the prices are used not only in the analysis, but also by the algorithms.

3.1 Analysis of Ranking

We will use an *economic setting* for analyzing Algorithm 2 as follows. Each buyer i has *unit-demand* and *0/1 valuations* over the goods she likes, i.e., she accrues unit utility from each good she likes, and she wishes to get at most one of them. The latter set is precisely the set of neighbors of i in G. If on arrival of i there are several of these which are still unmatched, i will pick one having the smallest price [6]. Therefore the buyers will maximize their utility as defined below.

For analyzing this algorithm, we will define two sets of random variables, u_i for $i \in B$ and r_j, for $j \in S$. These will be called utility of buyer i and revenue of good j, respectively. Each run of Ranking defines these random variables as follows. If Ranking matches buyer i to good j, then define $u_i = 1 - p_j$ and $r_j = p_j$, where p_j is the price of good j in this run of Ranking. Clearly, p_j is also a random variable, which is defined by Step (1) of the algorithm. If i remains unmatched, define $u_i = 0$, and if j remains unmatched, define $r_j = 0$. Observe that for each good j, $p_j \in [\frac{1}{e}, 1]$ and for each buyer i, $u_i \in [0, 1 - \frac{1}{e}]$. Let M be the matching produced by Ranking and let random variable $|M|$ denote its size.

Lemma 3 pulls apart the contribution of each matched edge (i, j) into u_i and r_j. Next, we established in Lemma 11 that for each edge (i, j) in the graph, the total expected contribution of u_i and r_j is at least $1 - \frac{1}{e}$. Then, linearity of expectation allows us to reassemble the $2n$ terms in the right hand side of Lemma 3 so they are aligned with a perfect matching in G, and this yields Theorem 13.

▶ **Lemma 3.**

$$\mathbb{E}[|M|] \; = \sum_{i}^{n} \mathbb{E}\left[u_i\right] \; + \; \sum_{j}^{n} \mathbb{E}[r_j].$$

[6] As stated above, with probability 1 there are no ties.

Algorithm 2 Ranking: Economic Viewpoint.

```
1. Initialization: $\forall j \in S$:   Pick $w_j$ independently and uniformly from $[0,1]$.
   Set price  $p_j \leftarrow e^{w_j - 1}$.

2. Online buyer arrival: When a buyer, say $i$, arrives, match her to the cheapest
   unmatched good she likes; if none, leave $i$ unmatched.

Output the matching, $M$, found.
```

Proof. By definition of the random variables,

$$\mathbb{E}[|M|] = \mathbb{E}\left[\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} r_j\right] = \sum_{i}^{n} \mathbb{E}[u_i] + \sum_{j}^{n} \mathbb{E}[r_j],$$

where the first equality follows from the fact that if $(i,j) \in M$ then $u_i + r_j = 1$ and the second follows from linearity of expectation. ◀

While running Algorithm 2, assume that the adversary has picked the order of arrival of buyers, say $\rho(B)$, and Step (1) has been executed. We next define several ways of executing Step (2). Let $\mathcal{R}$ denote the run of Step (2) on the entire graph G. Corresponding to each good j, let G_j denote graph G with vertex j removed. Define $\mathcal{R}_j$ to be the run of Step (2) on graph G_j.

Lemma 4 and Corollary 5 establish a relationship between the sets of available goods for a buyer i in the two runs $\mathcal{R}$ and $\mathcal{R}_j$; the latter is crucially used in the proof of Lemma 9. For ease of notation in proving these two facts, let us renumber the buyers so their order of arrival under $\rho(B)$ is $1, 2, \ldots n$. Let $T(i)$ and $T_j(i)$ denote the sets of unmatched goods at the time of arrival of buyer i (i.e., just before the buyer i gets matched) in the graphs G and G_j, in runs $\mathcal{R}$ and $\mathcal{R}_j$, respectively. Similarly, let $S(i)$ and $S_j(i)$ denote the set of unmatched goods that buyer i is incident to in G and G_j, in runs $\mathcal{R}$ and $\mathcal{R}_j$, respectively.

We have assumed that Step (1) of Algorithm 2 has already been executed and a price p_k has been assigned to each good k. With probability 1, the prices are all distinct. Let F_1 and F_2 be subsets of S containing goods k such that $p_k < p_j$ and $p_k > p_j$, respectively.

▶ **Lemma 4.** *For each i, $1 \le i \le n$, the following hold:*

1. $(T_j(i) \cap F_1) = (T(i) \cap F_1)$.
2. $(T_j(i) \cap F_2) \subseteq (T(i) \cap F_2)$.

Proof. Clearly, in both runs, $\mathcal{R}$ and $\mathcal{R}_j$, any buyer having an available good in F_1 will match to the most profitable one of these, without even considering the rest of the goods. Since $j \notin F_1$, the two runs behave in an identical manner on the set F_1, thereby proving the first statement.

The proof of the second statement is by induction on i. The base case is trivially true since $j \notin F_2$. Assume that the statement is true for $i = k$ and let us prove it for $i = k + 1$. By the first statement, we need to consider only the case that there are no available goods for the k^{th} buyer in F_1 in the runs $\mathcal{R}$ and $\mathcal{R}_j$. Assume that in run $\mathcal{R}_j$, this buyer gets matched to good l; if she remains unmatched, we will take l to be null. Clearly, l is the most profitable good she is incident to in $T_j(k)$. Therefore, the most profitable good she is incident to in run $\mathcal{R}$ is the best of l, the most profitable good in $T(k) - T_j(k)$, and j, in case it is available. In each of these cases, the induction step holds. ◀

In the corollary below, the first two statements follow from Lemma 4 and the third statement follows from the first two.

▶ **Corollary 5.** *For each i, $1 \leq i \leq n$, the following hold:*
1. $(S_j(i) \cap F_1) = (S(i) \cap F_1)$.
2. $(S_j(i) \cap F_2) \subseteq (S(i) \cap F_2)$.
3. $S_j(i) \subseteq S(i)$.

Next we define a new random variable, u_e, for each edge $e = (i, j) \in E$. This is called the *threshold* for edge e and is given in Definition 6. It is critically used in the proofs of Lemmas 9 and 11.

▶ **Definition 6.** *Let $e = (i, j) \in E$ be an arbitrary edge in G. Define random variable, u_e, called the* threshold *for edge e, to be the utility of buyer i in run $\mathcal{R}_j$. Clearly, $u_e \in [0, 1 - \frac{1}{e}]$.*

▶ **Property 7** (No-Surpassing for OBM). *Let p_j be such that the bid of j, namely $1 - p_j$, is better than the best bid that buyer i gets in run $\mathcal{R}_j$. Then, in run $\mathcal{R}$, no bid to i will surpass $1 - p_j$.*

▶ **Lemma 8.** *The No-Surpassing Property holds for OBM.*

Proof. Suppose the bid of j, namely $1 - p_j$, is better than the best bid that buyer i gets in run $\mathcal{R}_j$. If so, i gets no bid from F_1 in $\mathcal{R}_j$; observe that they are all higher than $1 - p_j$. Now, by the first part of Corollary 5, i gets no bid from F_1 in run $\mathcal{R}$ as well, i.e., in run $\mathcal{R}$, no bid to i will surpass $1 - p_j$. ◀

▶ **Lemma 9.** *Corresponding to each edge $(i, j) \in E$, the following hold.*
1. *$u_i \geq u_e$, where u_i and u_e are the utilities of buyer i in runs $\mathcal{R}$ and $\mathcal{R}_j$, respectively.*
2. *Let $z \in [0, 1 - \frac{1}{e}]$. Conditioned on $u_e = z$, if $p_j < 1 - z$, then j will definitely be matched in run $\mathcal{R}$.*

Proof.

1). By the third statement of Corollary 5, i has more options in run $\mathcal{R}$ as compared to run $\mathcal{R}_j$, and therefore $u_i \geq u_e$.

2). In run $\mathcal{R}$, if j is already matched when i arrives, there is nothing to prove. Next assume that j is not matched when i arrives. The crux of the matter is that by Lemma 8, the No-Surpassing Property holds. Therefore, in run $\mathcal{R}$, i will not have any option that is better than j and will therefore get matched to j. Since $1 - p_j > z$, $S_j(i) \cap F_1 = \emptyset$. Therefore by the first statement of Corollary 5, $S(i) \cap F_1 = \emptyset$. Since i will get no bid better than j in $\mathcal{R}$, the no-surpassing property indeed holds and i must get matched to j. ◀

▶ Remark 10. The random variable u_e is called *threshold* because of the second statement of Lemma 9. It defines a value such that whenever p_j is smaller than this value, j is definitely matched in run $\mathcal{R}$.

The intuitive reason for the next, and most crucial, lemma is the following. The smaller u_e is, the larger is the range of values for p_j, namely $[0, 1 - u_e)$, over which (i, j) will be matched and j will accrue revenue of p_j. Integrating p_j over this range, and adding $\mathbb{E}[u_i]$ to it, gives the desired bound. Crucial to this argument is the fact that p_j is independent of u_e. This follows from the fact that u_e is determined by run $\mathcal{R}_j$ on graph G_j, which does not contain vertex j.

▶ **Lemma 11.** *Corresponding to each edge $(i, j) \in E$,*

$$\mathbb{E}[u_i + r_j] \geq 1 - \frac{1}{e}.$$

Proof. By the first part of Lemma 9, $\mathbb{E}[u_i] \geq \mathbb{E}[u_e]$.

Next, we will lower bound $\mathbb{E}[r_j]$. Let $z \in [0, 1 - \frac{1}{e}]$ and let us condition on the event $u_e = z$. The critical observation is that u_e is determined by the run $\mathcal{R}_j$. This is conducted on graph G_j, which does not contain vertex j. Therefore u_e is independent of p_j.

By the second part of Lemma 9, $r_j = p_j$ whenever $p_j < 1 - z$. We will ignore the contribution to $\mathbb{E}[r_j]$ when $p_j \geq 1 - z$. Let w be s.t. $e^{w-1} = 1 - z$.

Now p_j is obtained by picking x uniformly at random from the interval $[0, 1]$ and outputting e^{x-1}. In particular, when $x \in [0, w)$, $p_j < 1 - z$. If so, by the second part of Lemma 9, j is matched and revenue is accrued in r_j, see Figure 2. Therefore,

$$\mathbb{E}[r_j \mid u_e = z] \geq \int_0^w e^{x-1}\, dx \;=\; e^{w-1} - \frac{1}{e} \;=\; 1 - \frac{1}{e} - z.$$

Let $f_{u_e}(z)$ be the probability density function of u_e; clearly, $f_{u_e}(z) = 0$ for $z \notin [0, 1 - \frac{1}{e}]$. Therefore,

$$\mathbb{E}[r_j] \;=\; \mathbb{E}[\mathbb{E}[r_j \mid u_e]] \;=\; \int_{z=0}^{1-1/e} \mathbb{E}[r_j \mid u_e = z] \cdot f_{u_e}(z) dz$$

$$\geq \int_{z=0}^{1-1/e} \left(1 - \frac{1}{e} - z\right) \cdot f_{u_e}(z) dz \;=\; 1 - \frac{1}{e} - \mathbb{E}[u_e],$$

where the first equality follows from the law of total expectation and the inequality follows from fact that we have ignored the contribution to $\mathbb{E}[r_j \mid u_e]$ when $p_j \geq 1 - z$. Hence we get

$$\mathbb{E}[u_i + r_j] = \mathbb{E}[u_i] + \mathbb{E}[r_j] \geq 1 - \frac{1}{e}.$$ ◀

▶ Remark 12. Observe that Lemma 11 is not a statement about i and j getting matched to each other, but about the utility accrued by i and the revenue accrued by j by being matched to various goods and buyers, respectively, over the randomization executed in Step (1) of Algorithm 2.

▶ **Theorem 13.** *The competitive ratio of* Ranking *is at least $1 - \frac{1}{e}$.*

Proof. Let P denote a perfect matching in G. The expected size of matching produced by Ranking is

$$\mathbb{E}\left[|M|\right] \;=\; \sum_i^n \mathbb{E}\left[u_i\right] \;+\; \sum_j^n \mathbb{E}[r_j] \;=\; \sum_{(i,j) \in P} \mathbb{E}[u_i + r_j] \;\geq\; n\left(1 - \frac{1}{e}\right),$$

where the first equality uses Lemma 3, the second follows from linearity of expectation and the inequality follows from Lemma 11 and the fact that $|P| = n$. The theorem follows. ◀

▶ Remark 14. In case G does not have a perfect matching, let P denote a maximum matching in G, of size k, say. Then summing $\mathbb{E}\left[u_i\right]$ and $\mathbb{E}[r_j]$ over the the vertices i and j matched by P, we get that the expected size of matching produced by Ranking is at least $k\left(1 - \frac{1}{e}\right)$.

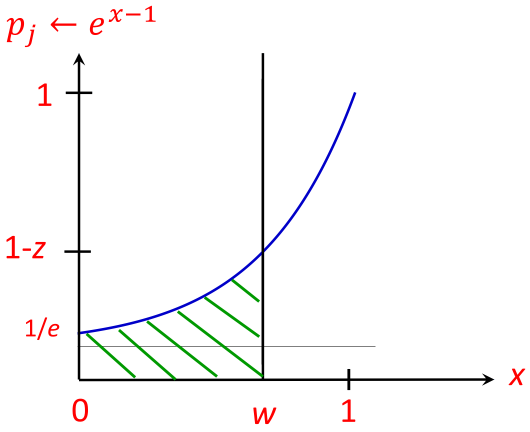


Figure 2 The shaded area is a lower bound on $\mathbb{E}[r_j \mid u_e = z]$.

References

1 Gagan Aggarwal, Ashwinkumar Badanidiyuru, and Aranyak Mehta. Autobidding with constraints. In *International Conference on Web and Internet Economics*, pages 17–30. Springer, 2019.

2 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1253–1264, 2011.

3 Susanne Albers and Sebastian Schubert. Optimal algorithms for online b-matching with variable vertex capacities. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

4 Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *ACM Sigact News*, 39(1):80–87, 2008.

5 Niv Buchbinder, Kamal Jain, and Joseph Seffi Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *European Symposium on Algorithms*, pages 253–264, 2007.

6 Nikhil Devanur and Aranyak Mehta. Online matching in advertisement auctions. In Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors, *Online and Matching-Based Market Design*. Cambridge University Press, 2022. [To appear] `https://www.ics.uci.edu/~vazirani/AdAuctions.pdf`.

7 Nikhil R Devanur, Kamal Jain, and Robert D Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 101–107. SIAM, 2013.

8 Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani. One-sided matching markets. In Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors, *Online and Matching-Based Market Design*. Cambridge University Press, 2022. [To appear] `https://www.ics.uci.edu/~vazirani/Chapter2.pdf`.

9 Alon Eden, Michal Feldman, Amos Fiat, and Kineret Segal. An economic-based analysis of ranking for online bipartite matching. In *SIAM Symposium on Simplicity in Algorithms*, 2021.

10 David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

11 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, volume 8, pages 982–991, 2008.

12 Zhiyi Huang and Thorben Trobst. Online matching. In Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors, *Online and Matching-Based Market Design*. Cambridge University Press, 2022. [To appear] `https://www.ics.uci.edu/~vazirani/Ch4.pdf`.

13 Zhiyi Huang, Qiankun Zhang, and Yuhao Zhang. Adwords in a panorama. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1416–1426. IEEE, 2020.

14 Simons Institute. Online and matching-based market design, 2019. URL: `https://simons.berkeley.edu/programs/market2019`.

15 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.

16 Bala Kalyanasundaram and Kirk R Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1-2):319–325, 2000.

17 Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.

18 Aranyak Mehta. *Online matching and ad allocation*, volume 8(4). Now Publishers, Inc., 2013.

19 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5), 2007.

20 Rajan Udwani. Adwords with unknown budgets. *arXiv preprint*, 2021. `arXiv:2110.00504`.

21 Vijay V Vazirani. Online bipartite matching and adwords. *arXiv preprint*, 2021. `arXiv:2107.10777`.

Parameterized Complexity of Non-Separating and Non-Disconnecting Paths and Sets

Ankit Abhinav ✉
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, Odisha, India

Susobhan Bandopadhyay ✉
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, Odisha, India

Aritra Banik ✉
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, Odisha, India

Yasuaki Kobayashi ✉
Kyoto University, Kyoto, Japan

Shunsuke Nagano ✉
Kyoto University, Kyoto, Japan

Yota Otachi ✉
Nagoya University, Nagoya, Japan

Saket Saurabh ✉
The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

For a connected graph $G = (V, E)$ and $s, t \in V$, a non-separating s-t path is a path P between s and t such that the set of vertices of P does not separate G, that is, $G - V(P)$ is connected. An s-t path P is non-disconnecting if $G - E(P)$ is connected. The problems of finding shortest non-separating and non-disconnecting paths are both known to be NP-hard. In this paper, we consider the problems from the viewpoint of parameterized complexity. We show that the problem of finding a non-separating s-t path of length at most k is W[1]-hard parameterized by k, while the non-disconnecting counterpart is fixed-parameter tractable (FPT) parameterized by k. We also consider the shortest non-separating path problem on several classes of graphs and show that this problem is NP-hard even on bipartite graphs, split graphs, and planar graphs. As for positive results, the shortest non-separating path problem is FPT parameterized by k on planar graphs and on unit disk graphs (where no s, t is given). Further, we give a polynomial-time algorithm on chordal graphs if k is the distance of the shortest path between s and t.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Non-separating path, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.6

Funding *Yasuaki Kobayashi*: JSPS KAKENHI Grant Numbers JP20H05793, JP20K19742, JP21H03499
Yota Otachi: JSPS KAKENHI Grant Numbers JP18H04091, JP18K11168, JP18K11169, JP20H05793, JP21K11752.

1 Introduction

Lovász's path removal conjecture states the following claim: There is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $f(k)$-connected graph G and every pair of vertices u and v, G has a path P between u and v such that $G - V(P)$ is k-connected. This claim still remains open, while

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 6; pp. 6:1–6:15
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

some special cases have been resolved [4, 15, 17, 23]. Tutte [23] proved that $f(1) = 3$, that is, every triconnected graph satisfies that for every pair of vertices, there is a path between them whose removal results a connected graph. Kawarabayashi et al. [15] proved a weaker version of this conjecture: There is a function $f\colon \mathbb{N} \to \mathbb{N}$ such that for every $f(k)$-connected graph G and every pair of vertices u and v, G has an induced path P between u and v such that $G - E(P)$ is k-connected.

As a practical application, let us consider a network represented by an undirected graph G, and we would like to build a private channel between a specific pair of nodes s and t. For some security reasons, the path used in this channel should be dedicated to the pair s and t, and hence all other connections do not use all nodes and/or edges on this path while keeping their connections. In graph-theoretic terms, the vertices (resp. edges) of the path between s and t does not form a separator (resp. a cut) of G. Tutte's result [23] indicates that such a path always exists in triconnected graphs, but may not exist in biconnected graphs. In addition to this connectivity constraint, the path between s and t is preferred to be short due to the cost of building a private channel. Motivated by such a natural application, the following two problems are studied.

▶ **Definition 1.** *Given a connected graph G, $s, t \in V(G)$, and an integer k,* Shortest Non-Separating Path *asks whether there is a path P between s and t in G such that the length of P is at most k and $G - V(P)$ is connected. When s and t are not part of the input, and we want to find a path P of length k, such that $G - V(P)$ is connected, then we call the problem* Terminal Independent Shortest Non-Separating Path (TI-Shortest Non-Separating Path)*.*

▶ **Definition 2.** *Given a connected graph G, $s, t \in V(G)$, and an integer k, Shortest Non-Disconnecting Path asks whether there is a path P between s and t in G such that the length of P is at most k and $G - E(P)$ is connected.*

Given, the Shortest Non-Separating Path problem, a natural question arises about the complexity of the problem, when in the problem we replace the demand of P being a path with P being a connected set. Given a connected graph G, and an integer k, Smallest Non-Separating Set asks whether there is a vertex subset X of size k in G such that $G[X]$ is connected and $G - X$ is connected. Similarly, an edge counterpart can be defined as Smallest Non-Disconnecting Set. An edge set X is said to be connected if the graph $G' = (V(X), X)$ is connected. We say that a path P is *non-separating* (in G) if $G - V(P)$ is connected and is *non-disconnecting* (in G) if $G - E(P)$ is connected. Similarly, we define the notion of *non-separating* set and *non-disconnecting* set.

1.1 Our Results and Methods

We investigate the parameterized complexity of above problems and obtain following results.

1. Shortest Non-Separating Path and Smallest Non-Separating Set are W[1]-hard. These are obtained by parameterized reductions from Multicolored Clique and Clique, respectively.
2. Shortest Non-Disconnecting Path and Smallest Non-Disconnecting Set are FPT parameterized by k. These algorithms are based on matroid based tools used in parameterized complexity [11]. In particular, given a graph G, there is a well-known matroid, defined by ground set being $E(G)$ and the family of independent sets being a subsets of Y of $E(G)$, such that $G - Y$ is connected. These are called cographic matroid.

A crucial observation for the FPT algorithms for Shortest Non-Disconnecting Path and Smallest Non-Disconnecting Set is that the set of edges in a non-disconnecting path or non-disconnecting set can be seen as an independent set of a cographic matroid. By applying the representative family of matroids [11], we show that Shortest Non-Disconnecting Path and Smallest Non-Disconnecting Set can be solved in $2^{\omega k}|V|^{O(1)}$ time, where ω is the exponent of the matrix multiplication. We also show that Shortest Non-Disconnecting Path is OR-compositional, that is, there is no polynomial kernel unless coNP $\subseteq$ NP/poly.

3. To cope with the intractability of Shortest Non-Separating Path, we consider the problem on planar graphs and unit disk graphs and show that Shortest Non-Separating Path is FPT parameterized by k on planar graphs and TI-Shortest Non-Separating Path is FPT parameterized by k on unit disk graphs. The result on planar graphs can be generalized to wider classes of graphs which have the *diameter-treewidth property* [9], which are precisely apex-minor-free graphs (includes, planar and graphs of bounded genus). For, Smallest Non-Separating Set we show that it does not have polynomial kernel even on planar graphs. We also consider Shortest Non-Separating Path on several classes of graphs. We can observe that the complexity of Shortest Non-Separating Path is closely related to that of Hamiltonian Cycle (or Hamiltonian Path with specified end vertices). This observation readily proves the NP-completeness of Shortest Non-Separating Path on bipartite graphs, split graphs, and planar graphs. For chordal graphs, we devise a polynomial-time algorithm for Shortest Non-Separating Path for the case where k is the shortest path distance between s and t.

Proofs of results for Shortest Non-Separating Path and Smallest Non-Separating Set are similar and the proofs of results for Shortest Non-Disconnecting Path and Smallest Non-Disconnecting Set are similar, in this version of the paper *we only focus on* Shortest Non-Separating Path and Shortest Non-Disconnecting Path.

Related work. The shortest path problem in graphs is one of the most fundamental combinatorial optimization problems, which is studied under various settings. Indeed, our problems Shortest Non-Separating Path and Shortest Non-Disconnecting Path can be seen as variants of this problem. From the computational complexity viewpoint, Shortest Non-Separating Path is known to be NP-hard and its optimization version cannot be approximated with factor $|V|^{1-\varepsilon}$ in polynomial time for $\varepsilon > 0$ unless P = NP [24]. Shortest Non-Disconnecting Path is shown to be NP-hard on general graphs and polynomial-time solvable on chordal graphs [19].

2 Preliminaries

We use standard terminologies and known results in matroid theory and parameterized complexity theory, which are briefly discussed in this section. See [6, 21] for details.

Graphs. Let G be a graph. The vertex set and edge set of G are denoted by $V(G)$ and $E(G)$, respectively. For $v \in V(G)$, the open neighborhood of v in G is denoted by $N_G(v)$ (i.e., $N_G(v) = \{u \in V(G) : \{u,v\} \in E(G)\}$) and the closed neighborhood of v in G is denoted by $N_G[v]$ (i.e., $N_G[v] = N_G(v) \cup \{v\}$). We extend this notation to sets: $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] = N_G(X) \cup X$ for $X \subseteq V(G)$. For $u, v \in V(G)$, we denote by $\mathrm{dist}_G(u,v)$ the length of a shortest path between u and v in G, where the length of a path is defined as the number of edges in it. We may omit the subscript of G from these notations when no confusion arises. For $X \subseteq V(G)$, we write $G[X]$ to denote the subgraph of G induced by X. For notational convenience, we may use $G - X$ instead of $G[V(G) \setminus X]$. For $F \subseteq E$, we also use $G - F$ to represent the subgraph of G consisting all vertices of G and all edges in $E \setminus F$. For vertices u and v, a path between u and v is called a *u-v path.* A vertex is called a *pendant* if its degree is exactly 1.

Matroids and representative sets. Let E be a finite set. If $\mathcal{I} \subseteq 2^E$ satisfies the following axioms, the pair $\mathcal{M} = (E, \mathcal{I})$ is called a *matroid*: (1) $\emptyset \in \mathcal{I}$; (2) $Y \in \mathcal{I}$ implies $X \in \mathcal{I}$ for $X \subseteq Y$; and (3) for $X, Y \in \mathcal{I}$ with $|X| < |Y|$, there is $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{I}$. Each set in $\mathcal{I}$ is called an *independent set* of $\mathcal{M}$. From the third axiom of matroids, it is easy to observe that every (inclusion-wise) maximal independent set of $\mathcal{M}$ have the same cardinality. The *rank* of $\mathcal{M}$ is the maximum cardinality of an independent set of $\mathcal{M}$. A matroid $\mathcal{M} = (E, \mathcal{I})$ of rank n is *linear* (or *representable*) over a field $\mathbb{F}$ if there is a matrix $M \in \mathbb{F}^{n \times |E|}$ whose columns are indexed by E such that $X \in \mathcal{I}$ if and only if the set of columns indexed by X is linearly independent in M.

Let $G = (V, E)$ be a graph. A *cographic matroid* of G is a matroid $\mathcal{M}(G) = (E, \mathcal{I})$ such that $F \subseteq E$ is an independent set of $\mathcal{M}(G)$ if and only if $G - F$ is connected. Every cographic matroid is linear and its representation can be computed in polynomial time [21]. Our algorithmic result for Shortest Non-Disconnected Path is based on *representative families* due to [11].

▶ **Definition 3.** *Let $\mathcal{M} = (E, \mathcal{I})$ be a matroid and let $\mathcal{F} \subseteq \mathcal{I}$ be a family of independent sets of $\mathcal{M}$. For an integer $q \geq 0$, we say that $\widehat{\mathcal{F}} \subseteq \mathcal{F}$ is* q-representative for $\mathcal{F}$ *if the following condition holds: For every $Y \subseteq E$ of size at most q, if there is $X \in \mathcal{F}$ with $X \cap Y = \emptyset$ such that $X \cup Y \in \mathcal{I}$, then there is $\widehat{X} \in \widehat{\mathcal{F}}$ with $\widehat{X} \cap Y = \emptyset$ such that $\widehat{X} \cup Y \in \mathcal{I}$.*

▶ **Theorem 4** ([11, 18]). *Given a linear matroid $\mathcal{M} = (E, \mathcal{I})$ of rank n that is represented as a matrix $M \in \mathbb{F}^{n \times |E|}$ for some field $\mathbb{F}$, a family $\mathcal{F} \subseteq \mathcal{I}$ of independent sets of size p, and an integer q with $p + q \leq n$, there is a deterministic algorithm computing a q-representative family $\widehat{\mathcal{F}} \subseteq \mathcal{F}$ of size $np\binom{p+q}{p}$ with*

$$O\left(|\mathcal{F}| \cdot \left(\binom{p+q}{p} p^3 n^2 + \binom{p+q}{q}^{\omega - 1} \cdot (pn)^{\omega - 1}\right)\right) + (n + |E|)^{O(1)}$$

field operations, where $\omega < 2.373$ is the exponent of the matrix multiplication.

Parameterized complexity. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. A *kernelization* for L is a polynomial-time algorithm that given an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, computes an "equivalent" instance $(I', k') \in \Sigma^* \times \mathbb{N}$ such that (1) $(I, k) \in L$ if and only if $(I', k') \in L$ and (2) $|I'| + k' \leq g(k)$ for some computable function g. We call (I', k') a *kernel.* If the function g is a polynomial, then the kernelization algorithm is called a *polynomial kernelization* and its output (I', k') is called a *polynomial kernel.* An *OR-composition* is an algorithm that given p instances $(I_1, k), \ldots (I_p, k) \in \Sigma^* \times \mathbb{N}$ of L, computes an instance $(I', k') \in \Sigma^* \times \mathbb{N}$ in time $(\sum_{1 \leq i \leq p} |I_i| + k)^{O(1)}$ such that (1) $(I', k') \in L$ if and only if $(I_i, k) \in L$ for some $1 \leq i \leq p$ and (2) $k' = k^{O(1)}$. For a parameterized problem L, its *unparameterized problem* is a language $L' = \{x\#1^k : (x, k) \in L\}$, where $\# \notin \Sigma$ is a blank symbol and $1 \in \Sigma$ is an arbitrary symbol.

▶ **Theorem 5** ([3]). *If a parameterized problem L admits an OR-composition and its unparameterized version is NP-complete, then L does not have a polynomial kernelization unless* $\mathrm{coNP} \subseteq \mathrm{NP/poly}$.

3 Shortest Non-Separating Path

We discuss our complexity and algorithmic results for Shortest Non-Separating Path.

3.1 Hardness on graph classes

We obverse that, in most cases, Shortest Non-Separating Path is NP-hard on classes of graphs for which Hamiltonian Path (with distinguished end vertices) is NP-hard. Let $G = (V, E)$ be a graph and $s, t \in V$ be distinct vertices of G. We add a pendant vertex p adjacent to s and denote the resulting graph by G'. Then, we have the following observation.

▶ **Observation 6.** *For every non-separating path P between s and t in G', $V(G) \setminus V(P) = \{p\}$.*

Suppose that for a class $\mathcal{C}$ of graphs,

- the problem of deciding whether given graph $G \in \mathcal{C}$ has a Hamiltonian path between specified vertices s and t in G is NP-hard and
- $G \in \mathcal{C}$ implies $G' \in \mathcal{C}$.

By Observation 6, G' has a non-separating s-t path if and only if G has a Hamiltonian path between s and t. This implies that the problem of finding a non-separating path between specified vertices is NP-hard on class $\mathcal{C}$.

▶ **Theorem 7.** *The problem of deciding if an input graph has a non-separating s-t path is NP-complete even on planar graphs, bipartite graphs, and split graphs.*

The classes of planar graphs and bipartite graphs are closed under the operation of adding a pendant. Recall that a graph G is a *split graph* if the vertex set $V(G)$ can be partitioned into a clique C and an independent set I. Thus, for the class of split graphs, we need the assumption that the pendant added is adjacent to a vertex in C.

As the problem trivially belongs to NP, it suffices to show that Hamiltonian Path (with distinguished end vertices) is NP-hard on these classes of graphs[1]. For split graphs, it is known that Hamiltonian Path is NP-hard even if the distinguished end vertices are contained in the clique C [20]. Let G be a graph and let $v \in V(G)$. We add a vertex v' that is adjacent to every vertex in $N_G(v)$, that is, v and v' are (false) twins. The resulting graph is denoted by G'. It is easy to verify that G has a Hamiltonian cycle if and only if G' has a Hamiltonian path between v and v'. The class of bipartite graphs is closed under this operation, that is, G' is bipartite. For planar graphs, G' may not be planar in general. However, Hamiltonian Cycle is NP-complete even if the input graph is planar and has a vertex of degree 2 [14]. We apply the above operation to this degree-2 vertex, and the resulting graph G' is still planar. As the problem of finding a Hamiltonian cycle is NP-hard even on bipartite graphs [20] and planar graphs [14], Theorem 7 follows.

[1] These results for bipartite graphs and planar graphs seem to be folklore but we were not able to find particular references.

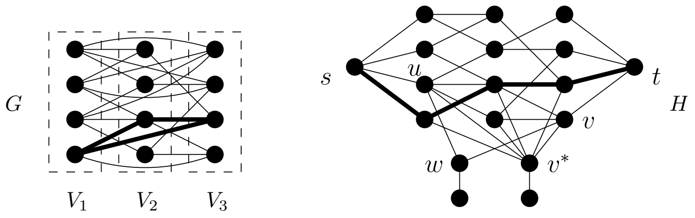


Figure 1 The left figure depicts an instance G of Multicolored Clique and the right figure depicts the graph H constructed from G. Some vertices and edges in H are not drawn in this figure for visibility. The edges of a clique C and the corresponding non-separating s-t path P are drawn as thick lines.

3.2 W[1]-hardness

Next, we show that Shortest Non-Separating Path is W[1]-hard parameterized by k. The proof is done by giving a reduction from Multicolored Clique, which is known to be W[1]-complete [10]. In Multicolored Clique, we are given a graph G with a partition $\{V_1, V_2, \dots, V_k\}$ of $V(G)$ and asked to determine whether G has a clique C such that $|V_i \cap C| = 1$ for each $1 \leq i \leq k$.

From an instance $(G, \{V_1, \dots, V_k\})$ of Multicolored Clique, we construct an instance of Shortest Non-Separating Path as follows. Without loss of generality, we assume that G contains more than k vertices. We add two vertices s and t, make s adjacent to all $v \in V_1$ and make t adjacent to all $v \in V_k$. For any pair of $u \in V_i$ and $v \in V_j$ with $|i - j| \geq 2$, we do the following. If $\{u, v\} \in E$, then we remove it. Otherwise, we add a path $P_{u,v}$ of length 2 and a pendant vertex that is adjacent to the internal vertex w of $P_{u,v}$. Finally, we add a vertex v^*, add an edge between v^* and each original vertex $v \in V(G)$, and add a pendant vertex p adjacent to v^*. The constructed graph is denoted by H. See Figure 1 for an illustration of the graph H.

▶ **Lemma 8.** *There is a clique C in G such that $|C \cap V_i| = 1$ for all $1 \leq i \leq k$ if and only if there is a non-separating s-t path of length at most $k + 1$ in H.*

Proof. Suppose first that G has a clique C with $C \cap V_i = \{v_i\}$ for all $1 \leq i \leq k$. Then, $P = \langle s, v_1, v_2, \dots, v_k, t\rangle$ is an s-t path of length $k + 1$ in H. To see the connectivity of $H - V(P)$, it suffices to show that every vertex is reachable from v^* in $H - V(P)$. By the construction of H, each vertex in $V(G) \setminus V(P)$ is adjacent to v^* in $H - V(P)$. Each vertex z in $V(H) \setminus (V(G) \cup \{v^*, p\})$ is either the internal vertex w of $P_{u,v}$ for some $u, v \in V(G)$ or the pendant vertex adjacent to w. In both cases, at least one of u and v is not contained in P as $V(P) \setminus \{s, t\}$ is a clique in G, implying that z is reachable to v^*.

Conversely, suppose that H has a non-separating s-t path P of length at most $k + 1$ in H. By the assumption that G has more than k vertices, there is a vertex of G that does not belong to P. Observe that P does not contain any internal vertex w of some $P_{u,v}$ as otherwise the pendant vertex adjacent to w becomes an isolated vertex by deleting $V(P)$, which implies $H - V(P)$ has at least two connected components. Similarly, P does not contain v^*. These facts imply that the internal vertices of P belong to $V(G)$, and we have $|V(P) \cap V_i| = 1$ for all $1 \leq i \leq k$. Let $C = V(P) \setminus \{s, t\}$. We claim that C is a clique in G. Suppose otherwise. There is a pair of vertices $u, v \in C$ that are not adjacent in G. This implies that H contains the path $P_{u,v}$. However, as P contains both u and v, the internal vertex of $P_{u,v}$ together with its pendant vertex forms a component in $H - V(P)$, yielding a contradiction that P is a non-separating path in H. ◀

Thus, we have the following theorem.

▶ **Theorem 9.** *Shortest Non-Separating Path is W[1]-hard parameterized by k.*

3.3 Graphs with the diameter-treewidth property

By Theorem 9, Shortest Non-Separating Path is unlikely to be fixed-parameter tractable on general graphs. To overcome this intractability, we focus on sparse graph classes. We first note that the algorithmic meta-theorems for FO Model Checking [12, 13] do not seem to be applicable to Shortest Non-Separating Path as we need to care about the connectivity of graphs. However, the property that vertex set X forms a non-separating s-t path can be expressed as:

$$\mathtt{conn}(V \setminus X) \wedge \mathtt{hampath}(X, s, t),$$

where $\mathtt{conn}(Y)$ and $\mathtt{hampath}(Y, s, t)$ are formulas in MSO_2 such that $\mathtt{conn}(Y)$ (resp. $\mathtt{hampath}(Y, s, t)$) is true if and only of the subgraph induced by Y is connected and (resp. the subgraph induced by Y has a Hamiltonian path between s and t). We omit the details of these formulas, which can be found in [6] for example[2]. By Courcelle's theorem [5] and its extension due to Arnborg et al. [1], we can compute a shortest non-separating s-t path in $O(f(\mathrm{tw}(G))n)$ time, where n is the number of vertices and $\mathrm{tw}(G)$ is the treewidth[3] of G. As there is an $O(\mathrm{tw}(G)^{\mathrm{tw}(G)^3}n)$-time algorithm for computing the treewidth of an input graph G [2], we have the following theorem.

▶ **Theorem 10.** *Shortest Non-Separating Path is fixed-parameter tractable parameterized by the treewidth of input graphs.*

A class $\mathcal{C}$ of graphs is *minor-closed* if every minor of a graph $G \in \mathcal{C}$ also belongs to $\mathcal{C}$. We say that $\mathcal{C}$ has the *diameter-treewidth property* if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $G \in \mathcal{C}$, the treewidth of G is upper bounded by $f(\mathrm{diam}(G))$, where $\mathrm{diam}(G)$ is the diameter of G. It is well known that every planar graph G has treewidth at most $3 \cdot \mathrm{diam}(G) + 1$ [22][4], which implies that the class of planar graphs has the diameter-treewidth property. This can be generalized to more wider classes of graphs. A graph is called an *apex graph* if it has a vertex such that removing it makes the graph planar.

▶ **Theorem 11** ([7, 9]). *Let $\mathcal{C}$ be a minor-closed class of graphs. Then, $\mathcal{C}$ has the diameter-treewidth property if and only if it excludes some apex graph.*

For $C \subseteq V(G)$ that induces a connected subgraph $G[C]$, we denote by G_C the graph obtained from G by contracting $G[C]$ into a single vertex v_C and making v_C adjacent to all the vertices in $N(C)$. Since $G[C]$ is connected, vertex v_C is well-defined.

▶ **Lemma 12.** *Let $C \subseteq V(G)$ be a vertex subset such that $G[C]$ is connected. Let P be an s-t path in G with $V(P) \cap C = \emptyset$. Then, P is non-separating in G if and only if it is non-separating in G_C.*

[2] In [6], they give an MSO_2 sentence $\mathtt{hamiltonicity}$ expressing the property of having a Hamiltonian cycle, which can be easily transformed into a formula expressing $\mathtt{hampath}(X, s, t)$.

[3] We do not give the definition of treewidth and (the optimization version of) Courcelle's theorem. We refer to [6] for details.

[4] More precisely, the treewidth of a planar graph is upper bounded by $3r + 1$, where r is the radius of the graph.

Proof. Suppose first that P is non-separating in G. Let $u, v \in V(G) \setminus V(P)$ be arbitrary. As P is non-separating, there is a u-v path P' in $G - V(P)$. Let u' be the vertex of G_C such that $u' = u$ if $u \notin C$ and $u' = v_C$ if $u \in C$. Let v' be the vertex defined analogously. We show that there is a u'-v' path in $G_C - V(P)$ as well. If P' does not contain any vertex in C, then it is also a u'-v' path in G_C, and hence we are done. Suppose otherwise. Let x and y be the vertices in $V(P') \cap C$ that are closest to u and v, respectively. Note that x and y can be the end vertices of P', that is, C may contain u and v. Let $P_{u,x}$ and (resp. $P_{y,v}$) be the subpath of P' between u and x (resp. y and v). Then, the sequence of vertices obtained by concatenating $P_{u,x}$ after $P_{y,v} - \{y\}$ and replacing exactly one occurrence of a vertex in C with v_C forms a path between u' and v'. Since we choose u, v arbitrarily, there is a path between any pair of vertices in $G_C - V(P)$ as well. Hence, P is non-separating in G_C.

Conversely, suppose that P is non-separating in G_C. For $u, v \in V(G_C) \setminus V(P)$, there is a path P' in $G_C - V(P)$. Suppose that neither $u = v_C$ nor $v = v_C$. Then, we can construct a u-v path in $G - V(P)$ as follows. If $v_C \notin V(P')$, P' is also a path in $G - V(P)$ and hence we are done. Otherwise, $v_C \in V(P')$. Let P_u and P_v be the subpaths in $P' - \{v_C\}$ containing u and v, respectively. From P_u and P_v, we have a u-v path in G by connecting them with an arbitrary path in $G[C]$ between the end vertices other than u and v. Note that such a bridging path in $G[C]$ always exists since $G[C]$ is connected. Moreover, as $V(P') \cap C = \emptyset$ and $V(P) \cap C = \emptyset$, this is also a u-v path in $G - V(P)$. Suppose otherwise that either $u = v_C$ or $v = v_C$, say $u = v_C$. In this case, we can construct a path between every vertex w in C and v by concatenating P' and an arbitrary path in $G[C]$ between w and the end vertex of the subpath $P' - \{v_C\}$ other than v. Therefore, P is non-separating in G. ◀

Now, we are ready to state the main result of this subsection.

▶ **Theorem 13.** *Let $\mathcal{C}$ be a graph class excluding a fixed apex graph H as a minor. Then, Shortest Non-Separating Path on $\mathcal{C}$ is fixed-parameter tractable parameterized by k.*

Proof. Let $G \in \mathcal{C}$. We first compute $B = \{v \in V(G) : \mathrm{dist}(s, v) \le k\}$. This can be done in linear time. If $t \notin B$, then the instance (G, s, t, k) is trivially infeasible. Suppose otherwise that $t \in B$. Let C be a component in $G - B$. By definition, every non-separating s-t path P of length at most k does not contain any vertex of C. Let G' be the graph obtained from G by contracting all edges in $E(G - B)$. For each component C in $G - B$, we denote by v_C the vertex of G' corresponding to C (i.e., v_C is the vertex obtained by contracting all edges in C). Then, we have $\mathrm{diam}(G') \le 2k + 2$ as $\mathrm{diam}(G[B]) \le k$ and every vertex in $V(G') \setminus B$ is adjacent to a vertex in B. By Lemma 12, G has a non-separating s-t path of length at most k if and only if so does G'. Since $\mathcal{C}$ is minor-closed, we have $G' \in \mathcal{C}$ and hence the treewidth of G' is upper bounded by $f(2k + 2)$ for some function f. By Theorem 10, we can check whether G' has a non-separating s-t path of length at most k in $O(g(k)|V(G')|)$ time. ◀

Theorem 10 does not give precise dependence on $\mathrm{tw}(G)$ in the running time of the algorithm. In fact, given a tree decomposition of G of width $\mathrm{tw}(G)$ we can design an algorithm for Shortest Non-Separating Path running in time $2^{O(\mathrm{tw}(G))}n^{O(1)}$, using matroid based tools [11]. Further, there exists a factor-2 approximation for $\mathrm{tw}(G)$ running in time $2^{O(\mathrm{tw}(G))}n$ [16]. Combined with this we get the following result.

▶ **Theorem 14.** *There exists an algorithm for Shortest Non-Separating Path running in time $2^{O(\mathrm{tw}(G))}n^{O(1)}$.*

The proof of this result is based on the standard dynamic programming over graphs of bounded treewidth together with representative sets and will appear in the final version of the paper. Applying the result of Theorem 14 in Theorem 13, we get the following results.

▶ **Theorem 15.** *Let $\mathcal{C}$ be a graph class excluding a fixed apex graph H as a minor. Then,* Shortest Non-Separating Path *on $\mathcal{C}$ admits an algorithm with running time $2^{O(k)}n^{O(1)}$.*

3.4 TI-Shortest Non-Separating Path on Unit Disk Graphs

Given n unit disks in the plane, a unit disk graph G consists of n vertices, corresponding to each disk, and there is an edge between two vertices if and only if the corresponding unit disks intersect. For our problem, we are given a unit disk graph $G = (V, E)$ and its representation $(\mathcal{D}, C)$. Let $V = \{v_1, v_2, \cdots, v_n\}$ and for each v_i, D_i be the corresponding disk centered at c_i. Here $\mathcal{D} = \{D_i| \ 1 \le i \le n\}$ and $C = \{c_i| \ 1 \le i \le n\}$.

▶ **Reduction Rule 1.** *If G has more than two connected components then return* NO.

▶ **Lemma 16.** *Reduction Rule 1 is safe and can be implemented in polynomial time.*

Let us assume that the graph has exactly two connected components. Observe that the given instance is a YES instance if and only if at least one of the two components contains exactly k vertices. Otherwise, the instance is a NO instance. The number of components in the given graph and the number of vertices in each component can be checked in polynomial time. Thus now onward we assume that G is connected. Let us consider a $(\frac{1}{2} \times \frac{1}{2})$ square grid on the plane. Let V_S be the set of centers of the disks that are contained inside a grid cell S; more formally, $V_S = \{v_i| \ c_i \in S\}$. We also define $N(V', S)$ as the set of vertices in the cell S that are neighbors of vertices in V'. For any cell S in the grid, next, we prove that if S contains at least $k + 24$ centers then the given instance is a YES instance.

▶ **Reduction Rule 2.** *If there exists a cell S with at least $k + 24$ centers, return YES.*

▶ **Lemma 17.** *Reduction Rule 2 is safe and can be implemented in polynomial time.*

From now onwards we assume that each cell has at most $k + 23$ centers.

▶ **Theorem 18.** TI-Shortest Non-Separating Path *can be solved in time $2^{O(k \log k)}n^{O(1)}$ on unit disk graphs.*

Proof. We guess the first vertex of the path, say v_i. Let Y be the set of vertices in the circle drawn centering c_i with radius k. Observe that the vertices of the solution path P must be a subset of Y. There are at most $\mathcal{O}(k^2)$ cells inside the circle drawn centering c_i with radius k. By Reduction Rule 2, no cell contains more than $k + 23$ vertices. Thus Y contains at most $O(k^3)$ vertices. Now all we need to do is to guess a subset X of size k as potential vertices of the path P and test that indeed it forms a path and $G - X$ is connected. All this can be done in $\binom{O(k^3)}{k} \times k! \times n^{O(1)} = 2^{O(k \log k)}n^{O(1)}$ time, concluding the proof. ◀

3.5 Chordal graphs with $k = \text{dist}(s, t)$

In Section 3.1, we have seen that Shortest Non-Separating Path is NP-complete even on split graphs (and thus on more general chordal graphs as well). To overcome this intractability, we restrict ourselves to finding a non-separating s-t path of length $\text{dist}(s, t)$ on chordal graphs.

A graph G is *chordal* if it has no cycles of length at least 4 as an induced subgraph. In the following, we fix a connected chordal graph G.

▶ **Lemma 19.** *Let $S \subseteq V(G)$ be a vertex set such that $G[S]$ is connected. For $u, v \in S$, every induced u-v path P in G satisfies that $V(P) \subseteq N[S]$.*

For $u, v \in V(G)$, a set of vertices $S \subseteq V(G) \setminus \{u, v\}$ is called a *u-v separator* of G if there is no u-v path in $G - S$. An inclusion-wise minimal u-v separator of G is called a *minimal u-v separator*. A *minimal separator* of G is a minimal u-v separator for some $u, v \in G$. Dirac's well-know characterization [8] of chordal graphs states that a graph is chordal if and only if every minimal separator induces a clique.

▶ **Lemma 20.** *Let $s, t \in V(G)$ be such that $\{s, t\} \notin E(G)$. If $v \in V(G) \setminus \{s, t\}$ is an internal vertex of a shortest s-t path P, then $N[v] \setminus \{s, t\}$ is an s-t separator of G.*

Proof. Let $d = \mathrm{dist}(s, t)$. For $0 \leq i \leq d$, let

$$D_i = \{v \in V(G) : \mathrm{dist}(s, v) = i \wedge \mathrm{dist}(v, t) = d - i\}$$

and $V(P) \cap D_i = \{u_i\}$. Observe that each D_i is a clique: if $i \in \{0, d\}$, then it is a singleton; otherwise, it is a minimal s-t separator of the chordal graph $G[\bigcup_{0 \leq j \leq d} D_j]$, meaning that D_i is a clique. From this observation, we have $D_i \subseteq N[u_i] \setminus \{s, t\}$ for $0 < i < d$. Let j ($0 < j < d$) be the index such that $v = u_j$.

Suppose to the contrary that there is an induced s-t path Q such that $V(Q) \cap (N[u_j] \setminus \{s, t\}) = \emptyset$. By Lemma 19, $V(Q) \subseteq N[V(P)] = \bigcup_{0 \leq i \leq d} N[u_i]$ holds. Since Q starts in $N[u_0]$ and ends in $N[u_d]$, there are indices i and k with $0 \leq i < j < k \leq d$ such that Q consecutively visits a vertex $v_i \in N[u_i]$ and then a vertex $v_k \in N[u_k]$ in this order. Since $\mathrm{dist}(u_i, u_k) = k - i \geq 2$ and $\{v_i, v_k\} \in E$, at least one of $v_i \neq u_i$ and $v_k \neq u_k$ holds. By symmetry, we assume that $v_i \neq u_i$.

If $v_k = u_k$, then $v_i \in N(u_i) \cap N(u_k)$. In this case, we have $i = j - 1$ and $k = j + 1$ since otherwise P admits a shortcut using the subpath $\langle u_i, v_i, u_k \rangle$. This implies that $\mathrm{dist}(s, v_i) \leq \mathrm{dist}(s, u_i) + 1 = i + 1 = j$ and $\mathrm{dist}(v_i, t) \leq 1 + \mathrm{dist}(v_k, t) = 1 + \mathrm{dist}(u_k, t) = 1 + (d - k) = d - j$. Since $\mathrm{dist}(s, v_i) + \mathrm{dist}(v_i, t) \geq d$, we have $\mathrm{dist}(s, v_i) = j$ and $\mathrm{dist}(v_i, t) = d - j$. This implies that $v_i \in D_j \subseteq N[u_j] \setminus \{s, t\}$, a contradiction.

Next we consider the case $v_k \neq u_k$. Recall that we also have $v_i \neq u_i$ as an assumption. In this case, we have $k - i \leq 3$ as $\langle u_i, v_i, v_k, u_k \rangle$ is not a shortcut for P. Assume first that $k - i = 3$. By symmetry, we may assume that $i = j - 1$ and $k = j + 2$. Since $\mathrm{dist}(s, v_i) \leq \mathrm{dist}(s, u_i) + 1 = j$ and $\mathrm{dist}(v_i, t) \leq 2 + \mathrm{dist}(u_k, t) \leq 2 + (d - k) = d - j$, we have $v_i \in D_j \subseteq N[u_j] \setminus \{s, t\}$, a contradiction. Next assume that $k - i = 2$. That is, $i = j - 1$ and $k = j + 1$. Since $v_i, v_k \notin N[u_j] \setminus \{s, t\}$ and P is shortest, the vertices v_i, u_i, u_j, u_k, v_k are distinct and form a cycle of length 5. Observe that $v_i \notin \{s, t\}$ since otherwise $\langle v_i = s, v_k, u_k \rangle$ or $\langle u_i, v_i = t \rangle$ is a shortcut. Similarly, $v_k \notin \{s, t\}$. Hence, $v_i, v_k \notin N[u_j]$. Therefore, the possible chords for the cycle $\langle v_i, u_i, u_j, u_k, v_k \rangle$ are $\{u_i, v_k\}$ and $\{u_k, v_i\}$. In any combination of them, the graph has an induced cycle of length at least 4. ◀

Let d and D_i be defined as in the proof of Lemma 20, and let $D = \bigcup_{0 \leq i \leq d} D_i$. Recall that each D_i is a clique. Observe that if $|D_i| = 1$ for all $0 \leq i \leq d$, then G contains a unique shortest s-t path, and thus the problem is trivial. Otherwise, we define ℓ to be the minimum index such that $|D_\ell| > 1$ and r to be the maximum index such that $|D_r| > 1$. Since $|D_0| = |D_d| = 1$, we have $0 < \ell \leq r < d$.

Our algorithm works as follows.

1. If G contains a unique shortest s-t path P, then test if P is non-separating.
2. Otherwise, find a shortest s-t path P satisfying the following conditions.
   a. $V(P)$ does not contain a minimal a-b separator for all $a \in D_\ell$ and $b \in V \setminus D$.
   b. $V(P)$ does not contain a minimal a-b separator for all $a \in D_\ell$ and $b \in D_r$.

▶ **Lemma 21.** *The algorithm is correct.*

▶ **Lemma 22.** *The algorithm has a polynomial-time implementation.*

We do not optimize the running time of the above algorithm, and a straightforward implementation runs in time $O(n^2m)$, where $n = |V(G)|$ and $m = |E(G)|$, which might be improved with some data structure.

▶ **Theorem 23.** *There is a polynomial-time algorithm for* Shortest Non-Separating Path *on chordal graphs, given that k is equal to the shortest path distance between s and t.*

4 Shortest Non-Disconnecting Path

The goal of this section is to establish the fixed-parameter tractability and a conditional lower bound on polynomial kernelizations for Shortest Non-Disconnecting Path.

4.1 Fixed-parameter tractability

▶ **Theorem 24.** *Shortest Non-Disconnecting Path can be solved in time $2^{\omega k} n^{O(1)}$, where ω is the matrix multiplication exponent and n is the number of vertices of G.*

To prove this theorem, we give a dynamic programming algorithm with the aid of representative families of cographic matroids. Let (G, s, t, k) be an instance of Shortest Non-Disconnecting Path. For $0 \le i \le k$ and $v \in V(G)$, we define $\mathsf{dp}(i, v)$ as the family of all sets of edges F satisfying the following two conditions: (1) F is the set of edges of an s-v path of length i and (2) $G - F$ is connected. An edge set F is *legitimate* if F forms a path and $G - F$ is connected. For a family of edge sets $\mathcal{F}$ and an edge e, we define $\mathcal{F} \bowtie e := \{F \cup \{e\} : F \in \mathcal{F}\}$ and $\mathsf{leg}(\mathcal{F})$ as the subfamily of $\mathcal{F}$ consisting of all legitimate $F \in \mathcal{F}$. The following simple recurrence correctly computes $\mathsf{dp}(i, v)$.

$$\mathsf{dp}(i,v) = \begin{cases} \{\emptyset\} & i = 0 \text{ and } s = v & (3) \\ \emptyset & i = 0 \text{ and } s \neq v & (4) \\ \mathsf{leg}\left(\bigcup_{u \in N(v)} (\mathsf{dp}(i-1,u) \bowtie \{u,v\})\right) & i > 0\ . & (5) \end{cases}$$

A straightforward induction proves that $\mathsf{dp}(i, t) \neq \emptyset$ if and only if G has a non-disconnecting s-t path of length exactly i and hence it suffices to check whether $\mathsf{dp}(i, t) \neq \emptyset$ for $0 \le i \le k$. However, the running time to evaluate this recurrence is $n^{O(k)}$. To reduce the running time of this algorithm, we apply Theorem 4 to each $\mathsf{dp}(i, v)$. Now, instead of (5), we define

$$\mathsf{dp}(i,v) = \mathsf{rep}_{k-i}\left(\mathsf{leg}\left(\bigcup_{u \in N(v)} (\mathsf{dp}(i-1,u) \bowtie \{u,v\})\right)\right), \qquad (3')$$

where $\mathsf{rep}_{k-i}(\mathcal{F})$ is a $(k - i)$-representative family of $\mathcal{F}$ for the cographic matroid $\mathcal{M} = (E(G), \mathcal{I})$ defined on G. In the following, we abuse the notation of dp to denote the families of legitimate sets that are computed by the recurrence composed of (3), (4), and (3').

▶ **Lemma 25.** *The recurrence composed of (3), (4), and (3') is correct, that is, G has a non-disconnecting s-t path of length at most k if and only if $\bigcup_{0 \le i \le k} \mathsf{dp}(i, t) \neq \emptyset$.*

Proof. It suffices to show that $\mathsf{dp}(k', t) \neq \emptyset$ if G has a non-disconnecting s-t path P of length $k' \leq k$. Let $P = (v_0 = s, v_1, \dots, v_{k'} = t)$ be a non-disconnecting path in G. We assume that G has no non-disconnecting s-t path of length strictly smaller than k'. For $0 \leq i \leq k'$, we let $P_i = (v_i, v_{i+1}, \dots, v_{k'})$. In the following, we prove, by induction on i, a slightly stronger claim that there is a legitimate set $F \in \mathsf{dp}(i, v_i)$ such that $F \cup E(P_i)$ forms a non-disconnecting s-t path in G for all $0 \leq i \leq k'$. As $\mathsf{dp}(0, s) = \{\emptyset\}$ and $P_0 = P$ itself is a non-disconnecting path, we are done for $i = 0$. Suppose that $i > 0$. By the induction hypothesis, there is a legitimate $F \in \mathsf{dp}(i-1, v_{i-1})$ such that $F \cup E(P_{i-1})$ forms a non-disconnecting s-t path in G. Note that $F \cap E(P_{i-1}) = \emptyset$ as otherwise G has a non-disconnecting s-t path of length smaller than k'. Let $\mathcal{F} = \mathsf{leg}(\bigcup_{u \in N(v_i)}(\mathsf{dp}(i-1) \bowtie \{u, v_i\}))$. Since $F \cup E(P_{i-1})$ is legitimate, $F \cup \{\{v_{i-1}, v_i\}\}$ is also legitimate, implying that $\mathcal{F}$ is nonempty. Let $\widehat{\mathcal{F}} = \mathsf{rep}_{k-i}(\mathcal{F})$ be $(k-i)$-representative for $\mathcal{F}$, $X = F \cup \{\{v_{i-1}, v_i\}\}$, and let $Y = E(P_i)$. As $|Y| \leq k - i$, $X \cap Y = \emptyset$, and $X \cup Y \in \mathcal{I}$, $\widehat{\mathcal{F}}$ contains an edge set $\widehat{X}$ with $\widehat{X} \cap Y = \emptyset$ and $\widehat{X} \cup Y \in \mathcal{I}$, implying that there is $\widehat{X} \in \mathsf{dp}(i, v_i)$ such that $\widehat{X} \cup E(P_i)$ forms a non-disconnecting s-t path in G. ◀

▶ **Lemma 26.** *The recurrence can be evaluated in time $2^{\omega k} n^{O(1)} \subset 5.18^k n^{O(1)}$, where $\omega < 2.373$ is the exponent of the matrix multiplication.*

Proof. By Theorem 4, $\mathsf{dp}(i, v)$ contains at most $2^k kn$ sets for $0 \leq i \leq k$ and $v \in V(G)$ and can be computed in time $2^{\omega k} n^{O(1)}$. ◀

Thus, Theorem 24 follows.

4.2 Kernel lower bound

It is well known that a parameterized problem is fixed-parameter tractable if and only if it admits a kernel (see [6], for example). By Theorem 24, Shortest Non-Disconnecting Path admits a kernel. A next natural step next is to explore the existence of polynomial kernel for Shortest Non-Disconnecting Path. However, the following theorem conditionally rules out the possibility of polynomial kernelization. We first show the following lemma.

▶ **Lemma 27.** *Let H be a connected graph. Suppose that H has a cut vertex v. Let C be a component in $H - \{v\}$ and let $F \subseteq E(H[C \cup \{v\}])$. Then, $H - F$ is connected if and only if $H[C \cup \{v\}] - F$ is connected.*

Proof. If $H - F$ is connected, then all the vertices in $C \cup \{v\}$ are reachable from v in $H - F$ without passing through any vertex in $V(H) \setminus (\{C\} \cup \{v\})$. Thus, such vertices are reachable from v in $H[C \cup \{v\}] - F$. Conversely, suppose $H[C \cup \{v\}] - F$ is connected. Then, every vertex in C is reachable from v in $H - F$. Moreover, as F does not contain any edge outside $H[C \cup \{v\}]$, every other vertex is reachable from v in $H - F$ as well. ◀

▶ **Theorem 28.** *Unless* coNP $\subseteq$ NP/poly*, Shortest Non-Disconnecting Path does not admit a polynomial kernelization (with respect to parameter k).*

Proof. We give an OR-composition for Shortest Non-Disconnecting Path. Let $(G_1, s_1, t_1, k), \dots, (G_p, s_p, t_p, k)$ be p instances of Shortest Non-Disconnecting Path.

We assume that for $1 \leq i \leq p$, t_i is not a cut vertex in G_i. To justify this assumption, suppose that t_i is a cut vertex in G_i. Let C be the component in $G_i - \{t_i\}$ that contains s_i. By Lemma 27, for any s_i-t_i path, it is non-disconnecting in G_i if and only if so is in $G_i[C \cup \{t_i\}]$. Thus, by replacing G_i with $G_i[C]$, we can assume that t_i is not a cut vertex in G_i.

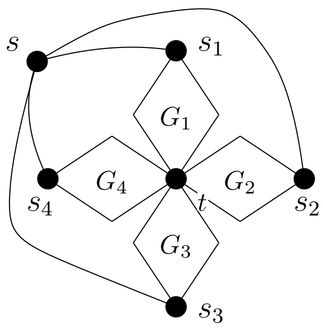


Figure 2 An illustration of the graph G obtained from $q = 4$ instances.

From the disjoint union of $G_1, \dots, G_p$, we construct a single instance (G, s, t, k') as follows. We first add a vertex s and an edge between s and s_i for each $1 \leq i \leq p$. Then, we identify all t_i's into a single vertex t. See Figure 2 for an illustration.

In the following, we may not distinguish t from t_i. Now, we claim that $(G, s, t, k+1)$ is a yes-instance if and only if (G_i, s_i, t_i, k) is a yes-instance for some i.

Consider an arbitrary s-t path in G. Observe that all edges in the path except for the one incident to s are contained in a single subgraph G_i for some $1 \leq i \leq p$ as $\{s, t\}$ separates $V(G_i) \setminus \{t_i\}$ from $V(G_j) \setminus \{t_j\}$ for $j \neq i$. Moreover, the path P forms $P = (s, s_i, v_1, \dots, v_q, t)$, meaning that the subpath $P' = (s_1, v_1, \dots, v_q, t_i)$ is an s_i-t_i path in G_i. This conversion is reversible: for any s_i-t_i path P' in G_i, the path obtained from P' by attaching s adjacent to s_i is an s-t path in G. Thus, it suffices to show that for $F \subseteq E(G_i)$, $F \cup \{\{s, s_i\}\}$ is a cut of G if and only if F is a cut of G_i. Since t is a cut vertex in $G - \{\{s, s_i\}\}$, by Lemma 27, the claim holds. ◀

We obtain the following result for the Smallest Non-Separating Set problem.

▶ **Theorem 29.** *Unless* coNP $\subseteq$ NP/poly, Smallest Non-Separating Set *does not admit a polynomial kernelization (with respect to parameter k) on planar graphs.*

Proof. We give an OR-composition for Smallest Non-Separating Set. Given t planar graphs $G_1, G_2, \cdots, G_t$ where each G_i contains n_i many vertices. We construct a new planar graph G' as follows. We create $k+2$ many copies for each planar graph G_i. For each planar graph G and a vertex $v \in V(G)$ there exists a planar embedding with v on the outer-face. Fix an arbitrary set of $k+2$ *distinct* vertices in G_i, say $v^1, v^2, \cdots, v^{k+2}$. Then, we obtain embeddings for $G_i^1, G_i^2, \cdots, G_i^{k+2}$ such that $v_i^1, v_i^2, \cdots, v_i^{k+2}$ are on the outer-face, respectively. Now add an edge between v_i^ℓ and $v_i^{\ell+1}$ for all $1 \leq \ell \leq k+1$. Also, add an edge between v_i^{k+2} and v_{i+1}^1 for all $1 \leq i \leq t-1$. That is these planar graphs are chained into a path. It is easy to see that the resulting instance G' is planar.

▷ Claim 30. G' is a YES instance if and only if at least one of the G_i is a YES instance.

Proof. We prove the forward direction first. Assume that in G', there exists a connected set X of size k in G' such that $G' - X$ is connected. Notice, deleting any vertex v_i^j from G_i^j where $j \in [k+1]$ makes the graph G' disconnected. Thus X can not contain any vertex v_i^j from G_i^j where $j \in [k+1]$. Hence, we can assume that $X \cap \{v_i^j\} = \emptyset$ in G_j for $1 \leq j \leq k+2$, i.e X is contained completely inside any one of G_i^j without containing any vertex at the outer face that shares edges with other copies of G_i. Therefore, X is of size k inside G_i^j and $G_i^j - X$ is connected. Hence, G_i is a YES instance.

Now, we prove the reverse direction. Assume that there is a graph, G_i in which there exists a connected set X of size k such that $G_i - X$ is connected. Without loss of generality, assume $X = \{v_1, v_2, \cdots, v_k\}$. Since G_i contains X, all the copies of G_i in G' also contain X. Now, we show that there exists a copy of G_i, say G_i^p in G', that does not contain any vertex of X on the outer face which shares edges with other copies of G_i. As $|X| = k$, observe that there can be at most k many different copies of G_i in G' which has a vertex of X on the outer faces which shares edges with the other copies. Hence, by pigeon hole principle, there exists at least a copy of G_i in G' which has no vertex on the outer face that shares edges with other copies. Thus, deleting X from that copy will not disconnect any G_i^a or G_j^b in the graph G'. Therefore, G' is also a YES instance. ◁

This concludes the proof. ◀

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

2 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.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

4 Guantao Chen, Ronald J. Gould, and Xingxing Yu. Graph connectivity after path removal. *Comb.*, 23(2):185–203, 2003. doi:10.1007/s003-0018-z.

5 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

7 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Diameter and treewidth in minor-closed graph families, revisited. *Algorithmica*, 40(3):211–215, 2004. doi:10.1007/s00453-004-1106-1.

8 G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961.

9 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/s004530010020.

10 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.

11 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.

12 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *AMS-ASL Joint Special Session*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2009.

13 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

14 Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11(4):676–686, 1982. doi:10.1137/0211056.

15 Ken-ichi Kawarabayashi, Orlando Lee, Bruce A. Reed, and Paul Wollan. A weaker version of lovász' path removal conjecture. *J. Comb. Theory, Ser. B*, 98(5):972–979, 2008. doi:10.1016/j.jctb.2007.11.003.

16 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.

17 Matthias Kriesell. Induced paths in 5-connected graphs. *J. Graph Theory*, 36(1):52–58, 2001. doi:10.1002/1097-0118(200101)36:1<52::AID-JGT5>3.0.CO;2-N.

18 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018.

19 Xiao Mao. Shortest non-separating st-path on chordal graphs. *CoRR*, abs/2101.03519, 2021. arXiv:2101.03519.

20 Haiko Müller. Hamiltonian circuits in chordal bipartite graphs. *Discret. Math.*, 156(1-3):291–298, 1996.

21 James G. Oxley. *Matroid Theory (Oxford Graduate Texts in Mathematics)*. Oxford University Press, Inc., USA, 2006.

22 Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

23 William T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, s3-13(1):743–767, 1963. doi:10.1112/plms/s3-13.1.743.

24 Bang Ye Wu and Hung-Chou Chen. The approximability of the minimum border problem. In *The 26th Workshop on Combinatorial Mathematics and Computation Theory*, 2009.

Comonadic semantics for hybrid logic

Samson Abramsky
University College London, London, UK

Dan Marsden
University of Oxford, Oxford, UK

Abstract

Hybrid logic is a widely-studied extension of basic modal logic, which corresponds to the bounded fragment of first-order logic. We study it from two novel perspectives: (1) We apply the recently introduced paradigm of comonadic semantics, which provides a new set of tools drawing on ideas from categorical semantics which can be applied to finite model theory, descriptive complexity and combinatorics. (2) We give a novel semantic characterization of hybrid logic in terms of *invariance under disjoint extensions*, a minimal form of locality. A notable feature of this result is that we give a uniform proof, valid for both the finite and infinite cases.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases comonads, model comparison games, semantic characterizations, hybrid logic, bounded fragment

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.7

1 Introduction

Hybrid logic (see e.g. [8, 7]) has been widely studied as an expressive extension of basic modal logic. It is semantically natural, e.g. in the analysis of temporal reasoning [10], and since it allows an internalisation of relational semantics, it has a very well-behaved proof theory [11], without needing to resort to explicit labelling of proofs or tableaux. The corresponding fragment of first-order logic under modal translation is the *bounded fragment*, in which quantification is relativized to atomic formulas from the relational vocabulary. This fragment is important in set theory [19], and has been studied in general proof- and model-theoretic terms in [16, 15].

In the present paper we study hybrid logic with inverse modalities, which we shall refer to as hybrid temporal logic, from two novel perspectives:

- Firstly, we apply the recently introduced paradigm of *comonadic semantics* [1, 5], which gives a uniform description of a wide range of logic fragments indexed by resource parameters. These fragments play a key role in finite model theory and descriptive complexity. Examples include the Ehrenfeucht-Fraïssé comonads $\mathbb{E}_k$, which capture the quantifier-rank fragments; the pebbling comonads $\mathbb{P}_k$, which capture the finite variable fragments; and the modal comonads $\mathbb{M}_k$, which capture the modal fragments of bounded modal depth. In each case, the comonads induce a number of resource-indexed equivalences on structures, which can be shown to capture the equivalences induced by the corresponding logic fragments. Moreover, the coalgebras for these comonads can be shown to characterise important combinatorial invariants of structures. For example, in the case of $\mathbb{P}_k$, the corresponding invariant is *tree-width* [5, 1].

  The common structure exhibited by this wide range of examples has been axiomatised in a very general setting in terms of arboreal categories [4]. This provides a new set of tools drawing on ideas from categorical semantics which can be applied to finite model theory, descriptive complexity and combinatorics. Early examples of the use of these ideas can be found in [25, 13, 2, 14], and further results are emerging rapidly, see e.g. [22, 12].

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 7; pp. 7:1–7:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the present paper, we extend the program of comonadic semantics to hybrid logic. The comonad which captures hybrid logic is a natural restriction of a pointed version of the Ehrenfeucht-Fraïssé comonad previously introduced in [5]. This comonadic analysis nicely reveals, in a clear and conceptual way, the way in which hybrid logic sits between basic modal logic and first-order logic. We characterise the coalgebras for this comonad as tree covers of a relational structure with additional locality constraints. This enables a uniform treatment of logical equivalences, bisimulation games, and combinatorial parameters, within the axiomatic framework recently given in [4].

- Secondly, we give a novel semantic characterization of the version of hybrid logic we study, in terms of *invariance under disjoint extensions* (and various equivalent formulations). This is a minimal form of locality relative to a given base-point, and shows that hybrid logic is the maximal fragment of first-order logic retaining a local character. A notable feature of this result is that we give a uniform proof, valid for both the finite and infinite cases. In particular, we make no use of the compactness theorem, and instead use game-theoretic constructions, specifically a result we call the Workspace Lemma.

  Apart from the interest of the results pertaining to hybrid logic in themselves, we see our work here as fitting into and refining a larger picture, of an emerging landscape in which the tractability of various logic fragments is mirrored in the structural properties of the corresponding comonads. In particular, hybrid logic is undecidable, but still retains a local character. A salient property which the modal and guarded fragments have, and hybrid logic lacks, is the bisimilar companion property. This property plays a key role in the uniform proofs of the van Benthem-Rosen Theorem for these fragments [24, 3]. We mitigate the failure of this property for hybrid logic by the use of game-theoretic arguments. All of this will be explained in detail in Section 6.

After some preliminaries in Section 2, we shall introduce the hybrid comonad in Section 3, and study the coalgebras for this comonad in Section 4. In Section 5, we characterize the equivalence on structures induced by hybrid logic in terms of *spans of open pathwise embeddings* for this comonad, following the pattern established in [5] and axiomatised in [4]. Then we develop the results on the semantic characterisation of hybrid logic in Section 6.

2 Preliminaries

We shall need a few notions on posets. Given $x, y \in P$ for a poset $(P, \leq)$, we write $x \uparrow y$ if x and y are comparable in the order, i.e. $x \leq y$ or $y \leq x$. We will use finite sequences extensively; these are partially ordered by prefix, with notation $s \sqsubseteq t$ indicating that list s is a prefix of list t.

A relational vocabulary σ is a set of relation symbols R, each with a specified positive integer arity. A σ-structure $\mathfrak{A}$ is given by a set A, the universe of the structure, and for each R in σ with arity n, a relation $R^{\mathfrak{A}} \subseteq A^n$. A homomorphism $h : \mathfrak{A} \to \mathfrak{B}$ is a function $h : A \to B$ such that, for each relation symbol R of arity n in σ, for all $a_1, \ldots, a_n$ in A: $R^{\mathfrak{A}}(a_1, \ldots, a_n) \Rightarrow R^{\mathfrak{B}}(h(a_1), \ldots, h(a_n))$. We write $\mathsf{Struct}(\sigma)$ for the category of σ-structures and homomorphisms.

Since evaluation in modal logics is relative to a given world, we shall also use the pointed category $\mathsf{Struct}_\star(\sigma)$. Objects are pairs $(\mathfrak{A}, a)$, where $\mathfrak{A}$ is a σ-structure, and $a \in A$. Morphisms $h : (\mathfrak{A}, a) \to (\mathfrak{B}, b)$ are homomorphisms $h : \mathfrak{A} \to \mathfrak{B}$ such that $h(a) = b$.

A *modal vocabulary* has only relation symbols of arity ≤ 2: a set of unary predicate symbols P, which will correspond to modal propositional atoms; and a binary relation symbol E, which we think of as a *transition relation* (more traditionally referred to as an *accessibility relation*).

2.1 Hybrid Temporal Logic

The main system we shall study is Hybrid Temporal Logic (HTL). HTL formulas are built from propositional atoms p and *world variables* x, with the following syntax:

$$\varphi ::= p \mid x \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \Box\varphi \mid \Diamond\varphi \mid \Box^{-}\varphi \mid \Diamond^{-}\varphi \mid {\downarrow}x.\,\varphi \mid @_x\varphi.$$

We use a redundant syntax to make it more convenient to discuss fragments. The new features compared with basic modal logic, augmented with backwards modalities as is standard in temporal logic, are the world variables, which can be bound with $\downarrow$, and used to force evaluation at a given world with @. Hybrid formulae are graded by their *hybrid modal depth.* This is the usual notion of modal depth, with the adjustment that sub-formulae of the form $\Diamond x$ or $\Diamond^{-} x$, for some world variable x, are deemed to have zero depth.

The semantics of hybrid temporal logic is given by translation into first-order logic with equality over a unary modal vocabulary, with a unary predicate P for each proposition atom p, and a single transition relation E. World variables are treated as ordinary first-order variables. The translation is parameterised on a variable, corresponding to the world at which the formula is to be evaluated. We write $\psi[x/y]$ for the result of substituting x for the free occurrences of y in ψ.

$$\begin{aligned}
\mathsf{ST}_x(p) &= P(x) \\
\mathsf{ST}_x(x') &= x = x' \\
\mathsf{ST}_x(\neg\varphi) &= \neg\,\mathsf{ST}_x(\varphi) \\
\mathsf{ST}_x(\varphi \wedge \varphi') &= \mathsf{ST}_x(\varphi) \wedge \mathsf{ST}_x(\varphi') \\
\mathsf{ST}_x(\varphi \vee \varphi') &= \mathsf{ST}_x(\varphi) \vee \mathsf{ST}_x(\varphi')
\end{aligned}
\qquad
\begin{aligned}
\mathsf{ST}_x(\Box\varphi) &= \forall y.[E(x,y) \to \mathsf{ST}_y(\varphi)] \\
\mathsf{ST}_x(\Diamond\varphi) &= \exists y.[E(x,y) \wedge \mathsf{ST}_y(\varphi)] \\
\mathsf{ST}_x(\Box^{-}\varphi) &= \forall y.[E(y,x) \to \mathsf{ST}_y(\varphi)] \\
\mathsf{ST}_x(\Diamond^{-}\varphi) &= \exists y.[E(y,x) \wedge \mathsf{ST}_y(\varphi)] \\
\mathsf{ST}_x({\downarrow}x'.\varphi) &= \mathsf{ST}_x(\varphi)[x/x'] \\
\mathsf{ST}_x(@_{x'}\varphi) &= \mathsf{ST}_x(\varphi)[x'/x]
\end{aligned}$$

The obvious stipulations about renaming bound variables to avoid variable capture apply.

The target of this translation is the *bounded fragment* of first-order logic with equality, with quantifiers restricted to those of the form $\exists y.[E(x,y) \wedge \varphi]$, $\forall y.[E(x,y) \to \varphi]$, $\exists y.[E(y,x) \wedge \varphi]$, $\forall y.[E(y,x) \to \varphi]$, with $x \neq y$. Hybrid temporal logic is in fact equiexpressive with this fragment [7].

Note that $\mathsf{ST}_x(\Diamond y)$ is logically equivalent to $E(x,y)$, and similarly $\mathsf{ST}_x(\Diamond^{-} y)$ is logically equivalent to $E(y,x)$. Thus these formulas test for the presence of a transition between worlds which have already been reached, justifying our assignment of modal depth 0.

3 The hybrid comonad

We shall now introduce the hybrid comonad on $\mathsf{Struct}_\star(\sigma)$ for modal vocabularies σ, motivating it as combining features of the Ehrenfeucht-Fraïssé and modal comonads from [5].

1. We recall firstly the Ehrenfeucht-Fraïssé comonad $\mathbb{E}_k$ on $\mathsf{Struct}(\sigma)$ for an arbitrary vocabulary σ. Given a structure $\mathfrak{A}$, the universe of $\mathbb{E}_k\mathfrak{A}$ is the set of non-empty sequences of elements of A of length $\leq k$. We think of these sequences as *plays* in the Ehrenfeucht-Fraïssé game on $\mathfrak{A}$. We define the map $\varepsilon_{\mathfrak{A}} : \mathbb{E}_k A \to A$ which sends a sequence to its last element, which we think of as the current move or *focus* of the play. For a relation R of arity n, we define $R^{\mathbb{E}_k\mathfrak{A}}(s_1, \ldots, s_n)$ to hold iff $s_i {\uparrow} s_j$ for all $1 \leq i, j \leq n$, and $R^{\mathfrak{A}}(\varepsilon_{\mathfrak{A}}(s_1), \ldots, \varepsilon_{\mathfrak{A}}(s_n))$. Explicitly, for unary predicates P, $P^{\mathbb{E}_k\mathfrak{A}}(s)$ iff $P^{\mathfrak{A}}(\varepsilon_{\mathfrak{A}}(s))$, and for a binary relation R, $R^{\mathbb{E}_k\mathfrak{A}}(s,t)$ iff $s{\uparrow}t$ and $R^{\mathfrak{A}}(\varepsilon_{\mathfrak{A}}(s), \varepsilon_{\mathfrak{A}}(t))$. Thus the relations hold *along* plays as one extends another, but not between *different* (i.e. incomparable) plays.

2. This construction lifts to the pointed category $\mathsf{Struct}_\star(\sigma)$. We define the universe of $\mathbb{E}_k(\mathfrak{A}, a)$ to comprise the non-empty sequences of length $\leq k+1$ which start with a. The distinguished element is $\langle a \rangle$. The relations are lifted in exactly the same way as previously.
3. The modal comonad $\mathbb{M}_k$ over a modal vocabulary with unary predicates P corresponding to propositional atoms, and a single transition relation E, restricts the sequences in $\mathbb{E}_k(\mathfrak{A}, a)$ to those of the form $\langle a_0, \ldots a_j \rangle$, $a_0 = a$, such that for all i with $0 \leq i < j$, $E^{\mathfrak{A}}(a_i, a_{i+1})$. Thus we can only extend a sequence with an element which the previous element "sees". Moreover, the transition relation E is lifted in a correspondingly local fashion, so that a sequence is only related to its immediate extensions: $E^{\mathbb{M}_k(\mathfrak{A},a)}(s,t)$ iff $t = s\langle a' \rangle$ and $E^{\mathfrak{A}}(\varepsilon_{\mathfrak{A}}(s), \varepsilon_{\mathfrak{A}}(t))$. This is the familiar *unravelling* construction for modal structures [9].
4. The hybrid comonad $\mathbb{H}_k$ is again defined on the pointed category $\mathsf{Struct}_\star(\sigma)$. $\mathbb{H}_k(\mathfrak{A}, a)$ has as universe the subset of $\mathbb{E}_k(A, a)$ of those sequences $\langle a_0, a_1, \ldots, a_l \rangle$ such that $a_0 = a$, and for all j with $0 < j \leq l$, for some i, $0 \leq i < j$, $E^{\mathfrak{A}}(a_i, a_j)$ or $E^{\mathfrak{A}}(a_j, a_i)$. Thus we relax the locality condition of $\mathbb{M}_k$ to the condition that a sequence can only be extended with an element if it is related to *some* element which has been played previously. The σ-relations on $\mathbb{H}_k(\mathfrak{A}, a)$ are defined exactly as for $\mathbb{E}_k(\mathfrak{A}, a)$, and the distinguished element is $\langle a \rangle$, so $\mathbb{H}_k(\mathfrak{A}, a)$ is the induced substructure of $\mathbb{E}_k(\mathfrak{A}, a)$ given by this restriction of the universe. In this sense, $\mathbb{H}_k$ is closer to $\mathbb{E}_k$ than to $\mathbb{M}_k$.

To complete the specification of $\mathbb{H}_k$, we define the *coKleisli extension*: given a morphism $h : \mathbb{H}_k(\mathfrak{A}, a) \to (\mathfrak{B}, b)$, we define $h^* : \mathbb{H}_k(\mathfrak{A}, a) \to \mathbb{H}_k(\mathfrak{B}, b)$ by

$$h^*(\langle a, a_1, \ldots, a_i \rangle) = \langle h(\langle a \rangle), h(\langle a, a_1 \rangle), \ldots, h(\langle a, a_1, \ldots, a_i \rangle) \rangle.$$

We can verify that for each structure $\mathfrak{A}$, $\varepsilon_{\mathfrak{A}} : \mathbb{H}_k \mathfrak{A} \to \mathfrak{A}$ is a morphism; that for each morphism $h : \mathbb{H}_k(\mathfrak{A}, a) \to (\mathfrak{B}, b)$, $h^* : \mathbb{H}_k(\mathfrak{A}, a) \to \mathbb{H}_k(\mathfrak{B}, b)$ is a morphism; and that the following equations are satisfied, for all morphisms $h : \mathbb{H}_k(\mathfrak{A}, a) \to (\mathfrak{B}, b)$, $g : \mathbb{H}_k(\mathfrak{B}, b) \to (\mathfrak{C}, c)$:

$$\varepsilon_{\mathfrak{A}} \circ h^* = h, \qquad \varepsilon_{\mathfrak{A}}^* = \mathsf{id}_{\mathbb{H}_k \mathfrak{A}}, \qquad (g \circ h^*)^* = g^* \circ h^*,$$

This establishes the following result.

▶ **Proposition 1.** *The triple* $(\mathbb{H}_k, \varepsilon, (\cdot)^*)$ *is a comonad in Kleisli form [21].*

It is then standard [21] that $\mathbb{H}_k$ extends to a functor by $\mathbb{H}_k f = (f \circ \epsilon)^*$; that ε is a natural transformation; and that if we define the comultiplication $\delta : \mathbb{H}_k \Rightarrow \mathbb{H}_k^2$ by $\delta_{\mathfrak{A}} = \mathsf{id}^*_{\mathbb{H}_k \mathfrak{A}}$, then $(\mathbb{H}_k, \varepsilon, \delta)$ is a comonad.

3.1 *I*-morphisms and equality

Like the Ehrenfeucht-Fraïssé comonad $\mathbb{E}_k$, and unlike the modal comonad $\mathbb{M}_k$, equality is important for $\mathbb{H}_k$, as we might expect from its appearance in the translation of hybrid temporal logic into first-order logic. We shall follow the procedure introduced in [5, Section 4] to ensure that equality is properly handled in $\mathbb{E}_k$.

The issue is that elements of A may be repeated in the plays in $\mathbb{H}_k(\mathfrak{A}, a)$. In particular, this happens when there are cycles in the graph $(A, E^{\mathfrak{A}})$ which are reachable from a. We wish to view coKleisli morphisms $f : \mathbb{H}_k(\mathfrak{A}, a) \to (\mathfrak{B}, b)$ as winning strategies for Duplicator in the one-sided (or existential) Spoiler-Duplicator game from $(\mathfrak{A}, a)$ to $(\mathfrak{B}, b)$, in which Spoiler plays in $\mathfrak{A}$ and Duplicator in $\mathfrak{B}$ [18]. In order to fulfil the partial homomorphism winning condition, f must map repeated occurrences of an element $a' \in A$ in a play s in $\mathbb{H}_k(\mathfrak{A}, a)$ to the same element of B. The same issue will recur when we deal with back-and-forth games in section 5. We seek a systematic means of enforcing this requirement.

Given a relational vocabulary σ, we produce a new one $\sigma^+ = \sigma \cup \{I\}$, where I is a binary relation symbol not in σ. If we interpret $I^{(\mathfrak{A},a)}$ and $I^{(\mathfrak{B},b)}$ as the identity relations on A and B, then, following the general prescription for relation lifting in $\mathbb{E}_k(\mathfrak{A}, a)$, and hence also in $\mathbb{H}_k(\mathfrak{A}, a)$ as an induced substructure of $\mathbb{E}_k(\mathfrak{A}, a)$, we have $I^{(\mathbb{H}_k\mathfrak{A},a)}(s,t)$ iff $s{\uparrow}t$ and $\varepsilon_{\mathfrak{A}}(s) = \varepsilon_{\mathfrak{A}}(t)$. Thus a σ-morphism $f : (\mathbb{H}_k\mathfrak{A}, a) \to (\mathfrak{B}, b)$ satisfies the required condition iff it is a σ^+-morphism.

As it stands, this is an ad hoc condition: it relies on a special interpretation of the I-relation. We want our objects to live in $\mathsf{Struct}_\star(\sigma)$, but our morphisms to live in $\mathsf{Struct}_\star(\sigma^+)$. To accomplish this, we use a simple special case of the notion of *relative comonad* [6]. We can take advantage of the fact that $\mathbb{E}_k$, and hence $\mathbb{H}_k$ as a sub-comonad of $\mathbb{E}_k$, is defined uniformly in the vocabulary. Given a vocabulary σ, there is a full and faithful embedding $J : \mathsf{Struct}_\star(\sigma) \to \mathsf{Struct}_\star(\sigma^+)$ such that $I^{J(\mathfrak{A},a)}$ is the identity on A. Moreover, we have a comonad $\mathbb{E}_k^I$, which is the $\mathbb{E}_k$ construction applied to $\mathsf{Struct}_\star(\sigma^+)$. Note that this treats I like any other binary relation in the vocabulary.

We correspondingly obtain $\mathbb{H}_k^I(\mathfrak{A}, a)$ as the substructure of $\mathbb{E}_k^I(\mathfrak{A}, a)$ induced by restricting the universe to that of $\mathbb{H}_k(\mathfrak{A}, a)$. It is important to note that only the transition relation E is used to restrict the universe.

We use this data to obtain the J-relative comonad $\mathbb{H}_k^+ = \mathbb{H}_k^I \circ J$ on $\mathsf{Struct}_\star(\sigma)$. The objects of the coKleisli category for this relative comonad are those of $\mathsf{Struct}_\star(\sigma)$. CoKleisli morphisms have the form $\mathbb{H}_k^I J(\mathfrak{A}, a) \to J(\mathfrak{B}, b)$. The counit and coextension are the restrictions of those for $\mathbb{H}_k^I$ to the image of J.

3.2 CoKleisli maps, existential games, and the existential positive fragment

The standard k-round existential Ehrenfeucht-Fraïssé game from $\mathfrak{A}$ to $\mathfrak{B}$ [18, 5] is defined as follows. In each round i, Spoiler moves by choosing an element a_i from A, and Duplicator responds by choosing an element b_i from B. The winning condition for Duplicator is that the correspondence $a_i \mapsto b_i$ is a partial homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$.

The k-round existential hybrid game from $(\mathfrak{A}, a)$ to $(\mathfrak{B}, b)$ is defined in exactly the same way, with two additional provisos:

- At round 0, Spoiler must play $a_0 = a$, and Duplicator must respond with $b_0 = b$.
- At round $j > 0$, Spoiler must play a move a_j such that, for some $i < j$, $E^{\mathfrak{A}}(a_i, a_j)$ or $E^{\mathfrak{A}}(a_j, a_i)$.

▶ **Proposition 2.** *There is a bijective correspondence between*

- *Winning strategies for Duplicator in the k-round existential hybrid game from $(\mathfrak{A}, a)$ to $(\mathfrak{B}, b)$*
- *CoKleisli morphisms $h : \mathbb{H}_k^+(\mathfrak{A}, a) \to J(\mathfrak{B}, b)$.*

The existential positive fragment $\mathsf{HTL}^\Diamond$ of hybrid temporal logic is defined by omitting negation and both $\Box$ and $\Box^-$ from the syntax for hybrid logic given in section 2.1. $\mathsf{HTL}_k^\Diamond$ is the fragment of $\mathsf{HTL}^\Diamond$ comprising formulas of hybrid modal depth $\leq k$.

This fragment induces a preorder on pointed structures. Define $(\mathfrak{A}, a) \Rrightarrow_k^{\mathsf{HTL}} (\mathfrak{B}, b)$ as:

$$\forall \varphi \in \mathsf{HTL}_k^\Diamond . [(\mathfrak{A}, a) \models \varphi \;\Rightarrow\; (\mathfrak{B}, b) \models \varphi].$$

Here by $(\mathfrak{A}, a) \models \varphi$ we mean $(\mathfrak{A}, a) \models \psi(x)$, where $\psi(x) = \mathsf{ST}_x(\varphi)$.

The following is a variation on standard results (see e.g. [9, 5]).

▶ **Proposition 3.** *There is a winning strategy for Duplicator in the k-round existential hybrid game from $(\mathfrak{A}, a)$ to $(\mathfrak{B}, b)$ iff $(\mathfrak{A}, a) \Rrightarrow_k^{\mathsf{HTL}} (\mathfrak{B}, b)$.*

We define another preorder on pointed structures: $(\mathfrak{A}, a) \rightarrow_k^{\mathbb{H}} (\mathfrak{B}, b)$ iff there is a coKleisli morphism $h : \mathbb{H}_k^+(\mathfrak{A}, a) \rightarrow J(\mathfrak{B}, b)$. The following is then an immediate consequence of Propositions 2 and 3.

▶ **Theorem 4.** *Let σ be a finite modal vocabulary. For all $(\mathfrak{A}, a)$, $(\mathfrak{B}, b)$ in $\mathsf{Struct}_\star(\sigma)$:*

$$(\mathfrak{A}, a) \Rrightarrow_k^{\mathsf{HTL}} (\mathfrak{B}, b) \iff (\mathfrak{A}, a) \rightarrow_k^{\mathbb{H}} (\mathfrak{B}, b).$$

4 Coalgebras

We now study coalgebras for the hybrid comonad. These will yield a natural combinatorial invariant associated with hybrid logic, and also provide a basis for the semantic characterization of the equivalence on structures induced by hybrid logic, which will be given in the following section.

A coalgebra for a comonad (G, ε, δ) is a morphism $\alpha : A \rightarrow GA$ such that $\varepsilon_A \circ \alpha = \mathsf{id}_A$ and $\delta_A \circ \alpha = G(\alpha) \circ \alpha$. Given G-coalgebras $\alpha : A \rightarrow GA$ and $\beta : B \rightarrow GB$, a coalgebra morphism from α to β is a morphism $h : A \rightarrow B$ such that $\beta \circ h = G(h) \circ \alpha$. This gives a category of coalgebras and coalgebra morphisms, denoted by $\mathbf{EM}(G)$, the *Eilenberg-Moore category* of G.

We will now analyze $\mathbf{EM}(\mathbb{H}_k)$, the category of coalgebras for the hybrid comonad on a unimodal vocabulary σ. This will lead to a natural combinatorial parameter associated with hybrid temporal logic, which is a refinement of *tree-depth* [23]. It will also provide a basis for a comonadic characterisation of bisimulation and the equivalence on structures induced by the full hybrid temporal logic, as we will see in the next section.

We will need a few more notions on posets. A chain in a poset $(P, \leq)$ is a subset $C \subseteq P$ such that, for all $x, y \in C$, $x \uparrow y$. A *forest* is a poset $(F, \leq)$ such that, for all $x \in F$, the set of predecessors $\downarrow(x) := \{y \in F \mid y \leq x\}$ is a finite chain. The height $\mathsf{ht}(F)$ of a forest F is $\sup_C |C|$, where C ranges over chains in F. Note that the height is either finite or ω. A *tree* is a forest with a least element (the root). We write the covering relation for a poset as $\prec$; thus $x \prec y$ iff $x \leq y$, $x \neq y$, and for all z, $x \leq z \leq y$ implies $z = x$ or $z = y$. Morphisms of trees are monotone maps preserving the root and the covering relation.

Given a σ-structure $\mathfrak{A}$, the *Gaifman graph* $\mathcal{G}(\mathfrak{A})$ is $(A, \frown)$, where $a \frown a'$ (a is adjacent to a') if they are distinct elements of A which both occur in a tuple of some relation $R^{\mathfrak{A}}$, R in σ.

A *tree cover* of a pointed σ-structure $(\mathfrak{A}, a)$ is a tree order $(A, \leq)$ on A with least element a, and such that if $a \frown a'$, then $a \uparrow a'$. Thus adjacent elements in the Gaifman graph must appear in the same branch of the tree. The tree cover is *generated* if for all $a' \in A$ with $a' \neq a$, for some $a'' \in A$, $a'' < a'$ and $a' \frown a''$. Tree covers of a pointed structure are neither unique, nor guaranteed to exist.

▶ **Theorem 5.** *For any pointed σ-structure $(\mathfrak{A}, a)$, and $k > 0$, there is a bijective correspondence between:*

- *$\mathbb{H}_k$-coalgebras $\alpha : (\mathfrak{A}, a) \rightarrow \mathbb{H}_k(\mathfrak{A}, a)$.*
- *Generated tree covers of $(\mathfrak{A}, a)$ of height $\leq k + 1$.*

We define the *generated tree depth* of $(\mathfrak{A}, a)$ to be the minimum height of any generated tree cover of $(\mathfrak{A}, a)$. This can be seen as a refinement of the standard notion of tree depth [23].

We define the *hybrid coalgebra number* of $(\mathfrak{A}, a)$ to be the least k such that there is an $\mathbb{H}_k$-coalgebra $\alpha : (\mathfrak{A}, a) \to \mathbb{H}_k(\mathfrak{A}, a)$. If there is no coalgebra for any k, the hybrid coalgebra number is ω. The following is an immediate consequence of Theorem 5.

▶ **Theorem 6.** *The generated tree depth of a structure $(\mathfrak{A}, a)$ coincides with its hybrid coalgebra number.*

We define a category $\mathbf{Tree}(\sigma)$ with objects $(\mathfrak{A}, a, \leq)$, where $(\mathfrak{A}, a)$ is a pointed σ-structure, and $\leq$ is a generated tree cover of $(\mathfrak{A}, a)$. Morphisms $h : (\mathfrak{A}, a, \leq) \to (\mathfrak{B}, b, \leq')$ are morphisms of pointed σ-structures which are also tree morphisms. That is, they preserve the covering relation $\prec$ in the tree order, and the root element. For each $k > 0$, there is a full subcategory $\mathbf{Tree}(\sigma)_k$ determined by those objects whose covers have height $\leq k$.

▶ **Theorem 7.** *For each $k > 0$, $\mathbf{Tree}(\sigma)_k$ is isomorphic to $\mathbf{EM}(\mathbb{H}_k)$.*

There is an evident forgetful functor $U_k : \mathbf{Tree}(\sigma)_k \to \mathsf{Struct}_\star(\sigma)$ which sends $(\mathfrak{A}, a, \leq)$ to $(\mathfrak{A}, a)$. The following is now an immediate consequence of Theorem 7.

▶ **Theorem 8.** *For each $k > 0$, U_k has a right adjoint $R_k : \mathsf{Struct}_\star(\sigma) \to \mathbf{Tree}(\sigma)_k$ given by $R_k(\mathfrak{A}, a) = (\mathbb{H}_k(\mathfrak{A}, a), \sqsubseteq)$. The comonad induced by this adjunction is $\mathbb{H}_k$. The adjunction is comonadic.*

5 Paths, open maps, and back-and-forth equivalence

The coalgebra category $\mathbf{EM}(\mathbb{H}_k)$ has a richer structure than $\mathsf{Struct}_\star(\sigma)$, articulated as $\mathbf{Tree}(\sigma)_k$ by Theorem 7. In fact, $\mathbf{Tree}(\sigma)_k$ is an *arboreal category* as defined in [4]. The axiomatic structure of an arboreal category allows us to define notions of bisimulation and games on this category, which can then be transferred to $\mathsf{Struct}_\star(\sigma)$ via the adjunction $U_k \dashv R_k$, following the general pattern laid out in [5]. This leads to a semantic characterisation of the equivalence on structures induced by hybrid logic.

To accommodate I-morphisms, as discussed in section 3.1, we work with the J-relative version of this adjunction, using $R_k^+ = R_k^I J$, where R_k^I is the instance of the adjunction for $\mathsf{Struct}_\star(\sigma^+)$.

5.1 Embeddings, paths and pathwise embeddings

A morphism e in $\mathbf{Tree}(\sigma)_k$ is an *embedding* if $U_k(e)$ is an embedding of relational structures. We write $e : T \rightarrowtail U$ to indicate that e is an embedding.

A *path* in $\mathbf{Tree}(\sigma)_k$ is an object P such that the associated tree cover is a finite linear order, so it comprises a single branch; moreover, I^P is the identity relation. We say that $e : P \rightarrowtail T$ is a *path embedding* if P is a path. A morphism $f : T \to U$ in $\mathbf{Tree}(\sigma)_k$ is a *pathwise embedding* if for any path embedding $e : P \rightarrowtail T$, $f \circ e$ is a path embedding.

5.2 Open maps

A morphism $f : T \to U$ in $\mathbf{Tree}(\sigma)_k$ is *open* if, whenever we have a commuting diagram such as 1, where P and Q are paths, there is an embedding $Q \rightarrowtail T$ such that 2 commutes.

$$\begin{tikzcd} P \arrow[r, tail] \arrow[d, tail] & Q \arrow[d, tail] \\ T \arrow[r, "f"'] & U \end{tikzcd} \qquad (1) \qquad\qquad \begin{tikzcd} P \arrow[r, tail] \arrow[d, tail] & Q \arrow[d, tail] \arrow[dl, tail] \\ T \arrow[r, "f"'] & U \end{tikzcd} \qquad (2)$$

This is often referred to as the *path-lifting property.* If we think of f as witnessing a simulation of T by U, path-lifting means that if we extend a given behaviour in U (expressed by extending the path P to Q), then we can find a matching behaviour in T to "cover" this extension. Thus it expresses an abstract form of the notion of "p-morphism" from modal logic [9], or of functional bisimulation.

5.3 Bisimulation

We can now define the *back-and-forth equivalence* $(\mathfrak{A}, a) \leftrightarrow_k^{\mathbb{H}} (\mathfrak{B}, b)$ between structures in $\mathsf{Struct}_\star(\sigma)$. This holds if there is a span of open pathwise embeddings in $\mathbf{Tree}(\sigma)_k$ $R_k^+(\mathfrak{A}, a) \leftarrow T \rightarrow R_k^+(\mathfrak{B}, b)$. Note that we are using the arboreal category $\mathbf{Tree}(\sigma)_k$ to define an equivalence on the "extensional category" $\mathsf{Struct}_\star(\sigma)$.

5.4 Games

We shall now define a *back-and-forth game* $\mathcal{G}_k((\mathfrak{A}, a), (\mathfrak{B}, b))$ played between $(\mathfrak{A}, a)$ and $(\mathfrak{B}, b)$, using the comonad $\mathbb{H}_k$. Positions of the game are pairs $(s, t) \in \mathbb{H}_k(\mathfrak{A}, a) \times \mathbb{H}_k(\mathfrak{B}, b)$. The initial position is $(\langle a \rangle, \langle b \rangle)$.

We define a relation $\mathsf{W}((\mathfrak{A}, a), (\mathfrak{B}, b))$ on positions as follows. A pair (s, t) is in $\mathsf{W}((\mathfrak{A}, a), (\mathfrak{B}, b))$ iff for some path P, path embeddings $e_1 : P \rightarrowtail \mathbb{H}_k(\mathfrak{A}, a)$ and $e_2 : P \rightarrowtail \mathbb{H}_k(\mathfrak{B}, b)$, and $p \in P$, $s = e_1(p)$ and $t = e_2(p)$. The intention is that $\mathsf{W}((\mathfrak{A}, a), (\mathfrak{B}, b))$ picks out the winning positions for Duplicator.

At the start of each round of the game, the position is specified by $(s, t) \in \mathbb{H}_k(\mathfrak{A}, a) \times \mathbb{H}_k(\mathfrak{B}, b)$. The round proceeds as follows. Either Spoiler chooses some $s' \succ s$, and Duplicator must respond with $t' \succ t$, resulting in a new position (s', t'); or Spoiler chooses some $t'' \succ t$ and Duplicator must respond with $s'' \succ s$, resulting in (s'', t''). Duplicator wins the round if they are able to respond, and the new position is in $\mathsf{W}((\mathfrak{A}, a), (\mathfrak{B}, b))$.

5.5 Results

▶ **Theorem 9.** *Given $(\mathfrak{A}, a)$, $(\mathfrak{B}, b)$ in $\mathsf{Struct}_\star(\sigma)$, then $(\mathfrak{A}, a) \leftrightarrow_k^{\mathbb{H}} (\mathfrak{B}, b)$ iff Duplicator has a winning strategy for $\mathcal{G}_k((\mathfrak{A}, a), (\mathfrak{B}, b))$.*

Proof. The proof is a minor variation of that for [5, Theorem 10.1], the corresponding result for $\mathbb{E}_k$. Alternatively, this is an instance of the very general [4, Theorem 6.9]. ◀

The standard k-round Ehrenfeucht-Fraïssé game between $\mathfrak{A}$ and $\mathfrak{B}$ [20] is defined as follows. In each round i, Spoiler moves by either

- choosing an $a_i \in A$, to which Duplicator responds by choosing a $b_i \in B$; or
- choosing a $b_i \in B$, to which Duplicator responds by choosing an $a_i \in A$.

The winning condition for Duplicator is that the correspondence $a_i \mapsto b_i$ is a partial isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$.

The k-round back-and-forth hybrid game between $(\mathfrak{A}, a)$ and $(\mathfrak{B}, b)$ is defined in exactly the same way, with two additional provisos:

- At round 0, Spoiler must either play $a_0 = a$, to which Duplicator must respond with $b_0 = b$; or $b_0 = b$, to which Duplicator must respond with $a_0 = a$
- At round $j > 0$, if Spoiler plays a move $a_j \in A$ then, for some $i < j$, $E^{\mathfrak{A}}(a_i, a_j)$ or $E^{\mathfrak{A}}(a_j, a_i)$; while if Spoiler plays a move $b_j \in B$ then, for some $i < j$, $E^{\mathfrak{B}}(b_i, b_j)$ or $E^{\mathfrak{B}}(b_j, b_i)$.

The partial isomorphism winning condition ensures that Duplicator is subject to the same constraints.

We write HTL_k for the set of hybrid formulas of modal depth k. We define an equivalence relation on pointed structures $(\mathfrak{A}, a) \equiv_k^{\mathsf{HTL}} (\mathfrak{B}, b)$ as $\forall \varphi \in \mathsf{HTL}_k. [(\mathfrak{A}, a) \models \varphi \iff (\mathfrak{B}, b) \models \varphi]$.

▶ **Theorem 10.** *Let σ be a finite unimodal vocabulary. For all $(\mathfrak{A}, a)$, $(\mathfrak{B}, b)$ in $\mathsf{Struct}_\star(\sigma)$, the following are equivalent:*

1. $(\mathfrak{A}, a) \leftrightarrow_k^{\mathbb{H}} (\mathfrak{B}, b)$.
2. *Duplicator has a winning strategy for the k-round back-and-forth hybrid game between $(\mathfrak{A}, a)$ and $(\mathfrak{B}, b)$.*
3. $(\mathfrak{A}, a) \equiv_k^{\mathsf{HTL}} (\mathfrak{B}, b)$.

6 Semantic characterization of hybrid temporal logic

We shall now prove a semantic characterisation of hybrid temporal logic in terms of invariance under disjoint extensions. A related result is already known [16, 7], however there are several novel features in our account:

- The previous results are for general (possibly infinite) structures, using tools from infinite model theory. We will give a uniform proof, which applies both to general structures, and to the finite case, which, as for the van Benthem-Rosen characterisation of basic modal logic in terms of bisimulation invariance [27, 26], is an independent result.
- Our proof follows similar lines to the uniform proof by Otto of the van Benthem-Rosen Theorem [24]. In particular, we use constructive arguments based on model comparison games, rather than model-theoretic constructions involving compactness. However, a key property used in his proof no longer holds for the hybrid fragment, so the argument has to take a different path.
- We also identify a key combinatorial lemma, implicit in [24], which we call the Workspace Lemma.
- One of the equivalent conditions in our characterization, invariance under disjoint extensions, appears to be new in this context. We can regard invariance under disjoint extensions as a minimal form of locality relative to a given basepoint. Thus this characterization shows that hybrid temporal logic defines the maximal fragment of first-order logic which retains a local character in this sense.

6.1 Comonadic aspects

Comonadic semantics have now been given for a number of important fragments of first-order logic: the quantifier rank fragments, the finite variable fragments, the modal fragment, and guarded fragments. In the landscape emerging from these constructions, some salient properties have come to the fore. These are properties which a comonad, arising from an *arboreal cover* in the sense of [4], may or may not have:

- The comonad may be *idempotent*, meaning that the comultiplication is a natural isomorphism. Idempotent comonads correspond to *coreflective subcategories*, which form the Eilenberg-Moore categories of these comonads. The modal comonads $\mathbb{M}_k$ are idempotent. The corresponding coreflective subcategories are of those modal structures which are tree-models to depth k [5].

- The comonad C may satisfy the following property: for each structure $\mathfrak{A}$, $C\mathfrak{A} \leftrightarrow^{C} \mathfrak{A}$, where $\leftrightarrow^{C}$ is the back-and-forth equivalence associated with C. We shall call this the *bisimilar companion* property. Note that an idempotent comonad, such as $\mathbb{M}_k$, will automatically have this property. The guarded comonads $\mathbb{G}_k$ from [2] are not idempotent, but have the bisimilar companion property, which is thus strictly weaker.
- Finally, the comonads $\mathbb{E}_k$ and $\mathbb{P}_k$ have neither of the above properties. Unlike the modal and guarded fragments, the quantifier rank and finite variable fragments cover the whole of first-order logic, so we call these comonads *expressive.*

Thus we have a strict hierarchy of comonads in the arboreal categories framework:

$$\text{idempotent} \Rightarrow \text{bisimilar companions} \Rightarrow \text{arboreal.}$$

This hierarchy is correlated with tractability: the modal and guarded fragments are decidable, and have the tree-model property [28, 17], while the expressive fragments do not. We can regard these observations as a small first step towards using structural properties of comonadic semantics to classify logic fragments and their expressive power. In [3], idempotence is used to give simple, general proofs of homomorphism preservation theorems for counting quantifier fragments, with an application to graded modal logic; while the bisimilar companion property is used to give a general, uniform Otto-style proof of van Benthem-Rosen theorems.

As we have already remarked, the hybrid comonads are closer to the Ehrenfeucht-Fraïssé comonads $\mathbb{E}_k$ than to the modal comonads $\mathbb{M}_k$. Indeed, the $\mathbb{H}_k$ comonads are neither idempotent, nor have the bisimilar companion property. On the tractability side, they are not decidable [7]. At the same time, they are not fully expressive for first-order logic, thus refining the above hierarchy.

Otto's proof of the van Benthem-Rosen theorem in [24] uses the bisimilar companion property. This is made explicit in the account given in [3]. Because $\mathbb{H}_k$ does not have this property, we shall use a different comonad in our invariance proof for the hybrid fragment.

Given a structure $\mathfrak{A}$, we can define a metric on A valued in the extended natural numbers $\mathbb{N} \cup \{\infty\}$, given by the path distance in the Gaifman graph $\mathcal{G}(\mathfrak{A})$ [20]. We set $d(a, b) = \infty$ if there is no path between a and b. We write $A[a; k]$ for the closed ball, centred on a, also referred to as the k-neighbourhood of a. Given $(\mathfrak{A}, a)$, we define $\mathbb{S}_k(\mathfrak{A}, a)$ to be $(\mathfrak{A}[a; k], a)$, where $\mathfrak{A}[a; k]$ is the substructure of $\mathfrak{A}$ induced by $A[a; k]$. This defines a comonad on $\mathsf{Struct}_\star(\sigma)$. The counit is the inclusion map, while coextension is the identity operation on morphisms, $h^* = h$. The fact that h is a σ-homomorphism implies that paths are preserved, so this is well defined. It is easily verified that $\mathbb{S}_k$ is an idempotent comonad. The corresponding coreflective subcategory of $\mathsf{Struct}_\star(\sigma)$ is the full subcategory of structures which are k-reachable from the initial elements. We can also define an idempotent comonad $\mathbb{S}$, where $\mathbb{S}(\mathfrak{A}, a) := \bigcup_{k \in \mathbb{N}} \mathbb{S}_k(\mathfrak{A}, a)$.

We can use this comonad to state the invariance property of interest. We say that a first-order formula $\varphi(x)$ is *invariant under generated substructures* if for all $(\mathfrak{A}, a)$ in $\mathsf{Struct}_\star(\sigma)$: $(\mathfrak{A}, a) \models \varphi \iff \mathbb{S}(\mathfrak{A}, a) \models \varphi$. It is *invariant under k-generated substructures* if for all $(\mathfrak{A}, a)$ in $\mathsf{Struct}_\star(\sigma)$: $(\mathfrak{A}, a) \models \varphi \iff \mathbb{S}_k(\mathfrak{A}, a) \models \varphi$. We use the standard disjoint union of structures, $\mathfrak{A} + \mathfrak{B}$. This is the coproduct in $\mathsf{Struct}(\sigma)$. We say that a sentence φ is *invariant under disjoint extensions* if for all $(\mathfrak{A}, a)$, $\mathfrak{B}$: $(\mathfrak{A}, a) \models \varphi \iff (\mathfrak{A} + \mathfrak{B}, a) \models \varphi$.

We can now state our main result.

▶ **Theorem 11** (Characterisation Theorem). *For any first-order formula $\varphi(x)$ with quantifier rank q, the following are equivalent:*

1. *φ is invariant under generated substructures.*
2. *φ is invariant under $q2^q$-generated substructures.*
3. *φ is invariant under disjoint extensions.*
4. *φ is equivalent to a sentence ψ of hybrid temporal logic with modal depth $\leq q2^q$.*

Note that this theorem has two versions, depending on the ambient category $\mathcal{C}$ relative to which equivalence is defined: $\forall(\mathfrak{A}, a) \in \mathcal{C}.\, (\mathfrak{A}, a) \models \varphi \iff (\mathfrak{A}, a) \models \psi$. The first version, for general models, takes $\mathcal{C} = \mathsf{Struct}_\star(\sigma)$. The second, for finite models, takes $\mathcal{C} = \mathsf{Struct}^f_\star(\sigma)$, the full subcategory of finite structures. Neither of these two versions implies the other. Following Otto [24], we aim to give a uniform proof, valid for both versions.

6.2 Proof of the Characterisation Theorem

Firstly, since any sentence can only use a finite vocabulary, we can assume without loss of generality in what follows that σ is finite. This implies that up to logical equivalence, the fragment HTL_k is finite.

Given a formula φ, we write $\mathsf{Mod}(\varphi) := \{(\mathfrak{A}, a) \mid (\mathfrak{A}, a) \models \varphi\}$. We shall use the following variation of a standard result.

▶ **Lemma 12** (Definability Lemma)**.** *For each $k > 0$ and structure $(\mathfrak{A}, a)$, there is a sentence $\theta^{(k)}_{(\mathfrak{A},\vec{a})} \in \mathsf{HTL}_k$ such that, for all $(\mathfrak{B}, b)$, $(\mathfrak{A}, a) \equiv^{\mathsf{HTL}}_k (\mathfrak{B}, \vec{b}) \iff (\mathfrak{B}, b) \models \theta^{(k)}_{(\mathfrak{A},\vec{a})}$.*

This says that $[(\mathfrak{A}, a)]_{\equiv^{\mathsf{HTL}}_k} = \mathsf{Mod}(\theta^{(k)}_{(\mathfrak{A},\vec{a})})$. Since $\equiv^{\mathsf{HTL}}_k$ has finite index, this implies that if $\mathsf{Mod}(\varphi)$ is saturated under $\equiv^{\mathsf{HTL}}_k$, φ is equivalent to a finite disjunction $\bigvee_{i=1}^n \theta^{(k)}_{(\mathfrak{A}_i,\vec{a}_i)}$, and hence to a formula in HTL_k.

The Workspace Lemma

A key step in the argument is a general result we call the Workspace Lemma. A special case of this is implicit in [24]. Note that $\equiv_q$ is elementary equivalence up to quantifier rank q.

▶ **Lemma 13** (Workspace Lemma)**.** *Given $(\mathfrak{A}, a)$ and $q > 0$, there is a structure $\mathfrak{B}$ such that $(\mathfrak{A} + \mathfrak{B}, a) \equiv_q (\mathfrak{A}[a; k] + \mathfrak{B}, a)$, where $k = 2^q$. Moreover, $|B| \leq 2q|A|$. Hence if $\mathfrak{A}$ is finite, so is $\mathfrak{B}$.*

The intuition for the workspace lemma is that the structure $\mathfrak{B}$, the *workspace*, contains enough disjoint copies of $\mathfrak{A}$ and $\mathfrak{A}[a; k]$ that Spoiler cannot tell the composite structures apart in q rounds of the Ehrenfeucht-Fraïssé game. The idea of Duplicator's strategy is that if Spoiler plays a move that is close to a previous position, in terms of distance in the Gaifman graph, Duplicator responds in the corresponding component of the other structure. If on the other hand Spoiler chooses a position that is "far away" from previously chosen positions, Duplicator responds in a fresh component of the appropriate type. The number of copies of both structures in the workspace, and the distances involved, are chosen so that everything is kept sufficiently far apart that Spoiler cannot see the difference between $\mathfrak{A} + \mathfrak{B}$ and $\mathfrak{A}[a; k] + \mathfrak{B}$. The formal argument is a delicate induction, which can be carried out at the level of generality of metric spaces.

We shall also require a few additional lemmas in our proof of the characterisation theorem. The following is immediate from the definitions.

▶ **Proposition 14.** *For each structure $(\mathfrak{A}, a)$ in $\mathsf{Struct}_\star(\sigma)$, we have $\mathbb{S}_k\mathbb{S}(\mathfrak{A}, a) = \mathbb{S}_k(\mathfrak{A}, a)$.*

The following lemma allows us to restrict our attention to generated substructures when considering HTL equivalence.

▶ **Lemma 15.** *For all $k, m > 0$, if $(\mathfrak{A}, a) \equiv^{\mathsf{HTL}}_m (\mathfrak{B}, b)$ then $\mathbb{S}_k(\mathfrak{A}, a) \equiv^{\mathsf{HTL}}_m \mathbb{S}_k(\mathfrak{B}, b)$.*

We also need the following result to strengthen HTL equivalence to first-order equivalence. We do so by lifting a Duplicator strategy for the hybrid game to one for the Ehrenfeucht-Fraïssé game, at the expense of some logical resources needed to traverse the structures step-by-step in the hybrid game.

▶ **Lemma 16.** *For all $k, q > 0$, if $\mathbb{S}_k(\mathfrak{A}, a) \equiv^{\mathsf{HTL}}_{kq} \mathbb{S}_k(\mathfrak{B}, b)$ then $\mathbb{S}_k(\mathfrak{A}, a) \equiv_q \mathbb{S}_k(\mathfrak{B}, b)$.*

Proof of the characterisation theorem

Proof of Theorem 11.

(2) ⇒ (1). Assume that φ is $\mathbb{S}_k$-invariant. Using Proposition 14, $(\mathfrak{A}, a) \models \varphi$ iff $\mathbb{S}_k(\mathfrak{A}, a) \models \varphi$ iff $\mathbb{S}_k\mathbb{S}(\mathfrak{A}, a) \models \varphi$ iff $\mathbb{S}(\mathfrak{A}, a) \models \varphi$.

(1) ⇒ (3). This follows immediately from the fact that $\mathbb{S}(\mathfrak{A} + \mathfrak{B}, a) = \mathbb{S}(\mathfrak{A}, a)$.

(3) ⇒ (4). Suppose that φ is invariant under disjoint extensions (abbreviated as IDE). Let $k = 2^q$. We shall use Lemma 12. Suppose that (i) $(\mathfrak{A}, a) \models \varphi$, and (ii) $(\mathfrak{A}, a) \equiv^{\mathsf{HTL}}_{kq} (\mathfrak{B}, b)$. We must show that $(\mathfrak{B}, b) \models \varphi$. Applying the Workspace Lemma twice, let $\mathfrak{C}$, $\mathfrak{D}$ be such that (iii) $(\mathfrak{A} + \mathfrak{C}, a) \equiv_q (\mathfrak{A}[a; k] + \mathfrak{C}, a)$ and (iv) $(\mathfrak{B} + \mathfrak{D}, b) \equiv_q (\mathfrak{B}[b; k] + \mathfrak{D}, b)$. From (ii), applying lemmas 15 and 16, we have (v) $\mathbb{S}_k(\mathfrak{A}, a) \equiv_q \mathbb{S}_k(\mathfrak{B}, b)$. Now

$$\begin{array}{rcll}
(\mathfrak{A}, a) \models \varphi & \Rightarrow & (\mathfrak{A} + \mathfrak{C}, a) \models \varphi & \mathsf{IDE} \\
& \Rightarrow & (\mathfrak{A}[a; k] + \mathfrak{C}, a) \models \varphi & (iii) \\
& \Rightarrow & \mathbb{S}_k(\mathfrak{A}, a) \models \varphi & \mathsf{IDE} \\
& \Rightarrow & \mathbb{S}_k(\mathfrak{B}, b) \models \varphi & (v) \\
& \Rightarrow & (\mathfrak{B}[b; k] + \mathfrak{D}, b) \models \varphi & \mathsf{IDE} \\
& \Rightarrow & (\mathfrak{B} + \mathfrak{D}, b) \models \varphi & (iv) \\
& \Rightarrow & (\mathfrak{B}, b) \models \varphi & \mathsf{IDE}
\end{array}$$

(4) ⇒ (2). We must show that if ψ is a formula in HTL_k, then it is invariant under k-generated substructures. This follows by a straightforward induction on syntax. ◀

▶ **Question 1.** *In his proof of the van Benthem-Rosen Theorem, Otto establishes an exponential succinctness gap between first-order logic and basic modal logic. A bisimulation-invariant first order formula of quantifier rank q has a modal equivalent of modal depth $\leq 2^q$. He shows that this is optimal. In our case, we have a gap of $q2^q$. Is this optimal for hybrid temporal logic?*

7 Further Directions

Everything which has been done for hybrid temporal logic in the present paper can be extended to the bounded fragment of first-order logic. This generalises the hybrid comonad, allowing both arbitrary relational vocabularies and constant symbols. Allowing for constants $c_1, \ldots, c_m$ involves working with the m-pointed category $\mathsf{Struct}_m(\sigma)$. This has objects $(\mathfrak{A}, \vec{a})$, where $\vec{a} = \langle a_1, \ldots, a_m \rangle \in A^m$. Morphisms $h : (\mathfrak{A}, \vec{a}) \to (\mathfrak{B}, \vec{b})$ must preserve these tuples. The intention is that $a_i = c_i^{\mathfrak{A}}$. Note that $\mathsf{Struct}_\star(\sigma) = \mathsf{Struct}_1(\sigma)$. The comonad constructions can be adapted smoothly to this setting, and the corresponding results go through without any problems.

Another variation is to consider hybrid logic without the backwards modalities $\Box^-$, $\Diamond^-$. The semantic significance of this is that directed rather than undirected reachability becomes the salient notion. The comonadic constructions can be adapted to this setting straightforwardly, but the semantic characterization results cannot be transferred directly. We leave the resolution of this issue to future work.

References

1 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, IEEE, 2017.

2 Samson Abramsky and Dan Marsden. Comonadic semantics for guarded fragments. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, IEEE, 2021.

3 Samson Abramsky and Luca Reggio. Aboreal categories: applications to preservation and invariance theorems, 2021. In preparation.

4 Samson Abramsky and Luca Reggio. Arboreal categories and resources. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 115:1–115:20. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

5 Samson Abramsky and Nihil Shah. Relating structure and power: Comonadic semantics for computational resources. *Journal of Logic and Computation*, 31(6):1390–1428, 2021.

6 Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In *International Conference on Foundations of Software Science and Computational Structures*, pages 297–311. Springer, 2010.

7 Carlos Areces, Patrick Blackburn, and Maarten Marx. Hybrid logics: Characterization, interpolation and complexity. *J. Symb. Log.*, 66(3):977–1010, 2001. doi:10.2307/2695090.

8 Patrick Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–365, 2000.

9 Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2002.

10 Patrick Blackburn and Klaus Frovin Jørgensen. Reichenbach, Prior and hybrid tense logic. *Synthese*, 193(11):3677–3689, 2016.

11 Torben Braüner. *Hybrid logic and its proof-theory*, volume 37. Springer Science & Business Media, 2010.

12 Adam Ó Conghaile. Cohomological k-consistency, 2021. Technical Report.

13 Adam Ó Conghaile and Anuj Dawar. Game comonads and generalised quantifiers. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, pages 16:1–16:17, 2021. arXiv:2006.16039.

14 Anuj Dawar, Tomáš Jakl, and Luca Reggio. Lovász-type theorems and game comonads. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470609.

15 Solomon Feferman. Persistent and invariant formulas for outer extensions. *Compositio Mathematica*, 20:29–52, 1968.

16 Solomon Feferman and Georg Kreisel. Persistent and invariant formulas relative to theories of higher order. *Bulletin of the American Mathematical Society*, 72(3):480–485, 1966.

17 Erich Grädel. Why are modal logics so robustly decidable? *Bull. EATCS*, 68:90–103, 1999.

18 Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of datalog: Tools and a case study. In Daniel J. Rosenkrantz and Yehoshua Sagiv, editors, *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, pages 61–71. ACM Press, 1990. doi:10.1145/298514.298542.

19 Azriel Lévy. *A hierarchy of formulas in set theory.* Number 57 in Memoirs of the American Mathematical Society. American Mathematical Soc., 1965.

20 Leonid Libkin. *Elements of Finite Model Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer, 2004.

21 Ernest G. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer Science & Business Media, 2012.

22 Yoàv Montacute and Nihil Shah. The pebble-relation comonad in finite model theory. *arXiv preprint*, 2021. `arXiv:2110.08196`.

23 Jaroslav Nešetřil and Patrice Ossona De Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

24 Martin Otto. Elementary proof of the van Benthem-Rosen characterisation theorem. Technical Report 2342, Department of Mathematics, Technische Universität Darmstadt, 2004.

25 Tom Paine. A pebbling comonad for finite rank and variable logic, and an application to the equirank-variable homomorphism preservation theorem. *Electronic Notes in Theoretical Computer Science*, 352:191–209, 2020.

26 Eric Rosen. Modal logic over finite structures. *Journal of Logic, Language and Information*, 6(4):427–439, 1997.

27 Johan van Benthem. *Modal logic and classical logic.* Bibliopolis, 1983.

28 Moshe Y Vardi. Why is modal logic so robustly decidable? Technical Report TR-97-274, Rice University, 1997.

The Complexity of Periodic Energy Minimisation

Duncan Adamson ✉
ICE-TCS, Department of Computer Science, Reykjavik University, Iceland

Argyrios Deligkas ✉
Department of Computer Science, Royal Holloway, University of London, UK

Vladimir V. Gusev ✉
Materials Innovation Factory, Department of Computer Science, University of Liverpool, UK

Igor Potapov ✉
Department of Computer Science, University of Liverpool, UK

Abstract

The computational complexity of pairwise energy minimisation of N points in real space is a long-standing open problem. The idea of the potential intractability of the problem was supported by a lack of progress in finding efficient algorithms, even when restricted the integer grid approximation. In this paper we provide a firm answer to the problem on $\mathbb{Z}^d$ by showing that for a large class of pairwise energy functions the problem of periodic energy minimisation is NP-hard if the size of the period (known as a unit cell) is fixed, and is undecidable otherwise. We do so by introducing an abstraction of pairwise average energy minimisation as a mathematical problem, which covers many existing models. The most influential aspects of this work are showing for the first time: 1) undecidability of average pairwise energy minimisation in general 2) computational hardness for the most natural model with periodic boundary conditions, and 3) novel reductions for a large class of generic pairwise energy functions covering many physical abstractions at once. In particular, we develop a new tool of overlapping digital rhombuses to incorporate the properties of the physical force fields, and we connect it with classical tiling problems. Moreover, we illustrate the power of such reductions by incorporating more physical properties such as charge neutrality, and we show an inapproximability result for the extreme case of the 1D average energy minimisation problem.

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

Keywords and phrases Optimisation of periodic structures, tiling, undecidability, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.8

Funding *Duncan Adamson*: supported by Icelandic Research Fund grant no. 217965.
Vladimir V. Gusev: The author is supported by the Leverhulme Trust via the Leverhulme Research Centre for Functional Materials Design.
Igor Potapov: partially supported by EP/R018472/1.

1 Introduction

Periodic structures and models with periodic boundary conditions appear both in nature and in mathematical interpretations of physical phenomena: spin systems in Ising models [8], Buckingham–Coulomb inter-atomic potential modelling crystal structures [32], Lennard-Jones potential in inter-molecular interaction [13]. Periodic boundary conditions are often used either to define, or to approximate, a large or infinite system from a small partition, known as a *unit cell*. The series of repeating unit cells in every dimension forms a periodic structure. A unit cell can be defined as a mapping from the points within a contiguous subspace of a lattice to a finite set of "colours", an abstraction that may be used to represent anything from ions to spin states. The main advantage of this model is that the periodic structure allows the properties of the effectively infinite global structure to be determined from the finite unit cell. This advantage has led to these structures attracting a great deal of attention in mathematics, physics, biology, chemistry and computer science [1, 12, 17, 18, 19, 20].

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 8; pp. 8:1–8:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

From the perspective of the physical sciences, one of the most fundamental properties is the *potential energy* of the structure, representing the sum of pairwise attractions between ions (or spins) within the unit cell. Usually, the energy landscape is a highly non-convex function with many local minima and saddle points. The fundamental optimisation problem, associated with predicting various physical phenomena in both periodic structures and a single unit cell, is known as the *cluster problem* [34]:

> "In which way can N points be occupied (in real space) so as to minimise the sum of their interactions?"

A lack of progress with the design of efficient algorithms for solving this fundamental optimisation problem led to various hypotheses about its intractability, which has not been formally addressed due to a large number of variants and complexity of practical details regarding the problem. However, very recently the first NP-hard result was shown for closely related *removal problem*: Given a cluster of N points, can a subset of them be removed to minimise the total energy of pairwise Buckingham–Coulomb interaction [3].

This paper builds upon these previous results, showing that hardness results apply even under the more realistic *periodic boundary conditions*. In the context of crystal structure prediction, this refers to the periodic structure of the crystal, meaning that the global structure of the crystal is represented by a repeating period. Previous work [3, 11] has focused on the interaction within the unit cell, while ignoring this periodic conditions.

In this paper, we propose a more universal approach and analyse the computational complexity of the original cluster problem for a large class of pairwise energy minimisation functions under more realistic periodic boundary conditions. We introduce an abstract class of r-distance *Common Minimum Value* functions ($\mathcal{CMV}(r)$), which capture typical properties of classical force-fields of pairwise interaction (attraction, repulsion) and incorporates the variable depth r of such interaction. Then we show that the cluster problem is NP-hard if the size of the unit cell is fixed and that it is undecidable otherwise for any function in the class $\mathcal{CMV}(r)$ defined on two or three dimensional grids ($\mathbb{Z}^2$ or $\mathbb{Z}^3$). Moreover, we show that particular known classical energy-interaction functions fit to this class and inherit the results on the computational complexity. In the case of 1D grids, we design a parameterised polynomial-time algorithm to solve the fixed period pairwise energy minimisation problem. Finally, we show that under the extra physical constraint of charge-neutrality (the total sum of charges/weights associated with points in the unit cell is zero) the problem still remains undecidable for 2 and 3 dimensions, and in dimension one it cannot be approximated within any constant factor unless $P = NP$.

Crystal Structure Prediction and Computational Complexity. Predicting crystal structures by computational methods without experimental input is the Holy Grail of crystallography and material science; it has remained a noted open problem for over 30 years [24]. In general, Crystal Structure Prediction (CSP) asks to identify the periodic crystal structure from a given set of *ions* – electromagnetically charged atoms – that minimise its potential energy based on some model of interaction.

A crystal is a structure defined by a repeating period called a unit cell. Informally, the unit cell can be thought of as a three dimensional box containing *ions*, see Figure 1. Each ion belongs to a class called a *species*, determining the properties of the ion. The unit cell acts as a periodic mapping from some set of ion species to the space $\mathbb{R}^3$, or in the discrete setting to a grid such as the integer grid $\mathbb{Z}^3$. In a discrete space, the unit cell can be represented as a necklace or bracelet [4, 2]. In the most general formulations of CSP the size and shape of the unit cell are unconstrained, however bounding the size and the shape is a common restriction for many cases of CSP [12].

Crystal structure prediction can be thought of as the problem of finding the "best" configuration of ions within a three-dimensional box, but it is also important for dimension one and two, see [12]. The quality of a configuration is determined by the average pairwise interaction between each ion in the structure. The pairwise interaction in turn is determined by an energy function, taking as input the distance between ions and a set of parameters based on the ion species. A positive interaction between ions corresponds to a force pushing the ions apart, while a negative interaction indicates a force of attraction. The goal of CSP is to find a stable structure, indicated by having the minimal average pairwise interaction [23].

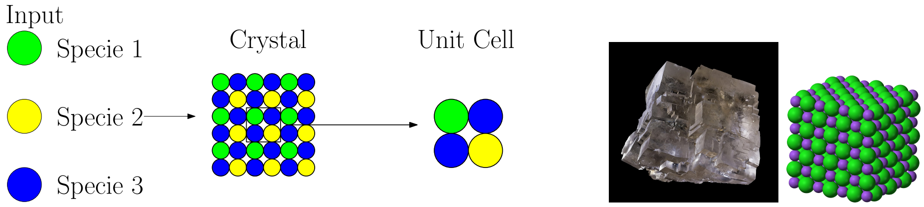


Figure 1 Left: A high level example of CSP. The input is a set of 3 species of ions (green, yellow, and blue), where each pair has an interaction computed by a given function determined by the distance and species. This set of species is transformed into a crystal structure (middle) defined by a unit cell (right). **Right:** Example of a Sodium Chloride crystal with the ionic structure. The middle and right pictures are shared under the creative commons licence.

Despite countless heuristics attempts such as quasi-random sampling [27, 29], basin hoping [6, 15, 16], simulated annealing [26, 30], swarm optimisation [9, 33], and genetic algorithms [14, 22, 25] the computational complexity of the CSP problem has not been investigated [10]. The recent interdisciplinary initiative to combine chemical knowledge with state of the art computer science techniques has lead to the first formalisation of CSP as a theoretical computer science problem [3, 6].

Our Contributions. This work introduces the average pairwise energy minimisation problem on $\mathbb{Z}^d$ as a generalisation of the physically motivated models and approximation of real space using the integer grid $\mathbb{Z}^d$. This problem can be seen as a variant on the well studied class of tiling problems [5, 7]. Rather than the "hard" constraints of a tiling problem, where tiles can only be placed adjacent to each other if they fulfil a set of strict conditions, our model uses "soft" constraints, giving an energy value to the interaction between each pair of vertices in the grid based on distance and the colour of the vertices.

Our main result is showing the average pairwise energy minimisation problem on $\mathbb{Z}^d$ to be NP-hard when the size of the unit cell is fixed and is undecidable otherwise. This strengthens the argument that CSP is intractable for a fixed-size unit cell and undecidable in general.

Our proof of both intractability and undecidability come by way of a series of reductions starting with the periodic tiling problem. This series of reductions is designed to enable us to more easily encode the concept of *orientation* into the pairwise interaction constraints that the average pairwise energy minimisation problem on $\mathbb{Z}^d$ uses. In the periodic tiling problem it is necessary for all tiles to have a shared orientation in order for the undecidability results to hold, however our physically motivated models determine interaction only by the colour of the vertices, and the distance between them. To encode this property we introduce two problems: the *k-unique radius tiling problem* (defined in Section 3.1), and the r-discretised rhombus all-distinct periodic complete assignment problem (defined in Section 3.2). Figure 2 provides a sketch of this process. We strengthen our results by showing that they hold under the further constraint of *charge neutrality*, an abstraction of the physical constraint that the periods of these structures must have an equal number of positive and negative charges.

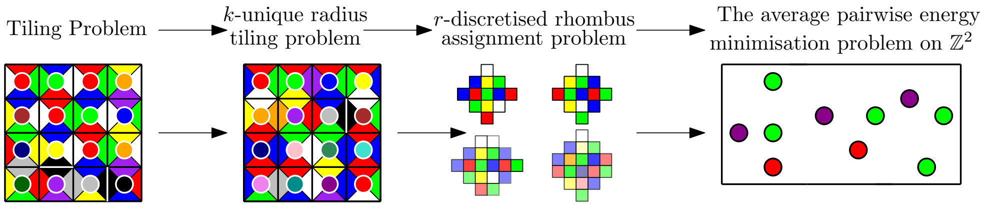

Tiling Problem
k-unique radius tiling problem
r-discretised rhombus assignment problem
The average pairwise energy minimisation problem on $\mathbb{Z}^2$

Figure 2 High level overview our series of reductions starting with the tiling problem.

2 Preliminaries

Informally, the average pairwise energy minimisation problem on $\mathbb{Z}^d$ can be thought of as the problem of determining a way of colouring the infinite grid $\mathbb{Z}^d$ with a *finite period*, while minimising the average pairwise interaction energy. The pairwise interaction energy between each pair of points on $\mathbb{Z}^d$ is determined by the colours of the points, and the distance between them. The period of a colouring is called the *unit cell*, which may equivalently be thought of as a mapping from the set of colours to the grid.

▶ **Definition 1** (Unit cell). *A* unit cell U *of size* $\vec{n} = (n_1, n_2, \ldots, n_d) \in \mathbb{N}^d$ *is a periodic mapping from the integer grid* $\mathbb{Z}^d$ *to some set of colours* $\mathcal{C}$*, defined by a colouring on the* d *dimensional grid* $n_1 \times n_2 \times \ldots \times n_d$*. Given a vector* $\vec{y} \in \mathbb{Z}^d$*,* $U(\vec{y})$ *returns the colour at position* $(y_1 \bmod n_1, y_2 \bmod n_2, \ldots, y_d \bmod n_d)$ *on the grid defining* U*.*

The number of vertices in a unit cell U of size $\vec{n}$ is denoted by $|U|$, i.e. $|U| = n_1 \cdot n_2 \cdot \ldots \cdot n_d$. Similarly $\vec{x} \in U$ is used to denote that $\vec{x}$ is a position in the finite grid defining U. Where it is clear from context, given any vector $\vec{x} \in \mathbb{Z}^d$ the colour of the vertex at position $\vec{x}$ in the grid $\mathbb{Z}^d$ coloured by U is denoted $U(\vec{x})$, giving $U(\vec{x}) = U((x_1 \bmod n_1, x_2 \bmod n_2, \ldots, x_d \bmod n_d))$.

The goal of these colourings is to minimise the *average pairwise energy per vertex* of the coloured grid. The energy between two vertices represents the force between them, with a negative energy indicating attraction and a positive energy indicating repulsion. The pairwise energy between a pair of vertices in the grid is determined by a *pairwise energy function.*

This work considers parametric pairwise energy functions f of the form $f(\overline{\theta}_{(c_i,c_j)}, r)$ where $c_i, c_j \in \mathcal{C}$ are a pair of colours, $r \in \mathbb{R}$ is a euclidean distance and $\overline{\theta}_{(c_i,c_j)} \in \mathbb{R}^p$ is a vector of p parameters determined by the colours c_i and c_j. Further, this work assumes that the vector of parameters $\overline{\theta}_{(c_i,c_j)}$ are predefined for every pair of colours $c_i, c_j \in \mathcal{C}$. Each function returns a scalar real value, i.e. $f\colon \left(\overline{\theta}_{(c_i,c_j)} \in \mathbb{R}^p, r \in \mathbb{R}\right) \mapsto \mathbb{R}$.

▶ **Definition 2** (Average pairwise energy per vertex). *Given a unit cell* U *of size* $\vec{n}$ *colouring the grid* $\mathbb{Z}^d$*, the* average pairwise energy per vertex *is given by:*

$$\mathrm{AE}(U) = \frac{1}{|U|} \sum_{\vec{x} \in U} \sum_{\vec{y} \in \mathbb{Z}^d} f(\overline{\theta}_{(U(\vec{x}),U(\vec{y}))}, D(\vec{x}, \vec{y}))$$

where f *is the pairwise energy function,* $D(\vec{x}, \vec{y})$ *denotes the euclidean distance between* $\vec{x}$ *and* $\vec{y}$*, and* $\overline{\theta}_{(c_i,c_j)} \in \mathbb{R}^p$ *is a vector of* p *parameters.*

In this paper we assume that each energy function has a cut off distance, allowing the average pairwise energy per vertex to be compute efficiently. One further constraint that we introduce is that of *charge neutrality.* In this setting, every colour is associated with a integer charge. Given a unit cell U, the charge of $\vec{x} \in U$ is denoted $Q(U(\vec{x}))$. Note that the charge of any two points assigned the same colour are equal, i.e. if $U(\vec{x}) = U(\vec{y})$ then

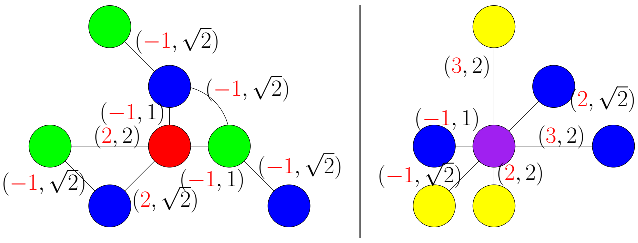


Figure 3 An overview of $\mathcal{CMV}(r)$. The energy and distance between each point is labelled as a pair (a, b) where a represents the energy and b represents the distance.

$Q(U(\vec{x})) = Q(U(\vec{y}))$ for any pair of vectors $\vec{x}, \vec{y} \in U$. A unit cell U is *charge neutral* if and only if $\sum_{\vec{x} \in U} Q(U(\vec{x})) = 0$. In general we assume that the charge neutrality constraint can be ignored, effectively assuming that the charge is 0 for every colour.

2.1 The r-Distance Common Minimal Value Class

In this paper we restrict our energy functions to the class of r-distance common minimal value functions, denoted $\mathcal{CMV}(r)$, introduced in this section. This class of functions focuses on the interactions between vertices within a distance of r of each other, for some distance $r \in \mathbb{R}$. In order to simplify reasoning on the set $\mathcal{CMV}(r)$, it is assumed that given any distance $d > r$ the value of $f(\overline{\theta}_{(c_i,c_j)}, d) = 0$ for every pair of colours $c_i, c_j \in \mathcal{C}$. Let $R(r) = \{(i,j) \in \mathbb{Z}^2 \mid \sqrt{i^2 + j^2} \leq r, (i,j) \neq (0,0)\}$ be the set of vertices on the integer grid $\mathbb{Z}^2$ within a distance of at most r of the central point $(0,0)$. Further, let $d(r) = \{\sqrt{i^2 + j^2} \mid (i,j) \in R(r)\}$ be the set of possible distances between the central vertex and any vertex in $R(r)$. As an example, $R(2) = \{(2,0), (1,1), (1,0), (1,-1), (0,2), (0,1), (0,-1), (0,-2), (-1,1), (-1,0), (-1,-1), (-2,0)\}$ and $d(r) = \{1, \sqrt{2}, 2\}$. The goal of this class is to be able to "fix" the optimal distance between pair of colours as either being some distance in $d(r)$, or as being outside of $R(r)$ – in effect penalising two colours at a distance of r or less. To this end this work uses the idea of a *common minimal value*. Informally, the common minimal value can be thought of as some negative value M such that the smallest possible interaction between any pair of vertices is M. Further, the functions in this work restrict M to appear at most once in the set of possible distance between each colour, meaning that given some pair of colours c_i and c_j, there exists at most one distance $d \in d(r)$ such that $f(\theta_{(c_i,c_j)}, d) = M$. The following definition formalises the *r-distance common minimal value class.*

▶ **Definition 3** (Common Minimal Value Functions ($\mathcal{CMV}(r)$)). *Let $\overline{\theta}_{(c_i,c_j)} \in \mathbb{R}^p$ denote the vector of parameters assigned to some pair of colours $c_i, c_j \in \mathcal{C}$. The function $f(\overline{\theta}_{(c_i,c_j)}, d)$: $\mathbb{R}^{p+1} \to \mathbb{R}$ belongs to the* **class of common minimal value functions** *$\mathcal{CMV}(r)$ for $r \in \mathbb{R}$ if there exists a common minimum value $M \in \mathbb{R}$ for which the following hold:*

1. *For any two points at a distance $d > r$ and any pair of colours c_i,$c_j \in \mathcal{C}$ the value of $f(\overline{\theta}_{(c_i,c_j)}, d) = 0$;* [**Cut-off property**].
2. *For any two colours c_i,$c_j \in \mathcal{C}$ it is possible to determine a vector $\overline{\theta}_{(c_i,c_j)}$ such that the energy between any pairs of points at any distance $d \in d(r)$ is $f(\overline{\theta}_{(c_i,c_j)}, d) > M$;* [**Separation property**].
3. *For any two colours c_i,$c_j \in \mathcal{C}$ and any distance $d \in d(r)$ it is possible to determine a vector $\overline{\phi}_{(c_i,c_j)}$ such that the energy between any pair of points at distance d is $f(\overline{\phi}_{(c_i,c_j)}, d) = M$ and the energy between any pair of points at any distance $d' \in d(r), d' \neq d$ is $f(\overline{\phi}_{(c_i,c_j)}, d') > M$;* [**Optimal pairwise distance property**].

An overview of these properties is given in Figure 3. These properties are used to encode the tiling problem into the average pairwise energy minimisation problem on $\mathbb{Z}^d$. The cut-off property (Property 1) ensures that there is no interaction between vertices over a certain cut off distance, allowing these interactions to be safely ignored. The separation property (Property 2) ensures that there exists a vector of parameters such that the corresponding colours must be placed further than r apart, or suffer a small energy penalty by having an interaction greater than M. Finally the optimal pairwise distance property (Property 3) ensures that there exists a vector of parameters such that the interaction of the corresponding colours is minimised at M at exactly one distance. The goal of these conditions is to be able to force a structure on the colouring based on the relative distances between colours. This allows the structure of the tiling problem to be utilised in the setting of the average pairwise energy minimisation problem on $\mathbb{Z}^d$.

2.2 The Pairwise Energy Minimisation Problem

This section introduces our central problem, the *average pairwise energy minimisation problem on* $\mathbb{Z}^d$. In this paper we consider two versions of this problem, depending on the constraints placed on the unit cell. In the most general case, the only constraint is that the average energy of the unit cell is below some bound g. In this paper, we limit the energy functions to the class $\mathcal{CMV}(r)$. The unit cell may be constrained by having the size given as part of the input. All values are given in binary as input to our problems.

▶ **Problem 1.** *The average pairwise energy minimisation problem on* $\mathbb{Z}^d$*.*

Input: A goal energy $g \in \mathbb{Q}$, a set of colours $\mathcal{C}$, a number of dimensions $d \in \mathbb{Z}$ an energy function $f \in \mathcal{CMV}$ and a set of $|\mathcal{C}|^2$ parameters $\overline{\theta}_{(c_i,c_j)} \in \mathbb{R}^p$.

Question: Does there exist a unit cell U of size $n_1 \times n_2 \times \ldots \times n_d$ for some $n_i \in \mathbb{N}^+$ where $\mathrm{AE}(U) \leq g$.

When the size of the unit cell is given as an input in the form of a vector of length d of the form $(n_1, n_2, \ldots, n_d)$, we refer to the problem as the **average pairwise energy minimisation problem on $\mathbb{Z}^d$ with a fixed period**. Here, *fixed period* refers to the size of the period being fixed as part of the input, in this case restricting the period to be of size $n_1 \times n_2 \times \ldots \times n_d$ for the given $n_1, n_2, \ldots, n_d$.

3 Undecidability for Unconstrained Period Size

We first look at the unbounded setting, where the size of the unit cell is not taken as part of the input. The main claim in this section is that Problem 1 is undecidable for any function in $\mathcal{CMV}(r)$ for $r \geq 2$. This section is split into three parts. First, we provide some background on the tiling problem that is used as the basis for this reduction. Second, we provide an auxiliary problem derived from the tiling problem to act as an intermediary step in proving the undecidablity of the average pairwise energy minimisation problem on $\mathbb{Z}^d$. Finally we prove the undecidablity of the average pairwise energy minimisation problem on $\mathbb{Z}^d$.

3.1 The Tiling Problem

In the tiling problem, we are given a set of *tiles*, square plates with a fixed orientation where each edge is coloured from some set of colours $\mathcal{C}$. The goal of a tiling problem is to completely cover the plane with tiles such that every pair of adjacent tiles is coloured the same along the shared edge. In this section, we introduce the further constraint that no two copies of the same tile may be within a distance of k or less of each other.

Before discussing the new variations of the tiling problem, let us first present some notation. The edges of the tiles are labelled *East, West, North* and *South* such that the East edge is opposite the West edge, and the South edge is opposite the North edge. More precisely, given two tiles, v at position (x_1, y_1) and u at position (x_2, y_2) respectively such that $|x_1 - x_2| \leq 1$ and $|y_1 - y_2| \leq 1$, we say that:

	$x_1 < x_2$	$x_1 = x_2$	$x_1 > x_2$
$y_1 < y_2$	v is **North-West** of u	v is **North** of u	v is **North-East** of u
$y_1 = y_2$	v is **West** of u	v is u	v is **East** of u
$y_1 > y_2$	v is **South-West** of u	v is **South** of u	v is **South-East** of u

A tile t is represented by the four edges composing it. For notation let t_e be the colour of the tile t along edge e. A tile t can be represented as $t = \{t_{East}, t_{South}, t_{West}, t_{North}\}$. A *Tile Set* is a set of tiles with the edges coloured from some set of colours $\mathcal{C}$. It is assumed the the tile set contains an infinite number of copies of each tile, allowing the complete plane to be covered with these tiles. The goal of the *Tiling problem* is to assemble copies of the tiles from a given tile set on an infinite plane ruled into squares of the size of one tile such that:

1. No tile is rotated or reflected.
2. A tile must be placed *exactly* over one square of the ruled integer plane.
3. The colour of adjacent edges must match.
4. Every square must be covered by one tile.

This problem is *solvable* for a given tile set if and only if such an assembly exists. An assembly is periodic if there exists some finite region of the plane that may be repeated so as to solve the tiling problem. The *Periodic Tiling Problem* asks if there is such a periodic assembly. Both the tiling problem and the periodic tiling problem are classical undecidable problems [5, 7]. Connections between this problem and chemistry are well established [28].

In this paper, we introduce the *k-unique radius* variant of the tiling problem to act as an intermediate problem between the general tiling problem and the average pairwise energy minimisation problem on $\mathbb{Z}^2$. The k-unique radius tiling problem is needed to help encode the notion of *orientation* into the average pairwise energy minimisation problem on $\mathbb{Z}^2$. In the tiling problem, it is integral that each tile is placed under the same orientation. This means that given two adjacent tiles, they must either touch West edge to East edge, or North edge to South edge. As our setting uses only the colours and distance between vertices to determine the pairwise energy, the concept of orientation is difficult to encode. Informally a tiling has a k-unique radius if and only if no two copies of a given tile are within a distance of at most k of each other. Let T be a tiling of $\mathbb{Z}^2$ such that $T(i, j)$ returns the tile at position (i, j). The tiling T has a k-unique radius if and only if for every $(i, j), (x, y) \in \mathbb{Z}^2$, where $D((i, j), (x, y)) \leq k$ the tile $T(i, j)$ is distinct from $T(x, y)$, i.e. $T(i, j) \neq T(x, y)$ where $D((i, j), (x, y))$ returns the distance between (i, j) and (x, y).

▶ **Problem 2.** *The periodic tiling problem with a k-unique radius.*

Input: A set of tiles, $\mathcal{T}$, and integer k
Question: Does there exist a periodic tiling of $\mathbb{Z}^2$ made from $\mathcal{T}$ such that given any tile t at position (x, y) there exists no other copy of t within a distance of k from (x, y)?

▶ **Proposition 4.** *The periodic k-unique radius tiling problem is undecidable for any $k \in \mathbb{N}$.*

Proof Sketch. The undecidability of the periodic k-unique radius tiling problem follows from the undecidability of the periodic domino problem [7]. The high level idea is to create a set of k^2 copies of each tile, labelled with $(x, y) \in [k]$. A set of additional colours are constructed such that given two tiles $t_{(x,y)}$ and $s_{(a,b)}$, $t_{(x,y)}$ can be placed adjacent to $s_{(a,b)}$ if and only if the original tiles t and s can be placed adjacently, and (a,b) is adjacent to (x,y) on the $k \times k$ toroidal grid. In one direction, any valid tiling with the original set of tiles can be transformed into a valid tiling for the new set by choosing an arbitrary origin point, and replacing the tile t at position (x,y) with the tile $t_{(x \bmod k, y \bmod k)}$. In the other direction, any tiling using the new tiles can be transformed into a tiling of the original tiles by simply replacing each tile $t_{(x,y)}$ with the tile t from the original set. ◀

▶ **Problem 3.** *The fixed period k-unique tiling problem.*

Input: A set of tiles, $\mathcal{T}$, integer k, and pair of lengths n_1, n_2.
Question: Does there exist a periodic tiling of the plane of size $n_1 \times n_2$ over $\mathcal{T}$ where every tile within a distance of k for every other tile is distinct.

▶ **Proposition 5.** *The fixed period k-unique tiling problem is NP-hard.*

Proof. Following the same arguments as in Proposition 4, the fixed period tiling problem can be reduced to the fixed period k-unique tiling problem. As the fixed period tiling problem is known to be NP-hard [5, 7, 21], the fixed period k-unique tiling problem is NP-hard. ◀

3.2 Tiling with Overlapping Digitised Rhombuses

This section covers the problem of completely covering the integer grid $\mathbb{Z}^2$ using *overlapping digitised rhombuses.* Informally, a *digitised rhombus* with radius r can be thought of as a set of mono-chromatically coloured tiles organised as a rhombus from some set of colours $\mathcal{C}$.

▶ **Definition 6** (Digitised rhombus). *A* ***digitised rhombus*** *of radius r is the mapping from the grid $\{(x,y) \in \mathbb{Z}^2 \mid |x| + |y| \leq r\}$ to a set of colours $\mathcal{C}$.*

Given a rhombus R and position $(i,j) \in \{(x,y) \in \mathbb{Z}^2 \mid |x| + |y| \leq r\}$, $R_{i,j}$ is used to denote the colour mapped by R to position (i,j), i.e. the colour of the tile at position (i,j) in the rhombus. A rhombus is *distinctly coloured* if $R_{i,j} \neq R_{l,m}$ for every pair of positions $(i,j),(l,m) \in \{(x,y) \in \mathbb{Z}^2 \mid |x| + |y| \leq r\}$ where $(i,j) \neq (l,m)$.

We use a set of rhombuses $\mathcal{R}$ analogously to the set of tiles $\mathcal{T}$ used in tiling problems. Given the integer grid $\mathbb{Z}^2$, the assignment of a rhombus R to the position $(x,y) \in \mathbb{Z}^2$ is equivalent to colouring every vertex within a radius of r of (x,y) using R. For the remainder of this section, we focus on the Manhattan distance, defined as $D((x_1,y_1),(x_2,y_2)) = |x_1 - x_2| + |y_1 - y_2|$. Given a rhombus R assigned to position $(x,y) \in \mathbb{Z}^2$, the position (x',y') at a distance of no more than r from (x,y) is coloured $R_{x'-x,y'-y}$.

The focus in this work is on *overlapping rhombuses.* Given a pair of digitised rhombuses of radius $r \in \mathbb{N}$, R and S at positions (x_1,y_1) and (x_2,y_2), R overlaps S if the Manhattan distance between R and S is no more than $2r$. The *overlap* between R and S are the set of positions that are assigned colours by both R and S. This corresponds to the set of positions $\{(a,b) \in \mathbb{Z}^2 \mid D((a,b),(x_1,y_1)) \leq r$ and $D((a,b),(x_2,y_2)) \leq r\}$. Informally, R and S properly overlap when centred at (x_1,y_1) and (x_2,y_2) if every position in the overlap is assigned the same colour by both R and S. See Figure 4 for an example.

▶ **Definition 7** (Overlapping rhombuses). *Let R and S be a pair of r-radius digitised rhombuses centred on positions (x_1,y_1) and (x_2,y_2) respectively. Rhombuses R and S* ***properly overlap*** *if and only if for every position $(i,j) \in \{(a,b) \in \mathbb{Z}^2 \mid D((a,b),(x_1,y_1)) \leq r$ and $D((a,b),(x_2,y_2)) \leq r\}$ it holds that $R_{i-x_1,j-y_1} = S_{i-x_2,j-y_2}$.*

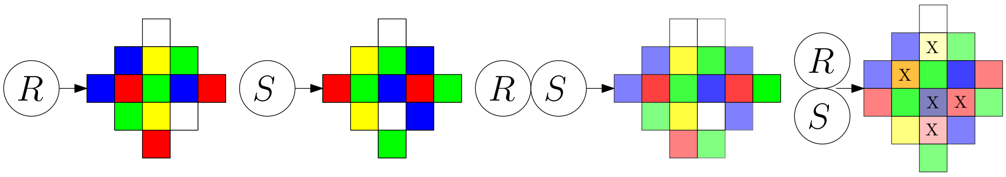


Figure 4 An example outlining how two rhombuses R and S may properly overlap (left) and when not (right). Note that X is used to denote a conflict between the two overlapping rhombuses.

Note that any two rhombuses at a distance greater than $2 \cdot r$ from each other properly overlap following Definition 7 as the set $\{(a,b) \in \mathbb{Z}^2 \mid D((a,b),(x_1,y_1)) \leq r$ and $D((a,b),(x_2,y_2)) \leq r\}$ is empty. An *assignment* of rhombuses $\mathcal{R}$ to the integer grid $\mathbb{Z}^2$ equates to a complete colouring of $\mathbb{Z}^2$ using the rhombuses in $\mathcal{R}$ as the colours. An assignment is valid if and only if a rhombus is centred on every vertex in $\mathbb{Z}^2$ and every pair of rhombuses properly overlap.

▶ **Definition 8** (Rhombus assignment). *An* ***assignment*** *A of rhombuses from the set $\mathcal{R}$ to $\mathbb{Z}^2$ is a mapping from $\mathbb{Z}^2$ to $\mathcal{R}$ such that $A : (x,y) \in \mathbb{Z}^2 \mapsto R$ for every $x,y \in \mathbb{Z}$. Let $A(x,y) : \mathbb{Z}^2 \mapsto \mathcal{R}$ return the rhombus assigned to position $(x,y) \in \mathbb{Z}^2$. An assignment A is* valid *if and only if $\forall (x_1,y_1),(x_2,y_2) \in \mathbb{Z}^2$, the rhombus $A(x_1,y_1)$ properly overlaps $A(x_2,y_2)$.*

▶ **Definition 9** (Periodic assignment). *An assignment A from $\mathbb{Z}^2$ to $\mathcal{R}$ is* ***periodic*** *if there exists a tuple $(a,b) \in \mathbb{Z}^2$ such that $A(x,y) = A(x \bmod a, y \bmod b)$ for every tuple $(x,y) \in \mathbb{Z}^2$.*

▶ **Problem 4.** *The r-discretised rhombus all-distinct periodic complete assignment problem.*

Input: A set $\mathcal{R}$ of r-radius digitised rhombuses
Question: Does there exist a valid periodic assignment of $\mathcal{R}$ to $\mathbb{Z}^2$?

▶ **Theorem 10.** *The r-discretised rhombus all-distinct periodic complete assignment problem is undecidable for any $r \geq 1$.*

Proof Sketch. This theorem is proven by transforming a set of tiles into a set of rhombuses. The set of rhombuses is constructed by taking the set of unique tilings on the grid $\{(x,y) \in \mathbb{Z}^2 \mid x^2 + y^2 \leq 1\}$, and creating a rhombus corresponding to each tiling. Each tile is represented in this model by a unique colour. ◀

The same construction as in Theorem 10 are used to derive an NP-completeness result for the fixed period r-discretised rhombus all-distinct periodic complete assignment problem. Observe that for a set of q rhombuses, and fixed period $n_1 \times n_2$, there are $q^{n_1 \cdot n_2}$ possible coverings, therefore a brute force algorithm can solve this problem in $O(q^{n_1 \cdot n_2})$-time, and therefore the problem belongs to NP. For hardness, Proposition 5 is used to establish the hardness of the fixed-period k-unique radius tiling problem, and by extension the hardness of the fixed period r-discretised rhombus all-distinct periodic complete assignment problem.

▶ **Corollary 11.** *The fixed period r-discretised rhombus all-distinct periodic complete assignment problem is NP-complete for any $r \geq 1$.*

3.3 Pairwise Energy Minimisation Problem on $\mathbb{Z}^2$ and $\mathbb{Z}^3$

With the undecidability of the r-discretised rhombus all-distinct periodic complete assignment problem established, the next step is to show how to reduce the r-discretised rhombus all-distinct periodic complete assignment problem to the average pairwise energy minimisation problem on $\mathbb{Z}^d$. In this section, the pairwise energy function is assumed to be a member of the 2-distance common minimal value class, $\mathcal{CMV}(2)$. At a high level the reduction from the r-discretised rhombus all-distinct periodic complete assignment problem is done by encoding each rhombus as a colour, then tuning the parameters of the pairwise energy function so that a valid colouring of the grid corresponds to a valid assignment of rhombuses.

The main challenge of this encoding is due to the definition of $\mathcal{CMV}(2)$. Namely, for any energy function in $\mathcal{CMV}(2)$ the pairwise energy between vertices is determined solely by the distance between vertices, colour of each vertex, and some given vector of parameters. This means that given two adjacent vertices v_i and v_j, coloured c_i and c_j respectively, the energy between v_i and v_j is the same irrespective of the relative direction of each tile.

To simplify our reduction, we introduce some additional notation. Given two vertices v_i at position (x_1, y_1) and v_j at position (x_2, y_2), v_i is said to be *directly adjacent* to v_j if $|x_1 - x_2| + |y_1 - y_2| = 1$. Similarly v_i is *diagonally adjacent* to v_j if $|x_1 - x_2| = 1$ and $|y_1 - y_2| = 1$. Finally, v_i is *peripherally adjacent* if either $|x_1 - x_2| = 2$ and $y_1 = y_2$, or $x_1 = x_2$ and $|y_1 - y_2| = 2$. Further, in Proposition 12 we assume M to be the common minimal value for all functions in $\mathcal{CMV}(r)$.

▶ **Proposition 12.** *Let $\mathcal{R}$ be a set of distinctly coloured rhombuses and let $\mathcal{C}(\mathcal{R})$ be a set of $|\mathcal{R}|$ colours such that for every rhombus $r \in \mathcal{R}$, there exists some colour $C_r \in \mathcal{C}(\mathcal{R})$. For any pairwise energy function $f \in \mathcal{CMV}(2)$, and pair of rhombuses $i, j \in \mathcal{R}$ there exists some vector of parameters $\overline{\theta}_{(c_i,c_j)}$ such that $f(\overline{\theta}_{(c_i,c_j)}, r)$ satisfies:*

1. *If i and j properly overlap when i is centred at some position directly adjacent to j then $f(\overline{\theta}_{(c_i,c_j)}, 1) = M$, and $f(\overline{\theta}_{(c_i,c_j)}, r) > M$ for any $r > 1$.*
2. *If i and j properly overlap when i is centred at some position diagonally adjacent to j then $f(\overline{\theta}_{(c_i,c_j)}, \sqrt{2}) = M$, and $f(\overline{\theta}_{(c_i,c_j)}, r) > M$ for either $r = 1$ or $r = 2$.*
3. *If i and j properly overlap when i is centred at some position peripherally adjacent to j then $f(\overline{\theta}_{(c_i,c_j)}, 2) = M$, and $f(\overline{\theta}_{(c_i,c_j)}, r) > M$ for $r < 2$.*
4. *Otherwise $f(\overline{\theta}_{(c_i,c_j)}, r) > M$ for any distance r.*

Proof. Recall that all functions in $\mathcal{CMV}(2)$ must have some vector of parameters $\overline{\theta}_{(c_i,c_j,d)} \in \mathbb{R}^p$ for every $d \in \{1, \sqrt{2}, 2\}$ such that $f(\overline{\theta}_{(c_i,c_j,d)}, d) = M$, and for every other distance distance $d' \in d(r)$ where $d' \neq d$ the value of the energy function $f(\overline{\theta}_{(c_i,c_j,d)}, d') > M$ by the optimal pairwise distance property (Property 3) of Definition 3. Therefore Conditions 1, 2, and 3 in the statement can be satisfied by choosing the appropriate vector of parameters for the distances of $1, \sqrt{2}$ and 2 respectively. Further, by the separation property (Property 2) there exists some vector of parameters $\overline{\theta}_{(c_i,c_j)} \in \mathbb{R}^p$ such that for every distance $d \in d(r)$ the energy $f(\overline{\theta}_{(c_i,c_j)}, d) > M$, satisfying Condition 4 above. ◀

Setting the parameter vectors so as to satisfy the conditions given in Proposition 12, Lemma 13 shows that a valid assignment of $\mathcal{R}$ to $\mathbb{Z}^2$ can be used to construct a valid colouring of $\mathbb{Z}^2$ using $\mathcal{C}(\mathcal{R})$. Lemma 14 shows that given such a colouring of $\mathbb{Z}^2$ using $\mathcal{C}(\mathcal{R})$, there must exist a valid assignment from $\mathbb{Z}^2$ to $\mathcal{R}$.

▶ **Lemma 13.** *Let A be a valid assignment of the set of distinctly coloured 2-radius rhombuses $\mathcal{R}$ to $\mathbb{Z}^2$ with a period of $n_1 \times n_2$. Given such an assignment there exists a periodic colouring of $\mathbb{Z}^2$ using the set of colours $\mathcal{C}(\mathcal{R})$ with an average energy per vertex of $12 \cdot M$.*

Proof Sketch. Observe that following Proposition 12 the interaction between any pair of colours within a distance of 2 is M. As A is a valid assignment of Rhombuses, the interaction between each point within a distance of at most 2 is M, giving an average energy of $12 \cdot M$. ◀

▶ **Lemma 14.** *Let U be a unit cell of size $n_1 \times n_2$ colouring $\mathbb{Z}^2$ with the colour set $\mathcal{C}(\mathcal{R})$ such that the average energy per vertex is $12 \cdot M$. Given such a unit cell there exists a valid assignment of the set of distinctly coloured 2-radius rhombuses $\mathcal{R}$ to $\mathbb{Z}^2$.*

Proof Sketch. The key observation behind this lemma is that given some vertex v at position (x, y) coloured with $c \in \mathcal{C}(\mathcal{R})$, there are exactly 4 colours $c_1, c_2, c_3, c_4 \in \mathcal{C}(\mathcal{R})$ that can be used to colour the vertices directly adjacent to v. Further, for the colour c_1 there exists exactly 1 colour that can be at a distance of 2 from a vertex coloured c_1 and a distance of 1 from a vertex coloured c. Therefore the local neighbourhood of each vertex must be coloured in such a way that the corresponding rhombuses correspond to a correct assignment. By extension, a valid colouring for the graph with an average energy per vertex of $12 \cdot M$ must correspond to a correct assignment of rhombuses to the plane $\mathbb{Z}^2$. ◀

▶ **Theorem 15.** *The average pairwise energy minimisation problem on $\mathbb{Z}^d$ is undecidable for any function in the 2-distance common minimal value class, and $d \in \{2, 3\}$.*

Proof. From Lemmas 13 and 14, there exists a valid colouring of $\mathbb{Z}^2$ with an average energy per vertex of $12M$ of $\mathcal{C}(\mathcal{R})$ if and only if there exists a valid assignment of $\mathcal{R}$ to $\mathbb{Z}^2$. As the r-discretised rhombus all-distinct periodic complete assignment problem is undecidable, the average pairwise energy minimisation problem on $\mathbb{Z}^2$ is undecidable.

To show the undecidability in 3D, consider the 3D tiling problem, where each tile is a 3D block with each face coloured. This problem is shown to be undecidable by reduction from the tiling problem. Let each block have a top and bottom face, along with the $North, West, South$ and $East$ faces. Given a set of tiles $\mathcal{T}$ a block b is constructed for each tile $t \in \mathcal{T}$ such that the colour of $North, West, South$ and $East$ faces of b match the corresponding colours of t, and the top and bottom faces of b are coloured with some universal colour c. See Figure 5 for an example. Observe that each plane of any valid tiling of these blocks on $\mathbb{Z}^3$ corresponds to a valid tiling of $\mathcal{T}$. As in the 2D setting, this problem can be restricted with the k-unique radius property. Similarly, k-unique radius tilings with 3D blocks can be converted into an all-distinct discretised rhombohedron in the same manner as the 2D case.

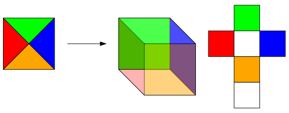


Figure 5 The transformation from a tile (left) to a 3 dimensional block (middle) and to an unfolded representation (right). The top and bottom faces are coloured with the same new colour.

An average pairwise energy minimisation problem on $\mathbb{Z}^d$ instance is constructed from these rhombohedrons in the same manner as in the 2D case. Note that an all discretised rhombohedron containing all points within a distance of 2 has 33 blocks. In this case, the average energy per vertex is $32 \cdot M$ if and only if there exists a valid tiling of $\mathbb{Z}^3$ of the original set of blocks. In one direction, if there exists such a tiling then the corresponding unit cell has an average energy per vertex of $32 \cdot M$. In the other direction, the same arguments as in the 2D case may be applied to show that any colouring with an average energy per vertex of $32 \cdot M$ corresponds to a valid tiling. ◀

Observe that the number of possible solution to the average pairwise energy minimisation problem on $\mathbb{Z}^2$ with a fixed period is at most $q^{n_1 \cdot n_2}$, where q is the number of tiles and (n_1, n_2) the size of the unit cell. Therefore, this problem is in NP. In the other direction the same arguments from Theorem 15 alongside Corollary 11 show the problem to be NP-hard.

▶ **Corollary 16.** *The fixed period average pairwise energy minimisation problem on $\mathbb{Z}^d$ is NP-complete for any function in the 2-distance common minimal value class, and $d \in \{2, 3\}$.*

4 Physically motivated pairwise energy functions

In this section, we apply the results from our abstract model to the problem of crystal structure prediction. In order to do so, we claim that the the *Buckingham-Coulomb* [32] and *Ising* [8] energy functions belong to $\mathcal{CMV}(2)$. The main results in this section, focus on *charge neutrality* in the context of the Buckingham-Coulomb potential, showing that the average pairwise energy minimisation problem on $\mathbb{Z}^d$ remains undecidable even with this restriction, and that the average pairwise energy minimisation problem on $\mathbb{Z}^d$ with a fixed period becomes hard to approximate within any positive factor in the 1D case. These results are strengthened in Section 4.2 by providing a parameterised algorithm for the average pairwise energy minimisation problem on $\mathbb{Z}$.

First, we outline the properties used by the Buckingham-Coulomb and 2-radius n-vector Ising energy functions to show that they belong to the class $\mathcal{CMV}(2)$.

The Buckingham-Coulomb Potential. The Buckingham-Coulomb energy between a pair of vertices coloured i and j at a distance of $r_{i,j}$ is given by the equation $BC(i, j, r_{i,k}) = \frac{A_{i,j}}{e^{B_{i,j} \cdot r_{i,j}}} - \frac{C_{i,j}}{r^6_{i,j}} + \frac{q_i \cdot q_j}{r_{i,j}}$ where $A_{i,j}, B_{i,j}$ and $C_{i,j}$ are a set of force field parameters, determined by the colours, q_i is the charge of colour i, and q_j is the charge of colour j. Here, we assume that rather than the integer grid, the Buckingham-Coulomb potential is performed on the set of points $\{(10 \cdot x_1, 10 \cdot x_2, \ldots, 10 \cdot x_d) \mid (x_1, x_2, \ldots, x_d) \in \mathbb{Z}^d\}$. In order to show that The Buckingham-Coulomb potential belongs to the class $\mathcal{CMV}(2)$, it is necessary to show that there exists a vector of parameters for each distance $d \in [10, \sqrt{200}, 20]$ such that (1) $BC(i, j, d) = M$, (2) $BC(i, j, d') > M$ for every $d' \in [10, \sqrt{200}, 20]$ where $d' \neq d$ and (3) there exists a vector of parameters such that $BC(i, j, d') > M$ for every $d' \in [10, \sqrt{200}, 20]$. Here $M = -1$ and the cutoff distance is set to 2. Conditions (1) and (2) are satisfied by using the faster convergence of the term $\frac{A_{i,j}}{e^{B_{i,j} \cdot r_{i,j}}}$ to 0 than the term $\frac{C_{i,j}}{r^6_{i,j}}$. Condition (3) can be satisfied by setting $A_{i,j}$ to a sufficiently large value, while setting $C_{i,j}$ to 0.

The 2-Radius n-Vector Ising Model. The second energy function we look at is a generalisation of the n-vector Ising model [31]. In the n-vector Ising model, each colour $c \in \mathcal{C}$ corresponds to a unit vector $\vec{c}$. Given a pair of adjacent vertices v and u, coloured c_v and c_u respectively, the energy between v and u is given by the dot product of the vectors, $\vec{c}_v \cdot \vec{c}_u$. In the 2-radius n-vector Ising model, each colour corresponds to a triple of n-length unit vectors. For notation, let $c[i]$ be the i^{th} vector in the triple corresponding to colour c. Given a pair of vertices v and u, coloured c_v and c_u respectively, the energy between v and u is given by $c_v[i] \cdot c_u[i]$ where i is 1 if v and u are at a distance of 1, 2 if v and u are at a distance of $\sqrt{2}$ or 3 if v and u are at a distance of 2. The value of each vector is chosen such that the product of any pair of vertices at distance d is either M, where M is some minimum value, or 0.

▷ Claim 17. The **Buckingham-Coulomb potential** and **2-radius n-vector Ising model** belong to $\mathcal{CMV}(2)$.

4.1 Charge Neutrality

In this section we focus on charge neutrality constraint. Recall that the charge of each colour, denoted $Q(c)$ is an integer value, and that a unit cell is charge neutral if $\sum_{\vec{x} \in U} Q(U(\vec{x})) = 0$.

▶ **Corollary 18.** *The Charge-Neutral average pairwise energy minimisation problem on $\mathbb{Z}^d$ with non-zero charges is undecidable for the Buckingham-Coulomb Potential for $d \in \{2, 3\}$.*

Theorem 19 compliments the proof of NP-hardness from Corollary 16 by showing that the charge-neutral fixed period pairwise energy minimisation problem is NP hard both to solve and to approximate within any constant factor for the Buckingham-Coulomb potential in 1D.

▶ **Theorem 19.** *The charge-neutral fixed period pairwise energy minimisation problem in 1D for the Buckingham-Coulomb potential can not be approximated within any constant factor in polynomial time unless $P = NP$.*

Proof Sketch. This theorem is proven via a reduction from the k-independent set problem. The high level idea is to construct a colour for each vertex in the input graph with a positive charge of $+1$, and a single negative ion of charge $-k$. The energy function is determined such that the interaction between any two colours representing adjacent vertices in the input graph is arbitrarily high, while the pairwise interaction between the positive and negative ions set to -1. This ensures that a valid solution to the charge-neutral fixed period pairwise energy minimisation problem can only be found if there exists an independent set. ◀

4.2 A Parameterised Algorithm for the 1D Setting

In this section we compliment the hardness results by providing a parameterised algorithm for solving the average pairwise energy minimisation problem on $\mathbb{Z}^d$ in 1D. Our algorithm provides solution in $O(n^3 \cdot q^{3 \cdot d})$ where n is the size of the unit cell, q is the number colours, and d is the cut off distance.

Construction. Given an instance of the fixed period pairwise energy minimisation problem for the 1D grid with length n, and set of colours $\mathcal{C}$, a graph G is constructed. Let $\mathcal{V}(d, \mathcal{C}) = \{(x_1, x_2, \dots, x_{d+1}) \mid x_1, x_2, \dots, x_{d+1} \in \mathcal{C}\}$. For notation given $l \in \mathcal{V}(d, \mathcal{C})$, l_i is used to denote the colour of the i^{th} position of l, i.e. given $l = (1, 2, 1, 2)$, $l_2 = 2$ while $l_3 = 1$. For every $i \in [n]$ and $l \in \mathcal{V}(d, \mathcal{C})$ the vertex $v_{i,l}$ is constructed and added to the set V of vertices. Given a pair vertices $v_{i,l}, v_{j,k} \in V$, the edge $v_{i,l}, v_{j,k}$ is added to the set of edges E if and only if $i + 1 = j$ and $l_2, l_3, \dots, l_{d+1} = k_1, k_2, \dots, k_d$. The weight of $(v_{i,l}, v_{j,k})$, denoted $w(v_{i,l}, v_{j,k})$, equals to $\sum_{i=1}^{d} f(i, \overline{\theta}_{(k_1, k_{i+1})})$. This means that each edge $(v_{i,l}, v_{j,k})$ corresponds to the pairwise interaction energy between k_1 and each subsequent vertex in k. In order to account for the energy from the first vector, an additional set of q^{d+1} vertices labelled v_l for every $l \in \mathcal{V}(d, \mathcal{C})$. The vertex v_l has only a single edge connecting it to $v_{1,l}$, weighted as before. Hence by constructing a path of length n starting at some vertex v_l and ending at the vertex $v_{n,l}$ the weight of the path with correspond to the total pairwise energy of the corresponding unit cell. Thus by finding such a path with minimum energy the solution to the fixed period pairwise energy minimisation problem may be found. Using the above construction, the solution to the fixed period pairwise energy minimisation problem instance is found by determining the shortest path from each vertex of the form v_l to the vertex $v_{n,l}$ for every $l \in \mathcal{V}(d, \mathcal{C})$. Note that this graph can be constructed in $O(((n+1) \cdot q^{d+1})^2)$ time for any energy function that can be computed in constant time, by simply constructing the full set of $(n+1) \cdot q^{d+1}$ vertices (corresponding to each position in the grid and list of d colours), and computing the energy between them using the energy function.

▶ **Theorem 20.** *There exists an algorithm to solve the fixed period pairwise energy minimisation problem in $O(n^3 \cdot q^{3 \cdot (d+1)})$ time for any function in $\mathcal{CMV}(d)$.*

Proof. Using the construction above, the solution to the corresponding fixed period pairwise energy minimisation problem instance can be found by using an efficient algorithm for solving the all pairs shortest path problem. Note the graph can be constructed in $O(((n+1) \cdot q^{d+1})^2) \approx O(n^2 \cdot q^{2(d+1)})$ time, assuming that the energy function can be evaluated in constant time. Using the Floyd–Warshall algorithm, the paths may be found in $O(|V|^3)$ time. Note that the number of vertices equals $(n+1) \cdot q^{d+1}$ giving a total complexity of $O\left((n+1)^3 \cdot q^{3\cdot(d+1)}\right) \approx O\left(n^3 \cdot q^{3\cdot(d+1)}\right)$. ◀

References

1 A. Adamatzky. On Diversity of Configurations Generated by Excitable Cellular Automata with Dynamical Excitation Intervals. *International Journal of Modern Physics C*, 23(12):1250085, November 2012. doi:10.1142/S0129183112500854.

2 D. Adamson. Ranking binary unlabelled necklaces in polynomial time. *24th International Conference on Descriptional Complexity of Formal systems (DFCS), Lecture Notes in Computer Science, Springer*, 2022.

3 D. Adamson, A. Deligkas, V. V. Gusev, and I. Potapov. On the hardness of energy minimisation for crystal structure prediction. *Fundamenta Informaticae*, 184:1–23, February 2022.

4 Duncan Adamson, Vladimir V. Gusev, Igor Potapov, and Argyrios Deligkas. Ranking Bracelets in Polynomial Time. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2021.4.

5 C. Allauzen and B. Durand. Tiling problems. In The Classical Decision Problems*, Perspectives in Mathematical Logic*, pages 407–420. Springer, 2001.

6 D. Antypov, A. Deligkas, V.V. Gusev, M. J. Rosseinsky, P. G. Spirakis, and M. Theofilatos. Crystal Structure Prediction via Oblivious Local Search. In *SEA 2020*, volume 160 of *LIPIcs*, pages 21:1–21:14, 2020.

7 R. Berger. *The undecidability of the domino problem.* Number 66 in memoirs of the american mathematical society. American Mathematical Soc., 1966.

8 A. Blanca, R. Gheissari, and E. Vigoda. Random-Cluster Dynamics in Z^2: Rapid Mixing with General Boundary Conditions. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 67:1–67:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.67.

9 S. T. Call, D. Y. Zubarev, and A. I. Boldyrev. Global minimum structure searches via particle swarm optimization. *Journal of computational chemistry*, 28(7):1177–1186, 2007.

10 R. Catlow and S. M. Woodley. Crystal structure prediction from first principles. *Nature materials*, 7(12):937, 2008.

11 B. A. Cipra. The Ising model is NP-complete. *SIAM News*, 33(6):1–3, 2000.

12 C. Collins, M.S. Dyer, M.J. Pitcher, G.F.S. Whitehead, M. Zanella, P. Mandal, J.B. Claridge, G.R. Darling, and M.J. Rosseinsky. Accelerated discovery of two crystal structure types in a complex inorganic phase field. *Nature*, 546(7657):280, 2017.

13 V. K de Souza and D. J Wales. The potential energy landscape for crystallisation of a Lennard-Jones fluid. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(7):074001, July 2016. doi:10.1088/1742-5468/2016/07/074001.

14 D. M. Deaven and K. M. Ho. Molecular geometry optimization with a genetic algorithm. *Physical review letters*, 75(2):288, 1995. Added up to here.

15 M. S. Dyer, C. Collins, D. Hodgeman, P. A. Chater, A. Demont, S. Romani, R. Sayers, M. F. Thomas, J. B. Claridge, G. R. Darling, and M. J. Rosseinsky. Computationally assisted identification of functional inorganic materials. *Science*, 340(6134):847–852, 2013.

16 S. Goedecker. Minima hopping: An efficient search method for the global minimum of the potential energy surface of complex molecular systems. *The Journal of chemical physics*, 120(21):9911–9917, 2004.

17 L. A. Goldberg and H. Guo. The Complexity of Approximating complex-valued Ising and Tutte partition functions. *Comput. Complex.*, 26(4):765–833, 2017. doi:10.1007/s00037-017-0162-2.

18 L. A. Goldberg and M. Jerrum. Approximating Pairwise Correlations in the Ising Model. *ACM Trans. Comput. Theory*, 11(4):23:1–23:20, 2019. doi:10.1145/3337785.

19 M. Hill, S. Stepney, and F. Wan. Penrose Life: Ash and Oscillators. In Mathieu S. Capcarrère, Alex Alves Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Advances in Artificial Life, 8th European Conference, ECAL 2005, Canterbury, UK, September 5-9, 2005, Proceedings*, volume 3630 of *Lecture Notes in Computer Science*, pages 471–480. Springer, 2005. doi:10.1007/11553090_48.

20 J. Kari and E. Moutot. Decidability and Periodicity of Low Complexity Tilings. In C. Paul and M. Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 14:1–14:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.14.

21 H. R. Lewis. Complexity of solvable cases of the decision problem for the predicate calculus. In *19th Annual Symposium on Foundations of Computer Science (FOCS 1978)*, pages 35–47, 1978. doi:10.1109/SFCS.1978.9.

22 D. C. Lonie and E. Zurek. XtalOpt: An open-source evolutionary algorithm for crystal structure prediction. *Computer Physics Communications*, 182(2):372–387, 2011.

23 A. O. Lyakhov, A. R. Oganov, and M. Valle. How to predict very large and complex crystal structures. *Computer Physics Communications*, 181(9):1623–1632, 2010.

24 J. Maddox. Crystals from first principles. *Nature*, 335(6187):201–201, 1988. doi:10.1038/335201a0.

25 A. R. Oganov and C. W. Glass. Crystal structure prediction using ab initio evolutionary techniques: Principles and applications. *The Journal of chemical physics*, 124(24), 2006.

26 J. Pannetier, J. Bassas-Alsina, J. Rodriguez-Carvajal, and V. Caignaert. Prediction of crystal structures from crystal chemistry rules by simulated annealing. *Nature*, 346(6282):343–345, 1990.

27 R.J. Pickard, C. J .and Needs. Ab initio random structure searching. *Journal of Physics: Condensed Matter*, 23(5):053201, 2011.

28 Z. Rotman and E. Eisenberg. Finite-temperature liquid-quasicrystal transition in a lattice model. *Phys. Rev. E*, 83:011123, January 2011. doi:10.1103/PhysRevE.83.011123.

29 M. U. Schmidt and U. Englert. Prediction of crystal structures. *Journal of the Chemical Society, Dalton Transactions*, 10:2077–2082, 1996.

30 J. C. Schön and M. Jansen. First step towards planning of syntheses in solid-state chemistry: determination of promising structure candidates by global optimization. *Angewandte Chemie International Edition in English*, 35(12):1286–1304, 1996.

31 H. E. Stanley. Dependence of Critical Properties on Dimensionality of Spins. *Phys. Rev. Lett.*, 20:589–592, March 1968.

32 D. J. Wales. Exploring Energy Landscapes. *Annual Review of Physical Chemistry*, 69(1):401–425, 2018. PMID: 29677468. doi:10.1146/annurev-physchem-050317-021219.

33 Y. Wang, J. Lv, L. Zhu, S. Lu, K. Yin, Q. Li, H. Wang, L. Zhang, and Y. ma. Materials discovery via CALYPSO methodology. *Journal of physics. Condensed matter : an Institute of Physics journal*, 27:203203, April 2015.

34 L. T. Wille and J. Vennik. Computational complexity of the ground-state determination of atomic clusters. *Journal of Physics A: Mathematical and General*, 18(8):L419–L422, June 1985.

Weighted Counting of Matchings in Unbounded-Treewidth Graph Families

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Mikaël Monet
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

We consider a weighted counting problem on matchings, denoted PrMatching($\mathcal{G}$), on an arbitrary fixed graph family $\mathcal{G}$. The input consists of a graph $G \in \mathcal{G}$ and of rational probabilities of existence on every edge of G, assuming independence. The output is the probability of obtaining a *matching* of G in the resulting distribution, i.e., a set of edges that are pairwise disjoint. It is known that, if $\mathcal{G}$ has bounded *treewidth*, then PrMatching($\mathcal{G}$) can be solved in polynomial time. In this paper we show that, under some assumptions, bounded treewidth in fact *characterizes* the tractable graph families for this problem. More precisely, we show intractability for all graph families $\mathcal{G}$ satisfying the following *treewidth-constructibility* requirement: given an integer k in unary, we can construct in polynomial time a graph $G \in \mathcal{G}$ with treewidth at least k. Our hardness result is then the following: for *any* treewidth-constructible graph family $\mathcal{G}$, the problem PrMatching($\mathcal{G}$) is intractable. This generalizes known hardness results for weighted matching counting under some restrictions that do not bound treewidth, e.g., being planar, 3-regular, or bipartite; it also answers a question left open in [1]. We also obtain a similar lower bound for the weighted counting of edge covers.

2012 ACM Subject Classification Mathematics of computing → Matchings and factors

Keywords and phrases Treewidth, counting complexity, matchings, Fibonacci sequence

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.9

Related Version *Full Version*: `https://arxiv.org/abs/2205.00851` [4]

Supplementary Material *Software (Computer calculations)*:
`https://gitlab.com/Gruyere/supplementary-material-for-Weighted-Counting-of-Matchings`, archived at `swh:1:dir:b01ffb42985d203c5cc1510dad134c9696e2d1f4`

Funding *Antoine Amarilli*: Partially supported by the ANR project EQUUS ANR-19-CE48-0019 and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758.

Acknowledgements We thank Florent Capelli and Sébastien Tavenas for discussions on the problem, in particular on how to prove Proposition 5.2.

1 Introduction

Many complexity results on computational problems rely on a study of fundamental graph patterns such as independent sets, vertex covers, edge covers, matchings, cliques, etc. In this paper we specifically study *counting problems* for such patterns, and for the most part focus on counting the *matchings*: given an input graph G, we wish to count how many edge subsets of G are a matching, i.e., each vertex has at most one incident edge.

Our goal is to address an apparent gap between the existing intractability and tractability results for counting matchings and similar patterns. On the one hand, counting the matchings is known to be #P-hard, and hardness is known even when the input graph is restricted in certain ways, e.g., being planar, being 3-regular, or being bipartite [18, 14, 27, 26]. On

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 9; pp. 9:1–9:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the other hand, some restrictions can make the problem tractable, e.g., imposing that the input graphs have *bounded treewidth* [5, 1], because matchings can be described in monadic second-order logic. But this does not settle the complexity of the problem; could there be other restrictions on graphs that makes it tractable to count matchings or other patterns?

This paper answers this question in the negative, for a *weighted* version of counting problems: we show that, at least for matchings and edge covers, and under a technical assumption on the graph family, the weighted counting problem is intractable if we do not bound the treewidth of the input graphs. Thus, treewidth is the right parameter to ensure tractability. Our weighted counting problems are of the following form: we fix a graph family $\mathcal{G}$ (e.g., 3-regular graphs, graphs of treewidth $\leqslant 2$), we are given as input a graph G of $\mathcal{G}$ along with an independent probability of existence for each edge, and the goal is to compute the probability in this distribution of the subsets of edges of G which have a certain property, e.g., they are a matching, they are an edge cover. Note that the class $\mathcal{G}$ restricts the shape of the graphs, but the edge probabilities are arbitrary – and indeed there are known tractability results when we restrict the graphs and probabilities to be symmetric [6]. Our paper shows the hardness of these problems when $\mathcal{G}$ is not of bounded treewidth; the specific technical assumption on $\mathcal{G}$ is that one can effectively construct graphs of $\mathcal{G}$ having arbitrarily high treewidth, i.e., the *treewidth-constructible* requirement from [1] (cf. Definition 2.2):

▶ **Result 1.** *Let $\mathcal{G}$ be an arbitrary family of graphs which is treewidth-constructible. Then the problem, given a graph $G = (V, E)$ of $\mathcal{G}$ and rational probability values $\pi(e)$ for every edge of G, of computing the probability of a matching in G under π, is #P-hard under ZPP reductions.*

We obtain an analogous result for edge covers. Thus, as bounded-treewidth makes the problems tractable, our results imply that treewidth characterizes the tractable graph families for these problems – for weighted counting, and assuming treewidth-constructibility. We leave open the complexity of unweighted counting, and of weighted counting on graph families that have unbounded treewidth but satisfy weaker requirements than treewidth-constructibility, e.g., being strongly unbounded poly-logarithmically [16, 13].

The paper is devoted to showing Result 1. Because of the page limit, the full proofs are deferred to the full version [4]. At a high level, we use the standard technique of reducing from the #P-hard problem of counting matchings on a 3-regular planar graph G [26], using the randomized polynomial-time grid minor extraction result of [10] as in [1]. However, the big technical challenge is to reduce the counting of matchings of G to the problem of computing the probability of a matching on the arbitrary subdivision G' of G that we extract. For this, we use the classical interpolation method, where we design a linear equation system relating the matchings to the result of polynomially many oracle calls on G', with different probability assignments; and we argue that the matrix is invertible. After the preliminaries (Section 2), we present this proof, first in the case where G' is a 6-subdivision of G (Section 3), and then when it is a n-subdivision, i.e., when all edges are subdivided to the same length n (Section 4). These special cases already pose some difficulties, most of which are solved by adapting techniques by Dalvi and Suciu [12]; e.g., to show invertibility, we study the Jacobian determinant of the mapping associating edge probabilities to the probability of matchings on paths with fixed endpoints, and we borrow a technique from [12] to effectively construct suitable rational edge probabilities.

The main novelties of this work are in Section 5, where we extend the proof to the general case: G' is a subdivision of G, and different edges of G may be subdivided in G' to different lengths. To obtain the equation system, we show that we can assign probabilities on short paths so that they "behave" like long paths. Proving this stand-alone *emulation result*

(Proposition 5.2) was the main technical obstacle; the proof is by solving a system of equations involving the Fibonacci sequence. It also introduces further complications, e.g., dealing with numerical error (because the resulting probabilities are irrational), and distinguishing even-length and odd-length subdivisions. After concluding the proof of Result 1 in Section 5, we adapt it in Section 6 to edge covers.

Related work. Our work follows a line of results that show the intractability of some problems on any "sufficiently constructible" unbounded-treewidth graph family. Kreutzer and Tazari [16] (see also [13]) show that there are formulas in an expressive formalism (MSO_2) that are intractable to check on any subgraph-closed unbounded treewidth graph family that is closed under taking subgraphs and satisfies a requirement of being *strongly unbounded poly-logarithmically*. This was extended in [1] to the weighted counting problem, this time for a query in first-order logic, with a different hardness notion (#P-hardness under randomized reductions), and under the stronger requirement of treewidth-constructibility. Our focus here is to show that the hardness of weighted counting already holds for natural and well-studied graph properties, e.g., "being a matching"; this was left as an open problem in [1].

For such weak patterns, lower bounds were shown in [1] and [3] on the *size of tractable representations*: for any graph G of bounded degree having treewidth k, any so-called *d-SDNNF* circuit representing the matchings (or edge covers) of G must have exponential size in k. However, this does not imply that the problems are intractable, as some tractable counting algorithms do not work via such circuit representations (e.g., the one in [12]). Thus, our hardness result does not follow from this size bound, but rather complements it.

The necessity of bounded treewidth has also been studied for graphical models [9] and Bayesian networks [17]. Specifically, [17] shows the intractability of inference in a Bayesian network as a function of the treewidth (but without otherwise restricting the class of network), and [9] restricts the shape of the graphical model but allows arbitrary "potential functions" (whereas we assume independence across edges). There are also necessity results on treewidth for the problem of *counting the homomorphisms* between two structures in the CSP context [11]; but this has no clear relationship to our problems, where we do (weighted) *counting of the substructures* that have a certain form (e.g., are matchings).

Note that, unlike our problem of weighted counting of matchings, the problem of *finding* a matching of maximal weight in a weighted graph is tractable on arbitrary graphs, using Edmond's blossom algorithm [21].

2 Preliminaries

We write $\mathbb{N}^+$ for $\mathbb{N}\setminus\{0\}$, and for $n \in \mathbb{N}^+$ we write $[n]$ the set $\{0, \ldots, n-1\}$. We write $\mathbb{R}$ the real numbers and $\mathbb{Q}$ the rational numbers. Recall that *decimal fractions* are rational numbers that can be written as a fraction $a/10^k$ of an integer a and a power of ten 10^k.

Reductions and complexity classes. Recall that #P is the class of counting problems that count the number of accepting paths of a nondeterministic polynomial-time Turing machine. A problem P_1 is #P-*hard* if every problem P_2 of #P reduces to P_1 in polynomial time; following Valiant [19, 20], we use here the notion of *Turing reductions*, i.e., P_2 can be solved in polynomial time with an oracle for P_1. We specifically study what we call #P*-hardness under zero-error probabilistic polynomial-time (ZPP) reductions*. To define these, we define a *randomized algorithm* as an algorithm that has access to an additional random tape. We say that a decision problem is in *ZPP* if there is a randomized algorithm that (always) runs in polynomial time on the input instance, and returns the correct answer on the instance (i.e., accepting or rejecting) with some constant probability, and otherwise returns a special failure value. The probabilities are taken over the draws of the contents of the random tape. The exact value of the acceptance probability is not important, because we can make it exponentially small by simply repeating the algorithm polynomially many times. Going beyond decision problems, a *ZPP algorithm* is a randomized algorithm that runs in polynomial time but may return a special failure value with some constant probability. A *ZPP (Turing) reduction* from a problem P_1 to a problem P_2 is then a ZPP algorithm having access to an oracle for P_2 that takes an instance of problem P_1, runs in polynomial time, returns the correct output (for P_1) with some constant probability, and returns the special failure value otherwise. Again, the failure probability can be made arbitrarily small by invoking the reduction multiple times. A problem P_2 is then said to be *#P-hard under ZPP reductions* if any #P-hard problem P_1 has a ZPP reduction to it. We will implicitly rely on the fact that we can show #P-hardness under ZPP reductions by reducing in ZPP from any problem which is #P-hard (under Turing reductions); see the full version [4] for details.

Graphs and problem studied. A finite undirected *graph* $G = (V, E)$ consists of a finite set V of *vertices* (or *nodes*) and of a set E of *edges* of the form $\{x, y\}$ for $x, y \in V$ with $x \neq y$. A *graph family* $\mathcal{F}$ is a (possibly infinite) set of graphs. For $v \in V$, we write $\mathcal{E}_G(v)$ for the set of edges that are incident to v. Recall that a *matching* of G is a set of edges $M \subseteq E$ that do not share any vertices, i.e., for every $e, e' \in M$ with $e \neq e'$ we have $e \cap e' = \emptyset$; or equivalently, we have $|\{\mathcal{E}_G(v) \cap M\}| \leqslant 1$ for all $v \in V$. For a graph family $\mathcal{F}$, we write #Matching($\mathcal{F}$) the problem of counting the matchings for graphs in $\mathcal{F}$: the input is a graph $G \in \mathcal{F}$, and the output is the number of matchings of G, written #Matching(G).

We study a weighted version of #Matching, defined on *probabilistic graphs.* A *probabilistic graph* is a pair (G, π) where $G = (V, E)$ is a graph and $\pi \colon E \to [0, 1]$ maps every edge e of H to a probability value $\pi(e)$. The probabilistic graph (G, π) defines a probability distribution on the set of subsets E' of E, where each edge $e \in E$ is in E' with probability $\pi(e)$, assuming independence across edges. Formally, the probability of each subset E' is:

$$\Pr_{G,\pi}(E') := \prod_{e \in E'} \pi(e) \times \prod_{e \in E \setminus E'} (1 - \pi(e)).$$

Given a probabilistic graph (G, π), the *probability of a matching in G under pi*, denoted $\Pr_{\text{matching}}(G, \pi)$, is the probability of obtaining a matching in the distribution. Formally:

$$\Pr_{\text{matching}}(G, \pi) \ := \sum_{\text{matching } M \text{ of } G} \Pr_{G,\pi}(M). \tag{1}$$

In particular, if π maps every edge to the probability 1/2, then we have $\Pr_{\text{matching}}(G, \pi) = \#\text{Matching}(G)/2^{|E|}$. For a graph family $\mathcal{F}$, we will study the problem PrMatching($\mathcal{F}$) of computing the probability of a matching: the input is a probabilistic graph (G, π) where $G \in \mathcal{F}$ and π is an arbitrary function with rational probability values, and the output is $\Pr_{\text{matching}}(G, \pi)$. Note that $\mathcal{F}$ only specifies the graph G and not the probabilities π, in particular π can give probability 0 to edges, which amounts to removing them.

Treewidth and topological minors. Treewidth is a parameter mapping any graph G to a number $\text{tw}(G)$ intuitively describing how far G is from being a tree. We omit the formal definition of treewidth (see [25]), as we only rely on the following *extraction result*: given any *planar graph H of maximum degree* 3, and a graph G of *sufficiently high* treewidth, it is possible (in randomized polynomial time) to find H as a *topological minor* of G. We now define this.

The *degree* of a node v in $H = (V_H, E_H)$ is simply $|\mathcal{E}_G(v)|$. We say that H is 3-*regular* if every vertex has degree 3, and call H *planar* if it can be drawn on the plane without edge crossings, in the usual sense [24]. Given H and $\eta\colon E_H \to \mathbb{N}^+$, the *$\eta$-subdivision of H*, written $\mathrm{Sub}(H, \eta)$, is the graph obtained from H by replacing every edge $e = \{x, y\}$ by a path of length $\eta(e)$, whose end vertices are identified with x and y, all intermediate vertices being fresh across all edges. We abuse notation and write $\mathrm{Sub}(H, i)$ for $i \in \mathbb{N}_+$ to mean $\mathrm{Sub}(H, \eta_i)$ for η_i the constant-i function. Note that $\mathrm{Sub}(H, 1) = H$. A *subgraph* of a graph $G = (V_G, E_G)$ is a graph (V'_G, E'_G) where $E'_G \subseteq E_G$ and $V'_G \subseteq V_G$ such that $e \subseteq V'_G$ for each edge $e \in E'_G$. The graph $H = (V_H, E_H)$ is a *topological minor* of the graph $G = (V_G, E_G)$ if there is a function $\eta\colon E_H \to \mathbb{N}^+$ such that there is an isomorphism f from the subdivision $\mathrm{Sub}(H, \eta) = (V'_H, E'_H)$ to some subgraph $G' = (V'_G, E'_G)$ of G, i.e., a bijection $f\colon V'_H \to V'_G$ such that for every $x, y \in V'_H$ we have $\{x, y\} \in E'_H$ if and only if $\{f(x), f(y)\} \in E'_G$.

We can now state the extraction result that we use, which follows from the work of Chekuri and Chuzhoy [10]:

▶ **Theorem 2.1** (Direct consequence of [10], see, e.g., [1], Lemma 4.4)**.** *There exists $c \in \mathbb{N}$ and a ZPP algorithm*[1] *that, given as input a planar graph $H = (V_H, E_H)$ of maximum degree 3 and another graph G with $\mathrm{tw}(G) \geqslant |V_H|^c$, computes a subgraph G' of G, a function $\eta : V_H \to \mathbb{N}^+$, and an isomorphism from Sub(H, η) to G' (witnessing that H is a topological minor of G).*

Our intractability result will apply to graph families where large treewidth graphs can be efficiently found, which we formalize as *treewidth-constructibility* like in [1]:

▶ **Definition 2.2.** *A graph family $\mathcal{F}$ is* treewidth-constructible *if there is a polynomial-time algorithm that, given an integer k written in unary*[2]*, outputs a graph $G \in \mathcal{F}$ with $\mathrm{tw}(G) \geqslant k$.*

Kronecker products and Vandermonde matrices. To simplify notation, we will work with matrices indexed with arbitrary finite sets (not necessarily ordered). Given two finite sets I, J of same cardinality, we write $\mathbb{R}^{I,\,J}$ (resp., $\mathbb{Q}^{I,\,J}$) the set of matrices with real values (resp., rational values) whose rows are indexed by I and columns by J. When $\boldsymbol{A} \in \mathbb{R}^{I,\,J}$ and $(i, j) \in I \times J$, we write $a_{i,j}$ the corresponding entry. We recall that the inverse of an invertible matrix M with entries in $\mathbb{Q}$ also has entries in $\mathbb{Q}$ and can be computed in polynomial time in the encoding size of M.

Given two matrices $\boldsymbol{A} \in \mathbb{R}^{I,\,J}$ and $\boldsymbol{B} \in \mathbb{R}^{K,\,L}$, the *Kronecker product of $\boldsymbol{A}$ and $\boldsymbol{B}$*, denoted $\boldsymbol{A} \otimes \boldsymbol{B}$, is the matrix $\boldsymbol{C} \in \mathbb{R}^{I\times K,\,J\times L}$ defined by $c_{(i,k),(j,l)} := a_{i,j} \times b_{k,l}$ for $(i, j, k, l) \in I \times J \times K \times L$. Recall that $\boldsymbol{A} \otimes \boldsymbol{B}$ is invertible if and only if both $\boldsymbol{A}$ and $\boldsymbol{B}$ are. For $n \in \mathbb{N}^+$ and $(p_0, \ldots, p_{n-1}) \in \mathbb{R}^n$, we denote by $\mathcal{V}(p_0, \ldots, p_{n-1})$ the *Vandermonde matrix with coefficients* $(p_0, \ldots, p_{n-1})$, i.e., the matrix in $\mathbb{R}^{[n],[n]}$ whose (i, j)-th entry is p_i^j. Recall that this matrix is invertible if and only if the $p_0, \ldots, p_{n-1}$ are pairwise distinct.

[1] The randomized algorithm from [10] is indeed a ZPP algorithm because the output that it returns (namely, a prospective embedding of a grid as a topological minor of the input graph) can be verified in (deterministic) polynomial time. Hence, we can always detect when the algorithm has failed, and then return the special failure value.

[2] Note that the existence of such an algorithm for k written in unary would be implied by the same claim but with k given in binary. In other words, the existence of an algorithm for k given in unary is a weaker requirement. This is simply because, given an integer in unary, we can convert it in PTIME to an integer in binary.

3 Proof When Every Subdivision Has Length 6

Towards showing our main result (Result 1), we first show in this section a much simpler result: counting the matchings of a graph G reduces to counting the probability of a matching on the graph where each edge is subdivided into a path of length 6. We use similar techniques to previous work, in particular Greenhill [14] and Dalvi and Suciu [12], but present them in detail because we will adapt them in the rest of the paper. Formally, in this section, we show:

▶ **Proposition 3.1.** *For any graph family $\mathcal{F}$, the problem #Matching($\mathcal{F}$) reduces in polynomial time to PrMatching($\mathcal{G}$) where $\mathcal{G} = \{Sub(H, 6) \mid H \in \mathcal{F}\}$.*

Let $H = (V, E)$ be a graph in $\mathcal{F}$ for which we wish to count the number of matchings, with $m := |E|$. Let us start by fixing for the remainder of this section an arbitrary orientation $\overrightarrow{H}$ of H obtained by choosing some orientation of the edges, i.e., $\overrightarrow{H} = (V, \overrightarrow{E})$ is a *directed* graph where for every edge $\{x, y\} \in E$ we add exactly one of (x, y) or (y, x) in $\overrightarrow{E}$. The high-level idea of the reduction is then the following. First, using $\overrightarrow{H}$, we define some sets S_τ, based on 4-tuples $\tau \in [m + 1]^4$, such that the number of matchings of H can be computed from the cardinalities $|S_\tau|$. Second, we argue that these cardinalities can be connected to the results of oracle calls for the PrMatching problem by a system of linear equations. Third, we argue that the matrix of this system can be made invertible. We now detail these three steps.

Step 1: Defining the sets S_τ and linking them to matchings. We define a *selection function* of the graph H as a function μ that maps each vertex $x \in V$ to at most one incident edge, i.e., to a subset of $\mathcal{E}_H(x)$ of size at most one. We will partition the set of selection functions by counting the number of edges of each *type* that each selection function has, as defined next. Given a selection function μ, consider each edge $e = (x, y)$ of $\overrightarrow{H}$. The edge e can have one of four types: letting b be 1 if $\mu(x)$ selects e (i.e., $\mu(x) = \{\{x, y\}\}$) and 0 otherwise (i.e., $\{x, y\} \notin \mu(x)$), and letting b' be 1 if $\mu(y)$ selects e and 0 otherwise, we say that *e has type bb' with respect to (w.r.t.) μ*. We now define the sets S_τ as follows.

▶ **Definition 3.2.** *For a 4-tuple $\tau \in [m + 1]^4$, indexed in binary, let $S_\tau \subseteq S$ be the set of the selection functions μ such that, for all $b, b' \in \{0, 1\}$, precisely $\tau_{bb'}$ edges have type bb' w.r.t. μ.*

Observe that S_τ is empty unless $\tau_{00} + \tau_{01} + \tau_{10} + \tau_{11} = m$. We can then easily connect the cardinalities $|S_\tau|$ to the number of matchings of H as follows (see the full version [4]):

▶ **Fact 3.3.** *We have that #Matching(H) $= \sum_{\substack{\tau \in [m+1]^4 \\ \tau_{01} = \tau_{10} = 0}} |S_\tau|$.*

Step 2: Recovering the $|S_\tau|$ from oracle calls. We now explain how to use the oracle for PrMatching($\mathcal{G}$) to compute in polynomial time all the values $|S_\tau|$, allowing us to conclude via Fact 3.3. We will invoke the oracle on $(m + 1)^4$ probabilistic graphs, denoted $H_6(\kappa)$ for $\kappa \in [m + 1]^4$, as defined next. To this end, let us consider $(m + 1)^4$ 4-tuples of probability values, written $\rho_\kappa = (\rho_{\kappa,00}, \rho_{\kappa,01}, \rho_{\kappa,10}, \rho_{\kappa,11}) \in [0, 1]^4$ for $\kappa \in [m + 1]^4$; the precise choice of these values will be explained in Step 3. For $\kappa \in [m + 1]^4$, we then define $H_6(\kappa)$ to be the probabilistic graph (H_6, π_κ) where $H_6 := \text{Sub}(H, 6)$ is the 6-subdivision of H and the probabilities π_κ are defined as follows. For every directed edge (x, y) of $\overrightarrow{H}$, the subdivision H_6 contains an (undirected) path between x and y, and we define π_κ on this path as follows:

$$x \overset{1/2}{\rule{1cm}{0.4pt}} v_1 \overset{\rho_{\kappa,00}}{\rule{1cm}{0.4pt}} v_2 \overset{\rho_{\kappa,01}}{\rule{1cm}{0.4pt}} v_3 \overset{\rho_{\kappa,10}}{\rule{1cm}{0.4pt}} v_4 \overset{\rho_{\kappa,11}}{\rule{1cm}{0.4pt}} v_5 \overset{1/2}{\rule{1cm}{0.4pt}} y$$

We now introduce some notation for the probability of matchings in paths of length 4. We write $\Pi_4(\rho_\kappa)$ the probability of having a matching in the 4-edge path with successive probabilities $\rho_{\kappa,00}, \rho_{\kappa,01}, \rho_{\kappa,10}, \rho_{\kappa,11}$. The value can be explicitly computed as a polynomial in the values $\rho_{\kappa,bb'}$, e.g., using Equation (1). Accordingly, we will also use Π_4 as a polynomial with real variables, i.e., $\Pi_4(\chi)$ for a 4-tuple χ of real values (which may not be in $[0,1]$). We also define variants of these definitions that account for the two surrounding edges, i.e., those with probability 1/2: for $b, b' \in \{0,1\}$, write $\Pi_4^{bb'}(\rho_\kappa)$ to denote the probability of having a matching in the same 4-edge path but when adding an edge incident to the first vertex with probability 1 if $b = 1$, and adding an edge incident to the last vertex with probability 1 if $b' = 1$. Equivalently, $\Pi_4^{bb'}(\rho_\kappa)$ is the probability of obtaining a matching where we further require if $b = 1$ that the edge with probability $\rho_{\kappa,00}$ is *not* taken, and if $b' = 1$ that the edge with probability $\rho_{\kappa,11}$ is *not* taken. The values $\Pi_4^{bb'}(\rho_\kappa)$ are also explicitly computable as polynomials, and again we also see $\Pi_4^{bb'}$ as a polynomial with real variables. To simplify notation, for $b, b' \in \{0,1\}$ and $\kappa \in [m+1]^4$ let us write $\Lambda_{\kappa,bb'} := \Pi_4^{bb'}(\rho_\kappa)$.

We then show that the probability of a matching in the subdivided graph H_6 can be obtained by first summing over the possible edge type cardinalities τ, and then regrouping the edges of the same type by noticing that the matchings corresponding to the selection functions in the set S_τ all have the same probability. Namely, we show (see the full version [4]):

▶ **Fact 3.4.** *For each $\kappa \in [m+1]^4$, we have:*

$$2^{2m} \times \Pr_{matching}(H_6(\kappa)) \;=\; \sum_{\tau \in [m+1]^4} |S_\tau| \times (\Lambda_{\kappa,00})^{\tau_{00}} \times (\Lambda_{\kappa,01})^{\tau_{01}} \times (\Lambda_{\kappa,10})^{\tau_{10}} \times (\Lambda_{\kappa,11})^{\tau_{11}}.$$

Now, let us write $c_\kappa := \Pr_{\text{matching}}(H_6(\kappa))$ the value returned by the oracle call on $H_6(\kappa)$, and let $\boldsymbol{C}$ be the vector of these oracle answers. Let $\boldsymbol{S}$ be the vector $|S_\tau|$ of the values that we wish to compute. Both these vectors are indexed by $[m+1]^4$. Observe that the equation above defines a system of linear equations $\boldsymbol{V}\boldsymbol{S} = \boldsymbol{C}$ with $\boldsymbol{V} \in \mathbb{R}^{[m+1]^4,[m+1]^4}$ defined by

$$v_{\kappa,\tau} \;:=\; 2^{-2m} \times (\Lambda_{\kappa,00})^{\tau_{00}} \times (\Lambda_{\kappa,01})^{\tau_{01}} \times (\Lambda_{\kappa,10})^{\tau_{10}} \times (\Lambda_{\kappa,11})^{\tau_{11}}.$$

Therefore, if we can choose 4-tuples of probability values ρ_κ that make $\boldsymbol{V}$ invertible, we would be able to recover all $|S_\tau|$ values from the oracle answers $\boldsymbol{C}$, from which we could compute the number of matchings of H using Fact 3.3. This is what we do next.

Step 3: Making $\boldsymbol{V}$ invertible. We now explain how to choose in polynomial time $(m+1)^4$ 4-tuples ρ_κ of rational probability values, for $\kappa \in [m+1]^4$, such that $\boldsymbol{V}$ is invertible. To this end, consider the matrix $\boldsymbol{U}$ defined like $\boldsymbol{V}$ except that each 4-tuple ρ_κ is replaced by a 4-tuple of variables $\chi_\kappa = (\chi_{\kappa,00}, \chi_{\kappa,01}, \chi_{\kappa,10}, \chi_{\kappa,11})$. Each cell $m_{\kappa,\tau}$ of $\boldsymbol{U}$ is then a polynomial P_τ in the 4 variables $\chi_{\kappa,bb'}$ for $b, b' \in \{0,1\}$; in particular, note that the polynomial only depends on the column τ, whereas the variables $\chi_{\kappa,bb'}$ only depend on the row κ. We can then find suitable values ρ_κ using a technique introduced by Dalvi and Suciu [12] (see the full version [4]):

▶ **Proposition 3.5** (From Proposition 8.44 of [12]). *Fix $k \in \mathbb{N}$, let $(x_i)_{i \in I}$ be k-tuples of real variables indexed by a finite set I, let $(P_j)_{j \in J}$ be polynomials in k variables indexed by a finite set J, and consider the matrix $\boldsymbol{M}$ indexed by $I \times J$ such that $m_{i,j} = P_j(x_i)$ for all $(i,j) \in I \times J$. Assume that* $\det(\boldsymbol{M})$ *is not the null polynomial. There is an algorithm that runs in polynomial time in $\boldsymbol{M}$ and finds $|I|$ k-tuples of decimal fractions $(a_i)_{i \in I}$ with values in $[0,1]$ such that the matrix obtained by substituting each x_i by a_i in $\boldsymbol{M}$ is invertible.*

If $\det(\boldsymbol{U})$ is not the null polynomial, we can invoke this result with $k = 4$ and $I = J = [m+1]^4$ on the matrix $\boldsymbol{U}$, which gives us in polynomial time the desired rational probability values ρ_κ (namely, the a_i from the proposition) and concludes the proof of Proposition 3.1.

Hence, the only remaining point is to argue that $\det(\boldsymbol{U})$ is not the null polynomial (in the χ_κ). To this end, let us study the mapping $\xi : \mathbb{R}^4 \to \mathbb{R}^4$, defined as follows, with χ denoting a 4-tuple of real variables: $\xi(\chi) := \big(\Pi_4^{00}(\chi), \Pi_4^{01}(\chi), \Pi_4^{10}(\chi), \Pi_4^{11}(\chi)\big)$. For a 4-tuple of reals ρ, we call the mapping ξ *invertible* around point ρ if there is $\epsilon > 0$ such that the *ϵ-neighborhood around* $\xi(\rho)$, i.e., the set $\{\alpha \in \mathbb{R}^4 \mid |\alpha_{bb'} - \xi(\rho)_{bb'}| \leqslant \epsilon \text{ for each } b, b' \in \{0,1\}\}$, is included in the image of ξ. We conclude by showing two claims:

▶ **Fact 3.6.** *The mapping ξ is invertible around some point.*

Proof. By the inverse function theorem [23], if the *Jacobian determinant* of ξ at a point is not null, then ξ is invertible around that point. Recall that the Jacobian determinant of ξ is the determinant of the *Jacobian matrix* of ξ, which is the 4×4 matrix $\boldsymbol{J}_\xi$ whose entry at cell $((b_1, b_2), (b'_1, b'_2))$ is $\frac{\partial \Lambda_{\chi, b_1 b_2}}{\partial \chi_{b'_1 b'_2}}$. We explicitly compute $\det(\boldsymbol{J})$ with the help of SageMath, showing that it is not the null polynomial (see the full version [4]). ◀

▶ **Fact 3.7.** *If ξ is invertible around some point ρ, then* $\det(\boldsymbol{U})$ *is not the null polynomial.*

Proof. The invertibility of ξ around ρ implies that there exist, for each $b, b' \in \{0,1\}$, a set of $m+1$ distinct values $\Psi_{bb'} := \{\psi_{bb',0}, \ldots, \psi_{bb',m-1}\}$ such that the Cartesian product $\Psi := \bigtimes_{b,b' \in \{0,1\}} \Psi_{bb'}$ is included in the ϵ-neighborhood of $\xi(\rho)$. Let us index the $(m+1)^4$ 4-tuples of Ψ as ψ_κ for $\kappa \in [m+1]^4$, i.e., $\psi_\kappa = (\psi_{00,\kappa_{00}}, \psi_{01,\kappa_{01}}, \psi_{10,\kappa_{10}}, \psi_{11,\kappa_{11}})$. Using invertibility, let α_κ be a preimage of each ψ_κ, i.e., $\xi(\alpha_\kappa) = \psi_\kappa$ for all $\kappa \in [m+1]^4$. But then observe that, for this choice of χ_κ (i.e., substituting the χ_κ by the α_κ), each cell $u_{\kappa,\tau}$ of the matrix $\boldsymbol{U}$ becomes:

$$u_{\kappa,\tau} = 2^{-2m} \times (\psi_{00,\kappa_{00}})^{\tau_{00}} \times (\psi_{01,\kappa_{01}})^{\tau_{01}} \times (\psi_{10,\kappa_{10}})^{\tau_{10}} \times (\psi_{11,\kappa_{11}})^{\tau_{11}}.$$

Thus, $\boldsymbol{U}$ is the Kronecker product of four Vandermonde matrices $\boldsymbol{U}_{bb'}$ for $b, b' \in \{0,1\}$, where $\boldsymbol{U}_{bb'}$ is $\mathcal{V}(\psi_{bb',0}, \ldots, \psi_{bb',m-1})$. As the $\Psi_{bb'}$ consist of pairwise distinct values, these Vandermonde matrices are invertible, and their Kronecker product $\boldsymbol{U}$ also is. ◀

4 Proof When All Subdivisions Have the Same Length $\geqslant 7$

We now prove a variant of Proposition 3.1 where all edges of the initial graph are subdivided the same number of times (at least 7). Given a graph H and integer $K > 0$, we write G_K to mean $\mathrm{Sub}(H, K)$. In this section we show:

▶ **Proposition 4.1.** *Fix an integer $K \geqslant 7$. Then, for any graph family $\mathcal{F}$, the problem #Matching($\mathcal{F}$) reduces in polynomial time to PrMatching($\mathcal{G}$), where $\mathcal{G} = \{H_K \mid H \in \mathcal{F}\}$.*

To prove this, we follow the same strategy as for Proposition 3.1. The first step – the definition of the S_τ – is strictly identical; for m the number of edges of H, we fix again an orientation $\vec{H}$ of H, and denote S_τ for $\tau \in [m+1]^4$ the $(m+1)^4$ sets of selection functions defined from $\vec{H}$ as in Definition 3.2. In particular, Fact 3.3 still holds. Now, we will again construct $(m+1)^4$ probabilistic graphs, denoted $H_K(\kappa)$ for $\kappa \in [m+1]^4$, such that, letting $c_\kappa := \mathrm{Pr}_{\mathrm{matching}}(H_K(\kappa))$, the $|S_\tau|$ and the c_κ form a linear system of equations $\boldsymbol{VS} = \boldsymbol{C}$. We will then again use the Jacobian technique to argue that the determinant of this matrix is not the null polynomial, and complete the proof using Proposition 3.5 to compute in polynomial time rational values that make $\boldsymbol{V}$ have rational entries and be invertible. The difference with Section 3 is in the construction of the probabilistic graphs $H_K(\kappa)$, and in the Jacobian determinant. Before we start, we need to extend the notation from Section 3.

Probabilistic path graphs. For $n \in \mathbb{N}^+$ we denote by P_n the *path of length* n, i.e., $P_n = (\{v_0, \ldots, v_n\}, E)$ where $E = \{\{v_i, v_{i+1}\} \mid 0 \leqslant i \leqslant n-1\}$. For $\rho \in [0,1]^n$, we let $P_n(\rho)$ be the probabilistic graph where each edge $\{v_i, v_{i+1}\}$ of P_n has probability ρ_i. We write $\Pi_n(\rho)$ the probability of a matching in $P_n(\rho)$. For $b, b' \in \{0,1\}$, we write $\Pi_n^{bb'}(\rho)$ to denote $\Pi_{n+2}(b, \rho, b')$, i.e., the probability of a matching in $P_n(\rho)$ where we add an edge to the left if $b = 1$ and add an edge to the right if $b' = 1$. In particular $\Pi_n^{00}(\rho) = \Pi_n(\rho)$. We call the quadruple of values $\Pi_n^{bb'}(\rho)$ for $b, b' \in \{0,1\}$ the *behavior* of the path $P_n(\rho)$. Each $\Pi_n^{bb'}(\rho)$ is a polynomial in the probabilities ρ, and thus we also see $\Pi_n^{bb'}$ as a polynomial with real variables as in Section 3. We will use the following two lemmas. The first one expresses the behavior of the concatenation of two paths as a function of the behavior of each path (see the full version [4]):

▶ **Lemma 4.2.** *Let $n, n' \in \mathbb{N}^+$ and $\rho \in [0,1]^n$, $\rho' \in [0,1]^{n'}$ be tuples of probability values. Then, for every $b, b' \in \{0,1\}$, we have:*

$$\Pi_{n+n'}^{bb'}(\rho, \rho') \;=\; (\Pi_n^{b0}(\rho) \times \Pi_{n'}^{1b'}(\rho')) + (\Pi_n^{b1}(\rho) \times \Pi_{n'}^{0b'}(\rho')) - (\Pi_n^{b1}(\rho) \times \Pi_{n'}^{1b'}(\rho')).$$

The second lemma expresses the values $\Pi_n^{bb'}(1/2, \ldots, 1/2)$ in terms of the *Fibonacci sequence*. Recall that this is the integer sequence defined by $f_0 := 0$, $f_1 := 1$, and $f_n := f_{n-1} + f_{n-2}$ for all $n \in \mathbb{N}^+$, and that this sequence satisfies *Cassini's identity* [22], which says that $f_n^2 = f_{n+1} f_{n-1} + (-1)^{n+1}$ for every $n \in \mathbb{N}^+$. We have (see the full version [4]):

▶ **Lemma 4.3.** *For all $n \in \mathbb{N}^+$, $b, b' \in \{0,1\}$, we have $\Pi_n^{bb'}(1/2, \ldots, 1/2) = \frac{f_{n+2-b-b'}}{2^n}$.*

Proving Proposition 4.1. Let us now build the graphs $H_K(\kappa)$. As before, consider $(m+1)^4$ 4-tuples of probability values $\rho_\kappa = (\rho_{\kappa,00}, \rho_{\kappa,01}, \rho_{\kappa,10}, \rho_{\kappa,11})$ for $\kappa \in [m+1]^4$, to be chosen later. Each graph $H_K(\kappa)$ has H_K as its underlying graph, and for every directed edge $(x, y) \in \overrightarrow{H}$, we set the probabilities on the corresponding undirected path in H_K as follows:

$$x \overset{1/2}{\rule{2em}{0.4pt}} v_1 \overset{\rho_{\kappa,00}}{\rule{2em}{0.4pt}} v_2 \overset{\rho_{\kappa,01}}{\rule{2em}{0.4pt}} v_3 \overset{\rho_{\kappa,10}}{\rule{2em}{0.4pt}} v_4 \overset{\rho_{\kappa,11}}{\rule{2em}{0.4pt}} v_5 \overset{1/2}{\rule{2em}{0.4pt}} v_6 \overset{1/2}{\rule{2em}{0.4pt}} \cdots \overset{1/2}{\rule{2em}{0.4pt}} v_{K-1} \overset{1/2}{\rule{2em}{0.4pt}} y$$

Note that this is like in Section 3, but giving probability $1/2$ to the $N := K - 6$ extra edges on the path. For $b, b' \in \{0,1\}$ we write again $\Lambda_{\kappa,bb'} := \Pi_4^{bb'}(\rho_\kappa)$ the behavior of the 4-path with probabilities ρ_κ, and we define the behavior $\Upsilon_{\kappa,bb'} := \Pi_{K-2}^{bb'}(\rho_\kappa, 1/2, \ldots, 1/2)$ of the path depicted above without the first and last edges. Note that with Lemma 4.2 and Lemma 4.3, we can then express the $\Upsilon_{\kappa,bb'}$ as a function of the $\Lambda_{\kappa,bb'}$ and of the Fibonacci numbers:

▶ **Fact 4.4.** *We have $\Upsilon_{\kappa,bb'} = 2^{-N} \times (\Lambda_{\kappa,b0} \times f_{N+1-b'} + \Lambda_{\kappa,b1} \times f_{N-b'})$ for $b, b' \in \{0,1\}$.*

Studying the graphs $H_K(\kappa)$, by the same reasoning as for Fact 3.4, we can easily show:

$$2^{2m} \times \Pr_{\text{matching}}(H_K(\kappa)) \;=\; \sum_{\tau \in [m+1]^4} |S_\tau| \times (\Upsilon_{\kappa,00})^{\tau_{00}} \times (\Upsilon_{\kappa,01})^{\tau_{01}} \times (\Upsilon_{\kappa,10})^{\tau_{10}} \times (\Upsilon_{\kappa,11})^{\tau_{11}}. \quad (2)$$

This is again a system of linear equations $\boldsymbol{VS} = \boldsymbol{C}$ with $\boldsymbol{V} \in \mathbb{R}^{[m+1]^4, [m+1]^4}$, where $v_{\kappa,\tau} := 2^{-2m} \times (\Upsilon_{\kappa,00})^{\tau_{00}} \times (\Upsilon_{\kappa,01})^{\tau_{01}} \times (\Upsilon_{\kappa,10})^{\tau_{10}} \times (\Upsilon_{\kappa,11})^{\tau_{11}}$. To show that we can compute in polynomial time 4-tuples of rational probability values ρ_κ for $\kappa \in [m+1]^4$ that make $\boldsymbol{V}$ have rational entries and be invertible, we reason as in Section 3. Specifically, we study the Jacobian determinant of the mapping $\xi_N : \chi \mapsto (\Pi_{K-2}^{00}(\chi, 1/2, \ldots, 1/2),\ \Pi_{K-2}^{01}(\chi, 1/2, \ldots, 1/2),$

$\Pi^{10}_{K-2}(\chi, 1/2, \ldots, 1/2)$, $\Pi^{11}_{K-2}(\chi, 1/2, \ldots, 1/2))$, where χ is a 4-tuple of real variables. We show that this determinant is not the null polynomial. To do this, starting from the Jacobian $\boldsymbol{J}_\xi$ of Section 3, using Fact 4.4 and Cassini's identity, and using the fact that the determinant is multilinear and alternating, we obtain (see the full version [4]):

▶ **Fact 4.5.** *We have:* $\det(\boldsymbol{J}_{\xi_N}) = 2^{-4N} \times \det(\boldsymbol{J}_\xi)$.

Hence, $\det(\boldsymbol{J}_{\xi_N})$ is not the null polynomial and, as in Section 3, we can use Proposition 3.5 to complete the proof of Proposition 4.1 (see the full version [4]).

5 Proof for Arbitrary Subdivisions

In this section we finally prove our main result (Result 1), which we re-state here:

▶ **Theorem 5.1.** *Let $\mathcal{G}$ be an arbitrary family of graphs which is treewidth-constructible. Then PrMatching($\mathcal{G}$) is #P-hard under ZPP reductions.*

We will reduce from the problem of counting matchings in 3-regular planar graphs of, which is #P-hard[3] by [26]. Our reduction will be similar to that of Section 4, with the major issue that the various edges of the input graph can now be subdivided to different lengths.

The proof consists of five steps. In step 1, we show a general result allowing us to assign probabilities to a path of length 4 so as to "emulate" the behavior of any long path of *even* length. We then revisit the proof of the previous section. Step 2 extracts the input graph H from the treewidth-constructible family. Step 3 relates the number of matchings of H to cardinalities similar to those of the previous section, but taking the parities of the subdivisions into account. Step 4 then explains how to conclude using emulation. Last, step 5 works around the issue that the probabilities of Step 1 could be irrational, by explaining how we can conclude with sufficiently precise approximations. We now detail these steps.

Step 1: Emulating long even paths. We start by presenting the main technical tool, namely, how to emulate long paths of even length by paths of length 4.

▶ **Proposition 5.2** (Emulation result). *There exist closed-form expressions, denoted $p(i), q(i)$, $r(i), s(i)$, such that for every even integer $i \geqslant 4$ the following hold:*

(A) *the expressions evaluate to well-defined probability values, i.e., we have* $0 \leqslant p(i), q(i), r(i), s(i) \leqslant 1$*; and*

(B) *the path of length 4 with probabilities $p(i), q(i), r(i), s(i)$ behaves like a path of length i with probabilities $1/2$, i.e., $\Pi^{bb'}_4(p(i), q(i), r(i), s(i)) = \Pi^{bb'}_i(1/2, \ldots, 1/2)$ for all $b, b' \in \{0, 1\}$.*

Further, each of these expressions is of the form $\frac{P \pm \sqrt{Q}}{R}$ where P, Q, R are polynomials in the Fibonacci numbers f_{i-1} and f_{i-2} and in 2^{-i}, with rational coefficients.

Proof sketch. The result is simple to state, but we did not find an elegant way to show it. Our proof consists of four steps: (i) rewriting condition (B) into a simpler equivalent system of equations (using Lemma 4.3), (ii) proving that any solution of that system must be in $(0, 1)^4$, (iii) exhibiting closed-form expressions that satisfy the system, found with the help of SageMath; and (iv) verifying that these expressions are well-defined. See the full version [4]. ◀

[3] Note that, in holographic literature, graphs may be multigraphs (i.e., can have multiple edges between two nodes) – see [15]. However, inspecting the proof of [26], we see that the graphs are in fact simple.

▶ Remark 5.3. As $\Pi_i^{bb'}(1/2,\ldots,1/2)$ is symmetric, one would expect the closed-form expressions to satisfy $p(i) = s(i)$ and $q(i) = r(i)$. However, surprisingly, numerical evaluation (already for $i = 6$) shows that our solution does not have this property.

▶ Remark 5.4. It is necessary to require that i is even, as otherwise Proposition 5.2 demonstrably does not hold. In fact, we can prove that, more generally, the behavior of a probabilistic path inherently depends on the parity of its length (see the full version [4]). This is why we will distinguish even-length and odd-length subdivisions in the sequel.

Step 2: Choosing the graph in $\mathcal{G}$. Let $H = (V, E)$ be the input to the reduction, i.e., the 3-regular planar graph for which we want to compute $\#\text{Matching}(H)$, and let $m := |E|$. We first build the graph $H_{10} = \text{Sub}(H, 10)$, writing $H_{10} = (V_{10}, E_{10})$ and we compute $k := |V_{10}|^c$ where c is the constant from Theorem 2.1. Notice that H_{10} is a planar graph of maximum degree 3, and that the size of k in unary is polynomial in (the encoding size of) H. Intuitively, this initial subdivision in 10 will ensure that we have enough room for our probabilistic gadgets. Now, we use the treewidth-constructibility of $\mathcal{G}$ to build in polynomial time a graph $G = (V_G, E_G) \in \mathcal{G}$ such that $\text{tw}(G) \geqslant k$, and using Theorem 2.1 we compute in ZPP a subgraph G' of G with a subdivision $\eta_{10} : E_{10} \to \mathbb{N}^+$ of H_{10} and an isomorphism from $\text{Sub}(H_{10}, \eta_{10})$ to G'. This gives us a subdivision $\eta : E \to \mathbb{N}^+$ of H and an isomorphism f from $\text{Sub}(H, \eta)$ to G', with the initial subdivision ensuring that $\eta(e) \geqslant 10$ for each $e \in E$.

Step 3: Defining the new sets $S_{\tau,\tau'}$ and linking them to matchings. As before, fix an orientation $\overrightarrow{H}$ of H. We call an edge e of H *even* if $\eta(e)$ is even, and *odd* otherwise. For $\tau, \tau' \in [m+1]^4$, both indexed in binary, we define $S_{\tau,\tau'}$ to be the set of selection functions μ of H such that, for $b, b' \in \{0,1\}$, precisely $\tau_{bb'}$ even edges e of H have type bb' w.r.t. μ, and precisely $\tau'_{bb'}$ odd edges e of H have type bb' w.r.t. μ. Then, as in Section 3, we have:

$$\#\text{Matching}(H) = \sum_{\substack{\tau,\tau' \in [m+1]^4 \\ \tau_{01}=\tau_{10}=\tau'_{01}=\tau'_{10}=0}} |S_{\tau,\tau'}|. \tag{3}$$

Step 4: Describing the probabilistic graphs and obtaining the system. To complete the definition of the reduction, let us build the $(m+1)^8$ probabilistic graphs on which we want to invoke the oracle, denoted $G(\kappa, \kappa')$ for $\kappa, \kappa' \in [m+1]^4$. Let $K := \max_{\substack{e \in E \\ \eta(e) \text{ is even}}} (\eta(e))$ and $K' := \max_{\substack{e \in E \\ \eta(e) \text{ is odd}}} (\eta(e))$ and $N := K - 6$ and $N' := K' - 6$. The underlying graph of $G(\kappa, \kappa')$ is G, every edge $e \in E_G$ that is not in G' is assigned probability zero, and we explain next what is the probability associated to the edges that are in G'. Consider $2 \times (m+1)^4$ 4-tuples of probability values $\rho_\kappa = (\rho_{\kappa,00}, \rho_{\kappa,01}, \rho_{\kappa,10}, \rho_{\kappa,11})$ and $\rho'_\kappa = (\rho'_{\kappa',00}, \rho'_{\kappa',01}, \rho'_{\kappa',10}, \rho'_{\kappa',11})$ for $\kappa, \kappa' \in [m+1]^4$, to be chosen later. For every directed edge $(x, y) \in \overrightarrow{H}$, let $\gamma := \eta(\{x, y\})$ be the length to which it is subdivided in G'. Letting $f(x), v_1, \ldots, v_{\gamma-1}, f(y)$ be the corresponding path in G', we set the probabilities of the γ edges along that path as follows:

- If γ is even (illustrated in Figure 1):
  - $1/2, \rho_{\kappa,00}, \rho_{\kappa,01}, \rho_{\kappa,10}, \rho_{\kappa,11}$ for the first 5 edges,
  - $p(N - \gamma + 10), q(N - \gamma + 10), r(N - \gamma + 10), s(N - \gamma + 10)$ for the next four edges,
  - $1/2$ for the remaining $\gamma - 9$ edges.
- If γ is odd:
  - $1/2, \rho'_{\kappa',00}, \rho'_{\kappa',01}, \rho'_{\kappa',10}, \rho'_{\kappa',11}$ for the first 5 edges,
  - $p(N' - \gamma + 10), q(N' - \gamma + 10), r(N' - \gamma + 10), s(N' - \gamma + 10)$ for the next four edges,
  - $1/2$ for the remaining $\gamma - 9$ edges.

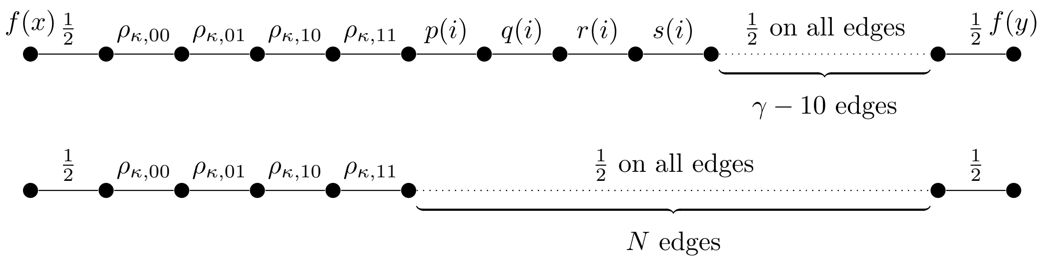


Figure 1 The upper path depicts how we set the probabilities along a path $f(x), v_1, \ldots, v_{\gamma-1}, f(y)$ corresponding to an edge $(x, y) \in \overrightarrow{H}$ such that $\gamma := \eta(\{x, y\})$ is even. We write $i := N - \gamma + 10$. By Lemma 4.2 and Proposition 5.2, this path has exactly the same behavior as the lower path.

We know that $N - \gamma + 10$ (resp., $N' - \gamma + 10$) is an even integer when γ is even (resp., when γ is odd); and it is $\geqslant 4$ by definition of K (resp., of K'). Thus, using Proposition 5.2 and then Lemma 4.2, we know that the path that we defined behaves exactly like the path $P_K(1/2, \rho_{\kappa,00}, \rho_{\kappa,01}, \rho_{\kappa,10}, \rho_{\kappa,11}, 1/2, \ldots, 1/2)$ if γ is even, and exactly like the path $P_{K'}(1/2, \rho'_{\kappa',00}, \rho'_{\kappa',01}, \rho'_{\kappa',10}, \rho'_{\kappa',11}, 1/2, \ldots, 1/2)$ if γ is odd (see again Figure 1).

We have now managed to ensure that all paths for even edges (resp., for odd edges) behave as if they had been subdivided to length K (resp., to length K'). We continue the proof as in the previous section, except that we distinguish odd and even edges. Specifically, for $b, b' \in \{0, 1\}$, we write as in the previous section $\Upsilon_{\kappa,bb'} := \Pi^{bb'}_{K-2}(\rho_\kappa, 1/2, \ldots, 1/2)$ and $\Upsilon'_{\kappa',bb'} := \Pi^{bb'}_{K'-2}(\rho'_{\kappa'}, 1/2, \ldots, 1/2)$. Using the same reasoning as for Equation 2, we obtain:

$$2^{2m} \times \Pr_{\text{matching}}(G(\kappa, \kappa')) = \sum_{\tau,\tau' \in [m+1]^4} |S_{\tau,\tau'}| \times (\Upsilon_{\kappa,00})^{\tau_{00}} \times (\Upsilon_{\kappa,01})^{\tau_{01}} \times (\Upsilon_{\kappa,10})^{\tau_{10}} \times (\Upsilon_{\kappa,11})^{\tau_{11}}$$
$$\times (\Upsilon'_{\kappa',00})^{\tau'_{00}} \times (\Upsilon'_{\kappa',01})^{\tau'_{01}} \times (\Upsilon'_{\kappa',10})^{\tau'_{10}} \times (\Upsilon'_{\kappa',11})^{\tau'_{11}}, \qquad (4)$$

i.e., we obtain a system of linear equations $\mathbf{\Gamma S} = \boldsymbol{C}$ with $\boldsymbol{S}$ the vector of the desired values $|S_{\tau,\tau'}|$, with $\boldsymbol{C}$ the vector of the oracle answers $\Pr_{\text{matching}}(G(\kappa, \kappa'))$, and with $\mathbf{\Gamma} \in \mathbb{R}^{[m+1]^8, [m+1]^8}$, whose entries are given according to the above equation. But notice that we have $\mathbf{\Gamma} = \boldsymbol{V} \otimes \boldsymbol{V}'$, with $v_{\kappa,\tau} := 2^{-m} \times (\Upsilon_{\kappa,00})^{\tau_{00}} \times (\Upsilon_{\kappa,01})^{\tau_{01}} \times (\Upsilon_{\kappa,10})^{\tau_{10}} \times (\Upsilon_{\kappa,11})^{\tau_{11}}$ and $v'_{\kappa',\tau'} := 2^{-m} \times (\Upsilon'_{\kappa',00})^{\tau'_{00}} \times (\Upsilon'_{\kappa',01})^{\tau'_{01}} \times (\Upsilon'_{\kappa',10})^{\tau'_{10}} \times (\Upsilon'_{\kappa',11})^{\tau'_{11}}$. Since $\boldsymbol{V}$ and $\boldsymbol{V}'$ share no variables and are identical up to renaming variables, to argue that there exist 4-tuples of probabilistic values ρ_κ and $\rho'_{\kappa'}$ for $\kappa, \kappa' \in [m+1]^4$ that make $\mathbf{\Gamma}$ invertible, it is enough to know that the Jacobian determinant of the mapping ξ_N is not identically null, as we showed in the previous section (Fact 4.5). Thus, we can again use Proposition 3.5 to compute in polynomial time $2 \times (m+1)^4$ 4-tuples of rational probability values ρ_κ and $\rho'_{\kappa'}$ such that the matrices $\boldsymbol{V}$ and $\boldsymbol{V}'$, hence $\mathbf{\Gamma}$, are invertible (see the full version [4]). By Equation 4, $\mathbf{\Gamma}$ has rational entries, and its inverse $\mathbf{\Gamma}^{-1}$ also does and is computable in polynomial time.

Step 5: Using decimal fractions approximations. The last issue is that we cannot really obtain $\boldsymbol{C}$ via oracle calls, because the graphs $G(\kappa, \kappa')$ may have irrational edge probabilities, namely, the $p(i), q(i), r(i), s(i)$. We now argue that we can still recover the $\boldsymbol{C}$, so that we can compute $\boldsymbol{S} = \mathbf{\Gamma}^{-1}\boldsymbol{C}$ and conclude. To do this, we first observe that $\boldsymbol{C}$ is in fact a vector of decimal fractions, as the graphs $G(\kappa, \kappa')$ emulate a graph where the probabilities are decimal fractions; further, we can bound the number of decimal places of its values to $[m \times (\max(N, N') + 10)] \times z$, with z the maximal number of decimal places of a decimal fraction in $\rho_\kappa, \rho'_\kappa$ Second, we show how to compute decimal fraction *approximations* $\widehat{p(i)}, \widehat{q(i)}, \widehat{r(i)}, \widehat{s(i)}$

of the $p(i), q(i), r(i), s(i)$, in polynomial time in the desired number of places, using the form that they have according to Proposition 5.2. Third, we argue that when invoking the oracles on the graphs where we replace $p(i), q(i), r(i), s(i)$ by $\widehat{p(i)}, \widehat{q(i)}, \widehat{r(i)}, \widehat{s(i)}$, then the error on the answer is bounded as a function of that of the approximations, so that we can recover $\boldsymbol{C}$ exactly if the approximations were sufficiently precise. See the full version [4] for detailed proofs.

6 Result for Edge Covers

Having shown Result 1, we now explain how to adapt its proof to obtain our analogous results for edge covers. We only sketch the argument, and refer to the full version [4] for more details. Recall that an *edge cover* of a graph $G = (V, E)$ is a set of edges $S \subseteq E$ such that $V = \bigcup_{e \in S} e$. Given a probabilistic graph (G, π), we define $\mathrm{Pr}_{\mathrm{edgeCover}}(G, \pi)$ to be the sum of the probabilities of all edge covers in the probability distribution induced by π, and define $\mathrm{PrEdgeCover}(\mathcal{F})$ for a graph family $\mathcal{F}$ to be the corresponding computational problem. We first note that, in this context, the strict analogue of Result 1 does not hold. Indeed, take some treewidth-constructible graph family $\mathcal{G}$, and consider the graph family $\mathcal{G}'$ obtained from $\mathcal{G}$ as follows: for every graph $G \in \mathcal{G}$, we add to $\mathcal{G}'$ the graph that is obtained from G by attaching a dangling edge with a fresh vertex to every node of G. The family $\mathcal{G}'$ is still treewidth-constructible, but $\mathrm{PrEdgeCover}(\mathcal{G}')$ is now tractable as it is easy to see that the edge covers of a graph in $\mathcal{G}'$ are precisely the edge subsets where all dangling edges are kept.

To avoid this, let us assume that $\mathcal{G}$ is closed under taking subgraphs, i.e., if $G \in \mathcal{G}$ and G' is a subgraph of G, then $G' \in \mathcal{G}$. We then have:

▶ **Theorem 6.1.** *Let $\mathcal{G}$ be an arbitrary family of graphs which is treewidth-constructible and closed under taking subgraphs. Then PrEdgeCover($\mathcal{G}$) is #P-hard under ZPP reductions.*

This is proved like Result 1, with the following modifications. We reduce from counting edge covers (instead of matchings) on 3-regular planar graphs: this is hard by [8], even on simple graphs [2, Appendix D]. We now define a selection function μ to map each vertex $x \in V$ to *at least* one incident edge, and we define the types and the sets $S_{\tau,\tau'}$ as before, via an arbitrary orientation of the graph H. We obtain the number of edge covers of H from the quantities $|S_{\tau,\tau'}|$ exactly as in Equation 3. We redefine $\Pi_n^{bb'}(\rho)$ to be the probability of an edge cover in a path of length n with probabilities ρ on the edges and with endpoint constraints given by b, b' as before. Lemma 4.3 then becomes $\Pi_n^{bb'}(1/2, \ldots, 1/2) = \frac{f_{n+b+b'}}{2^n}$, i.e., the role of b, b' is "reversed". Analogous versions of Lemma 4.2 and of Proposition 5.2 still hold, so the relevant Jacobian determinants are still non-identically null. We take the graph $G \in \mathcal{G}$ again via the topological minor extraction result, but this time directly extracting $\mathrm{Sub}(H, \eta) \in \mathcal{G}$ as $\mathcal{G}$ is subgraph-closed. The rest of the proof is identical.

We point out that the situation is different for *perfect* matchings. Indeed, using a weighted variant of the FKT algorithm [7, Chapter 4], the weighted counting of perfect matchings is polynomial-time over the class of planar graphs, which is treewidth-constructible.

We conclude by leaving open two directions for future work. The first one would be to obtain the same kind of lower bounds when the probabilities annotate the *nodes* instead of the edges, that is, studying the corresponding weighted counting problems for, e.g., independent sets, vertex covers, or cliques. We believe that the corresponding result should hold and do not expect any surprises. The second question would be to show our hardness results in the *unweighted* case, e.g., unweighted counting of matchings, assuming that the graph family is subgraph-closed. This appears to be much more challenging, as our current proof crucially relies on the ability to use arbitrary probability values.

References

1 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, pages 355–370, 2016. arXiv:1604.02761.

2 Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Conjunctive queries on probabilistic graphs: Combined complexity. In *PODS*, pages 217–232, 2017. arXiv:1703.03201.

3 Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Connecting width and structure in knowledge compilation. In *ICDT*, volume 98, pages 6:1–6:17, 2018. arXiv:1709.06188.

4 Antoine Amarilli and Mikaël Monet. Weighted counting of matchings in unbounded-treewidth graph families. In *MFCS*, 2022. Full version with proofs. arXiv:2205.00851.

5 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.2544&rep=rep1&type=pdf.

6 Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. Symmetric weighted first-order model counting. In *PODS*, pages 313–328, 2015. arXiv:1412.1505.

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

8 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holographic reduction, interpolation and hardness. *Computational Complexity*, 21(4):573–604, 2012. URL: https://pages.cs.wisc.edu/~jyc/papers/interpolation08.pdf.

9 Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. Complexity of Inference in Graphical Models. In *UAI*, pages 70–78, 2008. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1343&proceeding_id=24.

10 Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *Journal of the ACM*, 63(5):1–65, 2016. arXiv:1305.6577.

11 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1):315–323, 2004. URL: https://core.ac.uk/reader/81145200.

12 Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *Journal of the ACM*, 59(6):30, 2012. URL: https://homes.cs.washington.edu/~suciu/jacm-dichotomy.pdf.

13 Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO1 model-checking. *JCSS*, 1(80):180–194, 2014. arXiv:1109.5804.

14 Catherine Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *computational complexity*, 9(1):52–72, 2000. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.467.3100&rep=rep1&type=pdf.

15 M.Monet (https://cstheory.stackexchange.com/users/38111/m monet). Holant problems and holographic reduction: simple graphs or multigraphs? Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/43912 (version: 2019-05-18).

16 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *LICS*, pages 189–198, 2010. arXiv:1001.5019.

17 Johan Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *ECAI*, pages 237–242, 2010. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=17749.

18 Salil P Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001. URL: https://epubs.siam.org/doi/pdf/10.1137/S0097539797321602.

19 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979. URL: https://core.ac.uk/download/pdf/82500417.pdf.

20 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. URL: https://www.math.cmu.edu/~af1p/Teaching/MCC17/Papers/enumerate.pdf.

21 Wikipedia. Blossom algorithm, 2022. URL: https://en.wikipedia.org/wiki/Blossom_algorithm.

22 Wikipedia. Cassini and Catalan identities, 2022. URL: https://en.wikipedia.org/wiki/Cassini_and_Catalan_identities.

23 Wikipedia. Inverse function theorem, 2022. URL: https://en.wikipedia.org/wiki/Inverse_function_theorem.

24 Wikipedia. Planar graphs, 2022. URL: https://en.wikipedia.org/wiki/Planar_graphs.

25 Wikipedia. Treewidth, 2022. URL: https://en.wikipedia.org/wiki/Treewidth.

26 Mingji Xia, Peng Zhang, and Wenbo Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theor. Comput. Sci.*, 384(1):111–125, 2007. URL: https://core.ac.uk/download/pdf/82063901.pdf.

27 Mingji Xia and Wenbo Zhao. #3-regular bipartite planar vertex cover is #P-complete. In *International Conference on Theory and Applications of Models of Computation*, pages 356–364. Springer, 2006.

On Upward-Planar L-Drawings of Graphs

Patrizio Angelini
John Cabot University, Rome, Italy

Steven Chaplick
Maastricht University, The Netherlands

Sabine Cornelsen
University of Konstanz, Germany

Giordano Da Lozzo
Roma Tre University, Rome, Italy

Abstract

In an *upward-planar L-drawing* of a directed acyclic graph (DAG) each edge e is represented as a polyline composed of a vertical segment with its lowest endpoint at the tail of e and of a horizontal segment ending at the head of e. Distinct edges may overlap, but not cross. Recently, upward-planar L-drawings have been studied for st-graphs, i.e., planar DAGs with a single source s and a single sink t containing an edge directed from s to t. It is known that a *plane st-graph*, i.e., an embedded st-graph in which the edge (s, t) is incident to the outer face, admits an upward-planar L-drawing if and only if it admits a bitonic st-ordering, which can be tested in linear time.

We study upward-planar L-drawings of DAGs that are not necessarily st-graphs. On the combinatorial side, we show that a plane DAG admits an upward-planar L-drawing if and only if it is a subgraph of a plane st-graph admitting a bitonic st-ordering. This allows us to show that not every tree with a fixed bimodal embedding admits an upward-planar L-drawing. Moreover, we prove that any acyclic cactus with a single source (or a single sink) admits an upward-planar L-drawing, which respects a given outerplanar embedding if there are no transitive edges. On the algorithmic side, we consider DAGs with a single source (or a single sink). We give linear-time testing algorithms for these DAGs in two cases: (i) when the drawing must respect a prescribed embedding and (ii) when no restriction is given on the embedding, but the DAG is biconnected and series-parallel.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Graphs and surfaces

Keywords and phrases graph drawing, planar L-drawings, directed graphs, bitonic st-ordering, upward planarity, series-parallel graphs

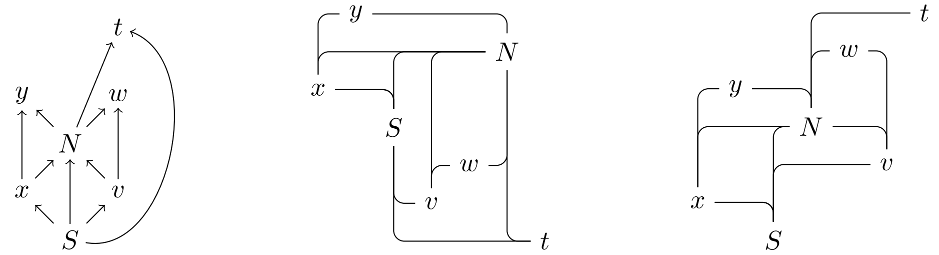
Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.10

Related Version *Full Version*: https://arxiv.org/abs/2205.05627

Funding *Sabine Cornelsen*: Funded by the DFG – Project-ID 50974019 – TRR 161 (B06).
Giordano Da Lozzo: Supported in part by MIUR grant 20174LF3T8 *"AHeAD"*.

1 Introduction

In order to visualize hierarchies, directed acyclic graphs (DAGs) are often drawn in such such a way that the geometric representation of the edges reflects their direction. To this aim *upward drawings* have been introduced, i.e., drawings in which edges are monotonically increasing curves in the y-direction. Sugiyama et al. [21] provided a general framework for drawing DAGs upward. To support readability, it is desirable to avoid edge crossings [20, 23]. However, not every planar DAG admits an *upward-planar drawing*, i.e., an upward drawing in which no two edges intersect except in common endpoints. A necessary condition is that the corresponding embedding is *bimodal*, i.e., all incoming edges are consecutive in the cyclic sequence of edges around any vertex. Di Battista and Tamassia [12] showed that a



(a) DAG G.

(b) Planar L-drawing.

(c) Upward-planar L-drawing of $G - \{S, t\}$.

Figure 1 (a) A single-source series-parallel DAG G. (b) A planar L-drawing of G. (c) An upward-planar L-drawing of G without the edge $\{S, t\}$.

DAG is upward-planar if and only if it is a subgraph of a *planar st-graph*, i.e., a planar DAG with a single source and a single sink that are connected by an edge. Based on this characterization, it can be decided in near-linear time whether a DAG admits an upward-planar drawing respecting a given planar embedding [5, 8]. However, it is NP-complete to decide whether a DAG admits an upward-planar drawing when no fixed embedding is given [16]. For special cases, upward-planarity testing in the variable embedding setting can be performed in polynomial time: e.g. if the DAG has only one source [6, 9, 19], or if the underlying undirected graph is series-parallel [14]. Furthermore, parameterized algorithms for upward-planarity testing exist with respect to the number of sources or the treewidth of the input DAG [11].

Every upward-planar DAG admits a straight-line upward-planar drawing [12], however such a drawing may require exponential area [13]. Gronemann introduced bitonic st-orderings for DAGs [17]. A plane st-graph that admits a bitonic st-ordering also admits an upward-planar drawing in quadratic area. It can be tested in linear time whether a plane st-graph admits a bitonic st-ordering [17], and whether a planar st-graph admits a planar embedding that allows for a bitonic st-ordering [1, 10]. By subdividing some transitive edges once, every plane st-graph can be extended such that it admits a bitonic st-ordering. Moreover, the minimum number of edges that have to be subdivided can be determined in linear time, both, in the variable [1] and the fixed embedding setting [17].

In an *L-drawing* of a directed graph [4] each edge e is represented as a polyline composed of a vertical segment incident to the tail of e and a horizontal segment incident to the head of e. In a *planar L-drawing*, distinct edges may overlap, but not cross. See Fig. 1b for an example. The problem of testing for the existence of a planar L-drawing of a directed graph is NP-complete [10]. On the other hand, every upward-planar DAG admits a planar L-drawing [3]. A planar L-drawing is *upward* if the lowest end vertex of the vertical segment of an edge e is the tail of e (see Fig. 1c). A planar st-graph admits an upward-planar L-drawing if and only it admits a bitonic st-ordering [10].

Our Contribution. We characterize in Theorem 2 the plane DAGs admitting an upward-planar L-drawing as the subgraphs of plane st-graphs admitting a bitonic st-ordering. We first apply this characterization to prove that there are trees with a fixed bimodal embedding that do not admit an upward-planar L-drawing (Theorem 3). Moreover, the characterization allows to test in linear time whether any DAG with a single source or a single sink admits an upward-planar L-drawing preserving a given embedding (Theorem 5).

We further show that every single-source acyclic cactus admits an upward-planar L-drawing by directly computing the x- and y-coordinates as post- and pre-order numbers, respectively, in a DFS-traversal (Theorem 4). The respective result holds for single-sink acyclic cacti. Finally, we use a dynamic-programming approach combined with a matching algorithm for regular expressions to decide in linear time whether a DAG with a single source or a single sink has an embedding admitting an upward-planar L-drawing if it is biconnected and series-parallel (Theorem 7). Observe that a plane st-graph does not necessarily admit an upward-planar L-drawing if the respective graph with reversed edges does. This justifies studying single-source and -sink graphs independently. Full details for proofs of statements marked with ($\star$) can be found in the full version of the paper [2].

2 Preliminaries

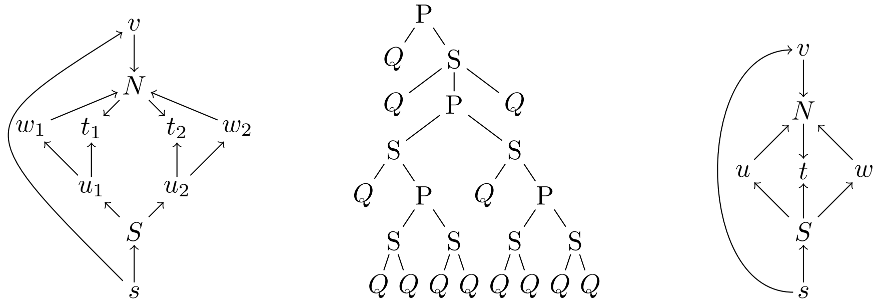
For standard graph theoretic notations and definitions we refer the reader to [22].

Digraphs. A *directed graph (digraph)* $G = (V, E)$ is a pair consisting of a finite set V of *vertices* and a set E of edges containing ordered pairs of distinct vertices. A vertex of a digraph is a *source* if it is only incident to outgoing edges and a *sink* if it is only incident to incoming edges. A *walk* is a sequence of vertices such that any two consecutive vertices in the sequence are adjacent. A *path* is a walk with distinct vertices. In this work we assume that all graphs are *connected*, i.e., that there is always a path between any two vertices. A *cycle* is a walk with distinct vertices except for the first and the last vertex which must be equal. A *directed path* (*directed cycle*) is a path (cycle) where for any vertex v and its successor u in the path (cycle) there is an edge directed from u to v. In the following, we only consider *acyclic digraphs (DAGs)*, i.e., digraphs that do not contain directed cycles. A DAG is a *tree* if it is connected and contains no cycles. It is a *cactus* if it is connected and each edge is contained in at most one cycle.

Drawings. In a *drawing (node-link diagram)* of a digraph vertices are drawn as points in the plane and edges are drawn as simple curves between their end vertices. A drawing of a DAG is *planar* if no two edges intersect except in common endpoints. A planar drawing splits the plane into connected regions – called *faces*. A *planar embedding* of a DAG is the counter-clockwise cyclic order of the edges around each vertex according to a planar drawing. A *plane* DAG is a DAG with a fixed planar embedding and a fixed unbounded face.

The *rotation* of an orthogonal polygonal chain, possibly with overlapping edges, is defined as follows: We start with rotation zero. If the curve bends to the left, i.e., if there is a convex angle to the left of the curve, then we add $\pi/2$ to the rotation. If the curve bends to the right, i.e., if there is a concave angle to the left of the curve, then we subtract $\pi/2$ from the rotation. Moreover, if the curve has a 2π angle to the left, we handle this as two concave angles and if there is a 0 angle to the left, we handle this as two convex angles. The rotation of a simple polygon – with possible overlaps of consecutive edges – traversed in counterclockwise direction is 2π.

Single-source series-parallel DAGs. Series-parallel digraphs are digraphs with two distinguished vertices, called *poles*, and can be defined recursively as follows: A single edge is a series-parallel digraph. Given k series-parallel digraphs $G_1, \ldots, G_k$ (*components*), with poles v_i, u_i, $i = 1, \ldots, k$, a series-parallel digraph G with poles v and u can be obtained in two



(a) DAG not upward-planar. **(b)** Decomposition-tree. **(c)** Embedding not upward-planar.

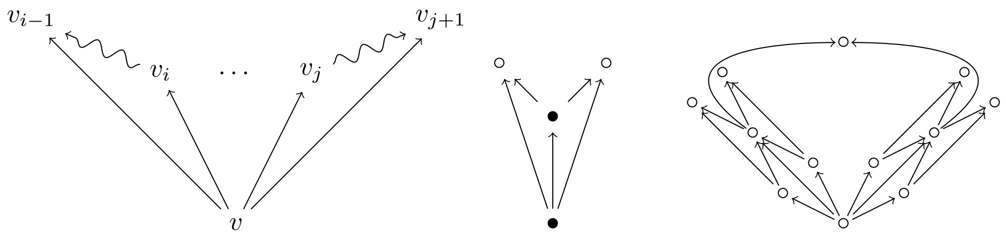
Figure 2 a) A bimodal single-source series-parallel DAG that is not upward-planar b) with its decomposition-tree. c) A single-source series-parallel DAG with an embedding that is not upward-planar. However, the DAG with a different embedding is upward-planar.

ways: by merging $v_1, \ldots, v_k$ and $u_1, \ldots, u_k$, respectively, into the new poles v and u (*parallel composition*), or by merging the vertices u_i and v_{i+1}, $i = 1, \ldots, k-1$, and setting $u = u_1$ and $v = v_k$ (*series composition*). The recursive construction of a series-parallel digraph is represented in a decomposition-tree T. We refer to the vertices of T as *nodes*. The leaves (vertices of degree one) of the decomposition-tree are labeled Q and represent the edges. The other nodes (*inner nodes*) are labeled P for parallel composition or S for series composition. No two adjacent nodes of T have the same label. Fig. 2b shows the decomposition-tree of the graph in Fig. 2a. Let μ be a node of T. We denote by $T(\mu)$ the subtree rooted at μ and by $G(\mu)$ the subgraph of G corresponding to $T(\mu)$, i.e., the subgraph of G formed by the edges corresponding to the leaves of $T(\mu)$. The vertices of $G(\mu)$ that are different from its poles are called *internal.* Given an arbitrary biconnected digraph G, it can be determined in linear time whether it is series-parallel, and a decomposition-tree of G can be computed also in linear time [18]. Moreover, rooting a decomposition-tree of a biconnected series-parallel digraph G at an arbitrary inner node yields again a decomposition-tree of G.

In the following, we assume that G is a series-parallel DAG with a single source (sink) s. If G has more than one edge, we root the decomposition-tree T at the inner node incident to the Q-node corresponding to an edge incident to s. This implies that for any node μ of T no internal vertex of $G(\mu)$ can be a source (sink) of $G(\mu)$ and at least one of the poles of $G(\mu)$ is a source (sink) of $G(\mu)$.

It follows from [6] that every single-source series-parallel DAG is upward-planar if each vertex is incident to at most one incoming or at most one outgoing edge. However, even in that case, not every bimodal embedding is already upward-planar, see Fig. 2c. Moreover, not every single-source series-parallel DAG is upward-planar, even if it admits a bimodal embedding, see Fig. 2a. The reason for that is a P-node μ with two children μ_1 and μ_2 such that a pole N of $G(\mu)$ is incident to an incoming edge in $G - G(\mu)$, and to both incoming and outgoing edges in both, $G(\mu_1)$ and $G(\mu_2)$. Bimodal single-source series-parallel DAGs without this property are always upward-planar [2].

Given an upward-planar drawing of G with distinct y-coordinates for the vertices, we call the pole of $G(\mu)$ with lower y-coordinate the *South pole* of $G(\mu)$ and the other pole the *North pole* of $G(\mu)$. Observe that the South (North) pole of G is the unique source (sink) s. If μ is a P-node with children $\mu_1, \ldots, \mu_\ell$, then the South pole of $G(\mu_i)$, $i = 1, \ldots, \ell$ is the South pole of $G(\mu)$. Finally, if μ is an S-node with children $\mu_1, \ldots, \mu_\ell$, then observe



(a) Valley.

(b) Fixed.

(c) Variable.

Figure 3 (a) Forbidden configuration for bitonic *st*-orderings. (b+c) Single-source series-parallel plane DAG that does not admit an upward-planar L-drawing since it contains a valley, in case (c) in any upward-planar embedding.

that at most one among the components $G(\mu_i)$, $i = 1, \dots, \ell$ can have more than one source (sink) – otherwise G would have more than one source (sink). The South (North) pole of all other components is their unique source (sink).

Bitonic st-ordering. A *planar st-graph* is a planar DAG with a single source s, a single sink t, and an edge (s,t). An *st-ordering* of a planar st-graph is an enumeration π of the vertices with distinct integers, such that $\pi(u) < \pi(v)$ for every edge (u,v). A *plane st-graph* is a planar st-graph with a planar embedding in which the edge (s,t) is incident to the outer face. Every plane st-graph admits an upward-planar drawing [12].

For each vertex v of a plane st-graph, we consider the ordered list $S(v) = \langle v_1, v_2, \dots, v_k \rangle$ of the successors of v as they appear from left to right in an upward-planar drawing. An st-ordering of a plane st-graph is *bitonic*, if there is a vertex v_h in $S(v) = \langle v_1, v_2, \dots, v_k \rangle$ such that $\pi(v_i) < \pi(v_{i+1})$, $i = 1, \dots, h-1$, and $\pi(v_i) > \pi(v_{i+1})$, $i = h, \dots, k-1$. We say that the successor list $S(v) = \langle v_1, v_2, \dots, v_k \rangle$ of a vertex v contains a *valley* if there are $1 < i \leq j < k$ such that there are both, a directed v_i-v_{i-1}-path and a directed v_j-v_{j+1}-path in G. See Fig. 3. Gronemann [17] characterized the plane st-graphs that admit a bitonic st-ordering as follows.

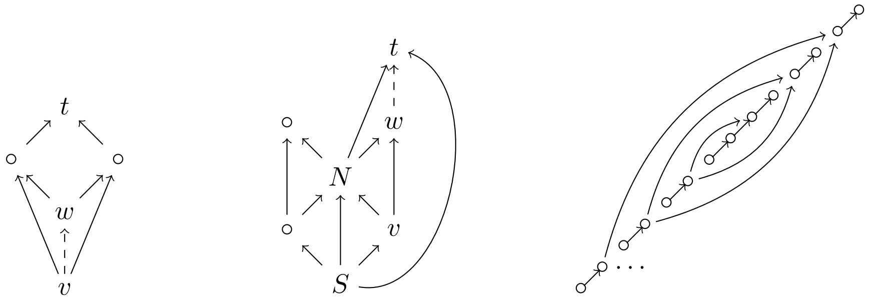
▶ **Theorem 1** ([17]). *A plane st-graph admits a bitonic st-ordering if and only if the successor list of no vertex contains a valley.*

3 Upward-Planar L-Drawings – A Characterization

A plane st-graph admits an upward-planar L-drawing if and only it admits a bitonic st-ordering [10]. We extend this result to general plane DAGs and discuss some consequences.

▶ **Theorem 2.** *A plane DAG admits an upward-planar L-drawing if and only if it can be augmented to a plane st-graph that admits an upward-planar L-drawing, i.e., a plane st-graph that admits a bitonic st-ordering.*

Proof. Let G be a plane DAG. Clearly, if an augmentation of G admits an upward-planar L-drawing, then so does G. Let now an upward-planar drawing of G be given. Add a directed triangle with a new source s, a new sink t, and a new vertex x enclosing the drawing of G. As long as there is a vertex v of G that is not incident to an incoming or outgoing edge, shoot a ray from v to the top or the right, respectively, until it hits another edge and follow the segment to the incident vertex – recall that one end of any segment is a vertex and one end is a bend. The orientation of the added edge is implied by the L-drawing. The result is an upward-planar L-drawing of a digraph with the single source s and the single sink t. ◀



(a) Single sink. **(b)** Single source. **(c)** Family of trees.

Figure 4 DAGs that do not admit an upward-planar L-drawing even though they do not contain a valley. Dashed edges indicate augmentations and are not part of the DAG.

Observe that every series-parallel st-graph admits a bitonic st-ordering [1, 10] and, thus, an upward-planar L-drawing. This is no longer true for upward-planar series-parallel DAGs with several sources or several sinks. Figs. 3b and 3c show examples of two single-source upward-planar series-parallel DAGs that contain a valley. There are even upward-planar series-parallel DAGs with a single source or a single sink that do not admit an upward-planar L-drawing, even though the successor list of no vertex contains a valley.

Consider the DAG G in Fig. 4a (without the dashed edge). G has a unique upward-planar embedding. Since no vertex has more than two successors there cannot be a valley. Assume G admits a planar L-drawing. By Theorem 2 there should be an extension of G to a plane st-graph G' that admits a bitonic st-ordering. But the internal source w can only be eliminated by adding the edge (v, w). Thus w is a successor of v in G'. Hence, the successor list of v in G' contains a valley. By Theorem 1, G' is not bitonic, a contradiction.

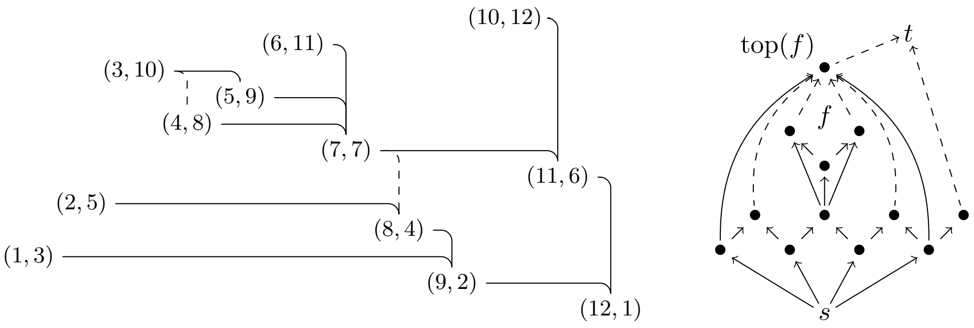
Now consider the DAG G in Fig. 4b (without the dashed edge). G has two symmetric upward-planar embeddings: with the curved edge to the right or the left of the remainder of the DAG. We may assume that the curved edge is to the right. But then an augmentation to a plane st-graph G' must contain the dashed edge, which completes a valley at the single source and its three rightmost outgoing edges. By Theorem 1, G' is not bitonic, a contradiction.

A planar L-drawing is *upward-leftward* [10] if all edges are upward and point to the left.

▶ **Theorem 3** (Trees). *Every directed tree admits an upward-leftward planar L-drawing, but not every tree with a fixed bimodal embedding admits an upward-planar L-drawing.*

Proof. If the embedding is not fixed, we can construct an upward-planar L-drawing of the input tree by removing one leaf v and its incident edge e, drawing the smaller directed tree inductively, and inserting the removed leaf into this upward-leftward planar L-drawing. To this end let u be the unique neighbor of v. We embed e as the first incoming or outgoing edge of u, respectively, in counterclockwise direction, and draw v slightly to the right and below u, if e is an incoming edge of u, or slightly to the left and above u, if e is an outgoing edge of v. This guarantees that the resulting L-drawing is upward-leftward and planar.

When the embedding is fixed, we consider a family of plane trees T_k, $k \geq 1$, proposed by Frati [15, Fig. 4a], that have $2k$ vertices and require an exponential area $\Omega(2^{k/2})$ in any embedding-preserving straight-line upward-planar drawing; see Fig. 4c. We claim that, for sufficiently large k, the tree T_k does not admit an upward-planar L-drawing. Suppose, for a contradiction, that it admits one. By Theorem 2, we can augment this drawing to an upward-planar L-drawing of a plane st-graph G with $n = 2k + 3$ vertices. This implies that G admits a bitonic st-ordering [10]. Hence, G (and thus T_k) admits a straight-line upward-planar drawing in quadratic area $(2n - 2) \times (n - 1)$ [17], a contradiction. ◀



(a) Cactus with post- and pre-order numbers as coordinates. **(b)** Augmentation.

Figure 5 (a) Single-source acyclic cactus. The dashed edges are the last edges on a left path cycles. (b) A new sink t and the dashed edges augment a plane single-source DAG to a plane st-graph.

4 Single-Source or -Sink DAGs with Fixed Embedding

In the fixed embedding scenario, we first prove that every single-source or -sink acyclic cactus with no transitive edge admits an upward-planar L-drawing and then give a linear-time algorithm to test whether a single-source or -sink DAG admits an upward-planar L-drawing.

▶ **Theorem 4** (Plane Single-Source or Single-Sink Cacti). *Every acyclic cactus G with a single source or single sink admits an upward-leftward outerplanar L-drawing. Moreover, if there are no transitive edges (e.g., if G is a tree) then such a drawing can be constructed so to maintain a given outerplanar embedding.*

Proof. We first consider the case that G has a single source s. Observe that then each biconnected component C of G (which is either an edge or a cycle) has a single source, namely the cut-vertex of G that separates it from the part of the DAG containing s. This implies that C also has a single sink (although G may have multiple sinks, belonging to different biconnected components). In particular, if C is a cycle, it consists of a *left* path P_ℓ and a *right* path P_r between its single source and single sink. By flipping the cycle C – maintaining outerplanarity – we can ensure that P_ℓ contains more than one edge. Note that this flipping is only performed if there are transitive edges. Consider the tree T that results from G by removing the last edge of every left path.

We perform a depth-first search on T starting from s where the edges around a vertex are traversed in clockwise order. We enumerate each vertex twice, once when we first meet it (DFS-number or *preorder* number) and once when we finally backtrack from it (*postorder* number). To also obtain that each edge points to the left, backtracking has to be altered from the usual DFS: Before backtracking on a left path P_ℓ of a cycle C, we directly jump to the single source s_C of C and continue the DFS from there, following the right path P_r of C. Only once we have backtracked from the single sink t_C of C, we give each vertex on P_ℓ, excluding s_C, a postorder number and then we continue backtracking on P_r. See Fig. 5a.

Let the y-coordinate of a vertex be its preorder number and let the x-coordinate be its thus constructed postorder number. Since each vertex has a larger preorder- and a lower postorder-number than its parent, the drawing is upward-leftward. In [2] we prove that it is also planar and preserves the embedding, which was updated only in the presence of transitive edges.

Now consider the case that G has a single sink. Flip the embedding, i.e., reverse the linear order of the incoming (outgoing) edges around each vertex. Reverse the orientation of the edges, construct the drawing of the resulting single-source DAG, rotate it by 90 degrees in counter-clockwise direction, and mirror it horizontally. This yields the desired drawing. ◀

General DAGs. Two consecutive incident edges of a vertex form an *angle*. A *large angle* in an upward-planar straight-line drawing is an angle greater than π between two consecutive edges incident to a source or a sink, respectively. An *upward-planar embedding* of an upward-planar DAG is a planar embedding with the assignment of large angles according to a straight-line upward-planar drawing. For single-source or single-sink DAGs, respectively, a planar embedding and a fixed outer face already determine an upward-planar embedding [6].

An angle is a *source-switch* or a *sink-switch*, respectively, if the two edges are both outgoing or both incoming edges of the common end vertex. Observe that the number $A(f)$ of source-switches in a face f equals the number of sink-switches in f. Bertolazzi et al. [5] proved that in biconnected upward-planar DAGs, the number $L(f)$ of large angles in a face f is $A(f) - 1$, if f is an inner face, and $A(f) + 1$, otherwise, and mentioned in the conclusion that this result could be extended to simply connected graphs. An explicit proof for single-source or -sink DAGs can be found in [2].

▶ **Theorem 5.** *Given an upward-plane single-source or single-sink DAG, it can be tested in linear time whether it admits an upward-planar L-drawing.*

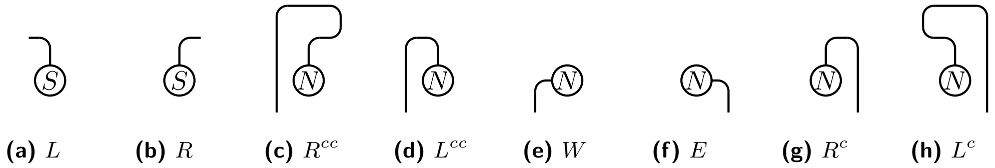
In the following, we prove the theorem for a DAG G with a single source s; the single-sink case is discussed in [2]. In an upward-planar straight-line drawing of G, the only large angle at a source-switch is the angle at s in the outer face. Thus, in the outer face all angles at sink-switches are large and in an inner face f all but one angle at sink-switches are large. For an inner face f, let $\text{top}(f)$ be the sink-switch of f without large angle. See Fig. 5b.

▶ **Lemma 6.** *Let G be a single source upward-planar DAG with a fixed upward-planar embedding, let f be an inner face, and let v be a sink with a large angle in f. Every plane st-graph extending G contains a directed v-$top(f)$-path.*

Proof. Consider a planar st-graph extending G. In this graph there must be an outgoing edge e of v towards the interior of f. Let w be the head of e. Follow a path from w on the boundary of f upward until a sink-switch v' is met. If this sink-switch is $\text{top}(f)$, we are done. Otherwise continue recursively by considering an outgoing edge e' of v' toward the interior of f. Eventually this process terminates when $\text{top}(f)$ is reached. ◀

Proof of Theorem 5, single-source case. Let G be an upward-planar single-source DAG with a fixed upward-planar embedding. Let G' be the DAG that results from G by adding in each inner face f edges from all sinks with a large angle in f to $\text{top}(f)$ and by adding a new sink t together with edges from all sink-switches on the outer face. We will show that G admits an upward-planar L-drawing if and only if G' does. This implies the statement, since testing whether G' admits a bitonic st-ordering can be performed in linear time [17].

Clearly, if $G' \supseteq G$ admits an upward-planar L-drawing, then so does G. In order to prove the other direction, suppose that G has an upward-planar L-drawing. In order to prove that G' admits an upward-planar L-drawing, we show that it is a planar st-graph that admits a bitonic st-ordering [10]. To show this, we argue that G' is acyclic, has a single source and a single sink, and the successor list of no vertex contains a valley by Theorem 1.



(a) L **(b)** R **(c)** R^{cc} **(d)** L^{cc} **(e)** W **(f)** E **(g)** R^{c} **(h)** L^{c}

Figure 6 The different types of a path between the poles. (a,b) South types; (c-h) North types.

Namely, the edges to the new sink t cannot be contained in any directed cycle. Furthermore, by Theorem 2, there is an augmentation G'' of G such that (a) G'' is a planar st-graph and such that (b) G'' admits an upward-planar L-drawing. By Lemma 6, the edges added into inner faces of G either belong to G'' or are transitive edges in G''. Thus, G' is acyclic.

Since G' does not have more sources than G, there is only one source in G'. Each sink has a large angle in some face. Thus, in G' each vertex other than t has at least one outgoing edge. Therefore, G' is a planar st-graph.

Assume now that there is a face f with a sink w such that the edge $(w,\text{top}(f))$ would be part of a valley at a vertex v in G', i.e., assume there are successors $v_{i-1}, v_i, v_j, v_{j+1}$ of v from left to right (with possibly $v_i = v_j$) such that there is both, a directed v_i-v_{i-1}-path and a directed v_j-v_{j+1}-path. Since the out-degree of w in G' is one, it follows that $w \neq v$. Thus, $(w,\text{top}(f))$ could only be part of the v_i-v_{i-1}-path or the v_j-v_{j+1}-path. But then, by Lemma 6, there would be such a path in any augmentation of G to a planar st-graph. Finally, the edges incident to t cannot be involved in any valley, since all the tails have out-degree 1. Thus, G' contains no valleys. ◀

5 Single-Source or -Sink Series-Parallel DAGs with Variable Embedding

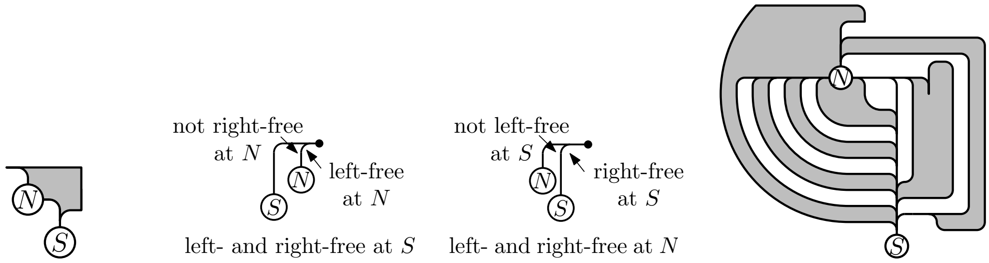
The goal of this section is to prove the following theorem.

▶ **Theorem 7.** *It can be tested in linear time whether a DAG with a single source or a single sink admits an upward-planar L-drawing if it is biconnected and series-parallel.*

In the following we assume that G is a biconnected series-parallel DAG.

Single Source. We follow a dynamic-programming approach inspired by Binucci et al. [7] and Chaplick et al. [11]. We define feasible types that combinatorially describe the "shapes" attainable in an upward-planar L-drawing of each component. We show that these types are sufficient to determine the possible types of a graph obtained with a parallel or series composition, and show how to compute them efficiently. The types depend on the choice of the South pole as the bottommost pole (if it is not uniquely determined by the structure of the graph, e.g., if one of them is the unique source), and on the type of the leftmost S-N-path P_L and the rightmost S-N-path P_R between the South-pole S and the North-pole N. Observe that P_L and P_R do not have to be directed paths.

More precisely, the type of an S-N-path P is defined as follows: There are two *South-types* depending on the edge incident to S: L (outgoing edge bending to the *left*; Fig. 6a) and R (outgoing edge bending to the *right*; Fig. 6b). For the last edge incident to the North pole N we have in addition the types for the incoming edges: W (incoming edge entering from the left – *West*; Fig. 6e) and E (incoming edge entering from the right – *East*; Fig. 6f). For the types R and L we further distinguish whether P passes to the left of N (R^{cc}/L^{cc}; Figs. 6c and 6d) or to the right of N (R^{c}/L^{c}; Figs. 6g and 6h): Let h be the horizontal line



(a) $\langle (E,L),(L^c,R)\rangle$ **(b)** $\langle (R^{cc},R),(R^{cc},R)\rangle$ **(c)** $\langle (R^c,R),(R^c,R)\rangle$ **(d)** Parallel composition.

Figure 7 (a–c) Illustrations for types of a component. (d) A parallel composition of eight components of the following types: $\langle (R^{cc},L),(W,L)\rangle$, $\langle (W,L),(W,L)\rangle$, $\langle (W,L),(W,L)\rangle$, $\langle (W,L),(E,L)\rangle$, $\langle (E,L),(E,L)\rangle$ single edge, $\langle (E,R),(E,R)\rangle$ not left-free at N, $\langle (R^c,R),(R^c,R)\rangle$. The result is of type $\langle (R^{cc},L),(R^c,R)\rangle$.

through N. We say that P *passes to the left (right)* of N if the last edge of P (from S to N) that intersects h does so to the left (right) of N. Thus, there are six *North-types* for a path between the poles: $R^{cc}, L^{cc}, W, E, R^c, L^c$. The superscripts c and cc stand for clockwise and counter-clockwise, respectively, to denote the rotation of a path that passes to the left (right) of N, when walking from N to S. This is justified in the next lemma and depicted e.g., in Fig. 7a, where the right S-N-path has type L^c, since (walking from N to S) it first bends to the left and then passes to the right of N by rotating clockwise.

▶ **Lemma 8** (⋆). *Let G be a series-parallel DAG with no internal sources. Let an upward-planar L-drawing of G be given where the poles S and N are incident to the outer face and S is below N. Let P be a not necessarily directed S-N-path. Let P' be the polygonal chain obtained from P by adding a vertical segment pointing from N downward. The rotation of P' is*

- *π if the type of P at N is in $\{E, L^c, R^c\}$.*
- *$-\pi$ if the type of P at N is in $\{L^{cc}, R^{cc}, W\}$.*

We say that the *type of a path between the poles* is (X, x), if X is the North-type and x is the South-type of the path, e.g., the type of a path that bends right at the South-pole, passes to the right of the North-pole and ends in an edge that leaves the North-pole bending to the left is (L^c, R), see P_R in Fig. 7a. For two North-types X and Y, we say $X < Y$ if X is before Y in the ordering $[R^{cc}L^{cc}WER^cL^c]$. The South-types are ordered $L < R$. For two types (X, x) and (Y, y) we say that $(X, x) \leq (Y, y)$ if $X \leq Y$ and $x \leq y$, and $(X, x) < (Y, y)$ if $(X, x) \leq (Y, y)$ and $X < Y$ or $x < y$.

The *type of a component* is determined by eight entries, whether the component is a single edge or not, the choice of the bottommost pole (South pole), the type of P_L, the type of P_R, and additionally four FREE-*flags*: For each pole, two flags *left-free* and *right-free* indicating whether the bend on P_L and P_R, respectively, on the edge incident to the pole is *free* on the left or the right, respectively: More precisely, let P be an S-N-path and let e be an edge of P incident to a pole X. We say that e is *free* on the right (left) at X if e bends to the right (left) – walking from S to N – or if the bend on e is not contained in an edge not incident to X. See Figs. 7b and 7c. We denote a type by $\langle (X, x), (Y, y)\rangle$ where (X, x) is the type of P_L and (Y, y) is the type of P_R without explicitly mentioning the flags or the choice of the South pole. Observe that $Y < L^c$ if $X = R^{cc}$ and $\langle (X, x), (Y, y)\rangle$ is the type of a component. Fig. 7d illustrates how components of different types can be composed in parallel.

▶ **Lemma 9** (Parallel Composition (⋆)). *A component C of type $\langle (X, x), (Y, y)\rangle$ with the given four* FREE-*flags can be obtained as a parallel composition of components $C_1, \ldots, C_\ell$ of type $\langle (X_1, x_1), (Y_1, y_1)\rangle, \ldots, \langle (X_\ell, x_\ell), (Y_\ell, y_\ell)\rangle$ from left to right at the South pole if and only if*

- $X_1 = X$, $Y_\ell = Y$, $x_1 = x$, $y_\ell = y$,
- *C is left(right)-free at the North- and South-pole, respectively, if and only if C_1 (C_ℓ) is,*
- *$Y_i \le X_i$ and*
  - *C_i is right-free if $Y_i = X_{i+1} \in \{R^{cc}, E, R^c\}$*
  - *C_{i+1} is left-free if $Y_i = X_{i+1} \in \{L^{cc}, W, L^c\}$*
  - *C_i is right-free or C_{i+1} is left-free if $Y_i \in \{L^{cc}, L^c\}$ and $X_{i+1} \in \{R^{cc}, R^c\}$ or vice versa.*
- - *$y_i = x_{i+1} = L$ and C_i is right-free, or*
  - *$y_i = x_{i+1} = R$ and C_{i+1} is left-free, or*
  - *$y_i = L$ and $x_{i+1} = R$ and C_i is right-free or C_{i+1} is left-free.*
- *and single edges are the first among the components with a boundary path of type (W, R) and the last among the components with a boundary path of type (E, L).*

Sketch of Proof. Since the necessity of the conditions is evident, we shortly sketch how to prove sufficiency. By construction, we ensure that the angle between two incoming edges is 0 or π and the angle between an incoming and an outgoing edge is $\pi/2$ or $3\pi/2$. It remains to show the following three conditions [10]: (i) The sum of the angles at a vertex is 2π, (ii) the rotation at any inner face is 2π, (iii) and the *bend-or-end property* is fulfilled, i.e., there is an assignment that assigns each edge to one of its end vertices with the following property. Let e_1 and e_2 be two incident edges that are consecutive in the cyclic order and attached to the same side of the common end vertex v. Let f be the face/angle between e_1 and e_2. Then at least one among e_1 and e_2 is assigned to v and its bend leaves a concave angle in f. ◀

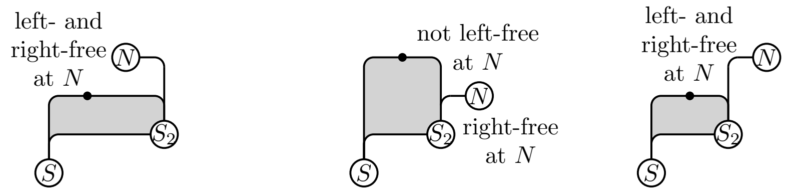
Lemma 9 yields a strict order of the possible types from left to right that can be composed in parallel. Moreover, let σ be a sequence of types of components from left to right that can be composed in parallel and let τ be a type in σ. Then Lemma 9 implies that τ appears exactly once in σ or the leftmost path and the rightmost path have both the same type in τ and all four free-flags are positive. In that case the type τ might occur arbitrarily many times and all appearances are consecutive. Thus, σ can be expressed as a *simple regular expression* on an alphabet $\mathcal{T}$, i.e., a sequence ρ of elements in $\mathcal{T} \cup \{^\star\}$ such that * does not occur as the first symbol of ρ and there are no two consecutive * in ρ. A sequence s of elements in $\mathcal{T}$ is *represented* by a simple regular expression ρ if it can be obtained from ρ by either removing the symbol preceding a * or by repeating it arbitrarily many times.

Observe that the elements in the simple regular expression ρ representing σ are distinct, thus, the length of ρ is linear in the number of types, i.e., constant. In particular, to obtain a linear-time algorithm to enumerate the attainable types of a series-parallel DAG obtained via a parallel composition, it suffices to establish the following algorithmic lemma.

▶ **Lemma 10** (Simple Regular Expression Matching ($\star$))**.** *Let $\mathcal{T}$ be a constant-size alphabet (set of types), and ρ be a constant-length simple regular expression over $\mathcal{T}$. For a collection $\mathcal{C}$ of items where each $C \in \mathcal{C}$ has a set $\mathcal{T}(C) \subseteq \mathcal{T}$, one can test in $\mathcal{O}(|\mathcal{C}|)$ time, if there is a selection of a type from each $\mathcal{T}(C)$, $C \in \mathcal{C}$ that can be ordered to obtain a sequence represented by ρ.*

▶ **Corollary 11.** *The types of a parallel composition can be computed in time linear in the number of its children.*

In order to understand how the type of a series composition is determined from the types of the children, let us first have a look at an easy example: Assume that G_1 and G_2 consist both of a single edge e_1 and e_2, respectively, and that the type of both is (W, R). Assume further that G is obtained by merging the North poles N_1 and N_2 of G_1 and G_2, respectively.



(a) G_1 has type L^{cc}, G_2 type L.

(b) G_1 has type L^{cc}, G_2 type R at S_2.

Figure 8 Different free-flags in the case that the North pole is merged with the South pole.

There are two ways how this can be done, namely e_1 can be attached to $N_1 = N_2$ before e_2 or after it in the counterclockwise order starting from R^{cc} and ending at L^c. In the first case the North type of G is R^{cc}, in the second case it is R^c. Moreover, in the first case G is left-free but not right-free at the North-pole, while in the second case it is right-free but not left-free at the South-pole. See Figs. 7b and 7c.

▶ **Lemma 12** (Series Composition). *Let G_1 and G_2 be two series-parallel DAGs with no internal source that admit an upward-planar L-drawing of a certain type $\langle(X_1, x_1), (Y_1, y_1)\rangle$ and $\langle(X_2, x_2), (Y_2, y_2)\rangle$, respectively, with the poles on the outer face. Let G be the DAG obtained by a series combination of G_1 and G_2 such that the common pole of G_1 and G_2 is not a source in both, G_1 and G_2. Then the possible types of G in an upward-planar L-drawing maintaining the types of G_1 and G_2 can be determined in constant time.*

Proof. Let S_i and N_i, respectively, be the South and North pole of G_i, $i = 1, 2$. We may assume that S_1 is the South pole of G and, thus, N_1 is the common pole of G_1 and G_2.

First suppose that $N_1 = S_2$, i.e., that N_2 is the North pole of G. It follows that N_1 cannot be a source of G_1. Then G admits an upward-planar L-drawing if and only if $x_2 = L$ and $X_1 \neq R^{cc}$ or $y_2 = R$ and $Y_1 \neq L^c$, and the respective FREE-conditions are fulfilled at $N_1 = S_2$. The South-type of G is the South type of G_1. The North-type of G is the North-type of G_2 except for the FREE-flags, which might have to be updated if the next-to-last edge on the leftmost or rightmost path, respectively, is already contained in G_1 and is an outgoing edge of N_1. This might yield different North-types concerning the flags. See Fig. 8.

Now suppose that $N_1 = N_2$. Then G admits an upward-planar L-drawing if and only if G_1 can be embedded before G_2, i.e., $Y_1 \leq X_2$ and $X_1 < Y_2$, and not ($X_1 = R^{cc}$ and $Y_2 = L^c$) or G_2 can be embedded before G_1, i.e., $Y_2 \leq X_1$ and $X_2 < Y_1$, and not ($X_2 = R^{cc}$ and $Y_1 = L^c$) and the respective FREE-conditions are fulfilled at $N_1 = N_2$. If $X_1 = Y_1 = X_2 = Y_2 \in \{E, W\}$ then both conditions are fulfilled which might give rise to two upward-planar L-drawings with distinct labels by adding G_2 before or after G_1 at the common pole. The FREE-flags might have to be updated if the second edge on the leftmost or rightmost path, respectively, is already contained in G_2 or if the next-to-last edge on the leftmost or rightmost path, respectively, is already contained in G_1 and the type of G_1 at N_1 equals the respective type of G_2 at N_2. Except for the flags, the South-type of G is the South type of G_1 and the North type of G_2 yields the North type of G except for the specifications c or cc: First observe that both, the leftmost path and the rightmost path, either have both type c or both type cc. Otherwise, G_1 would be contained in an inner face of G_2. The North type of both paths is indexed c if G_2 is embedded before G_1. Otherwise, the North type is indexed cc. Regarding the time complexity, observe that our computation of the set of possible types of G does not depend on the size of G_1 and G_2, but only on the number of types in their admissible sets. Since these sets have constant size and the above conditions on the types of G_1 and G_2 can be tested in constant time, we thus output the desired set in constant time. ◀

▶ **Lemma 13.** *The types of a series composition can be computed in time linear in the number of its children.*

Proof. Let $C_1, \dots, C_\ell$ be the components of a series component C and let $\mathcal{T}(C_i)$, $i = 1, \dots, \ell$ be the set of possible types of C_i. For $k = 1, \dots, \ell$, we inductively compute the set $\mathcal{T}_i$ of possible types of the series combination C^k of $C_1, \dots, C_k$, where $\mathcal{T}_1 = \mathcal{T}(C_1)$. To compute $\mathcal{T}_k$ for some $k = 2, \dots, \ell$, we combine all possible combinations of a type in $\mathcal{T}_{k-1}$ and a type in $\mathcal{T}(C_k)$ and, applying Lemma 12, we check in constant time which types (if any) they would yield for C^k. Since the number types is constant each step can be done in constant time. ◀

Single Sink. For the case that G has a single sink, the algorithmic principles are the same as in the single-source case. The main difference is the type of an N-S-path P in a component C, where S and N are the South- and North-pole of C. The North pole of a component is always a sink and the North-type of P is W or E in this order from left to right. The South-type is one among $E^c, W^c, L, R, E^{cc}, W^{cc}$ in this order from left to right (according to the outgoing edges at N), depending on whether the last edge of P (traversed from N to S) is an incoming edge entering from the left (W) or the right (E), or an outgoing edge bending to the left (L) or the right (R), and whether the last edge of P leaving the half-space above the horizontal line through S does so to the right of S (cc) or the left of S (c).

The type consists again of the choice of the topmost pole (North pole), the type of the leftmost N-S-path, the type of the rightmost N-S-path, the four FREE-flags – which are defined the same way as in the single source case – and the information whether the component is a single edge or not.

6 Conclusion and Future Work

We have shown how to decide in linear time whether a plane single-source or -sink DAG admits an upward-planar L-drawing. A natural extension of this work would be to consider plane DAGs with multiple sinks and sources, the complexity of which is open. In the variable embedding setting, we have presented a linear-time testing algorithm for single-source or -sink series-parallel DAGs. Some next directions are to consider general single-source or -sink DAGs or general series-parallel DAGs. We remark that the complexity of testing for the existence of upward-planar L-drawings in general also remains open.

References

1 Patrizio Angelini, Michael A. Bekos, Henry Förster, and Martin Gronemann. Bitonic st-orderings for upward planar graphs: The variable embedding setting. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science - 46th International Workshop, WG 2020*, volume 12301 of *LNCS*, pages 339–351. Springer, 2020. doi:10.1007/978-3-030-60440-0_27.

2 Patrizio Angelini, Steven Chaplick, Sabine Cornelsen, and Giordano Da Lozzo. On upward-planar L-drawings of graphs. *CoRR*, abs/2205.05627, 2022. doi:10.48550/arXiv.2205.05627.

3 Patrizio Angelini, Steven Chaplick, Sabine Cornelsen, and Giordano Da Lozzo. Planar L-drawings of bimodal graphs. *Journal of Graph Algorithms and Applications*, 26(3):307–334, 2022. doi:10.7155/jgaa.00596.

4 Patrizio Angelini, Giordano Da Lozzo, Marco Di Bartolomeo, Valentino Di Donato, Maurizio Patrignani, Vincenzo Roselli, and Ioannis G. Tollis. Algorithms and bounds for L-drawings of directed graphs. *Int. J. Found. Comput. Sci.*, 29(4):461–480, 2018. doi:10.1142/S0129054118410010.

5 Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(4):476–497, 1994. doi:10.1007/BF01188716.

6 Paola Bertolazzi, Giuseppe Di Battista, Carlo Mannino, and Roberto Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM Journal on Computing*, 27(1):132–169, 1998. doi:10.1137/S0097539794279626.

7 Carla Binucci, Giordano Da Lozzo, Emilio Di Giacomo, Walter Didimo, Tamara Mchedlidze, and Maurizio Patrignani. Upward book embeddings of st-graphs. In Gill Barequet and Yusu Wang, editors, *35th International Symposium on Computational Geometry, SoCG 2019*, volume 129 of *LIPIcs*, pages 13:1–13:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.13.

8 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM Journal on Computing*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.

9 Guido Brückner, Markus Himmel, and Ignaz Rutter. An SPQR-tree-like embedding representation for upward planarity. In D. Archambault and C. D. Toth, editors, *Graph Drawing and Network Visualization (GD'19)*, volume 11904 of *LNCS*, pages 517–531. Springer, 2016. doi:10.1007/978-3-030-35802-0_39.

10 Steven Chaplick, Markus Chimani, Sabine Cornelsen, Giordano Da Lozzo, Martin Nöllenburg, Maurizio Patrignani, Ioannis G. Tollis, and Alexander Wolff. Planar L-drawings of directed graphs. In Fabrizio Frati and Kwan-Liu Ma, editors, *GD 2017*, volume 10692 of *LNCS*, pages 465–478. Springer, 2018. doi:10.1007/978-3-319-73915-1_36.

11 Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Parameterized algorithms for upward planarity. In *38th International Symposium on Computational Geometry, SoCG 2022*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

12 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.

13 Giuseppe Di Battista, Roberto Tamassia, and Ioannis G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discret. Comput. Geom.*, 7:381–401, 1992. doi:10.1007/BF02187850.

14 Walter Didimo, Francesco Giordano, and Giuseppe Liotta. Upward spirality and upward planarity testing. *SIAM Journal on Discrete Mathematics*, 23(4):1842–1899, 2010. doi:10.1137/070696854.

15 Fabrizio Frati. On minimum area planar upward drawings of directed trees and other families of directed acyclic graphs. *International Journal of Computational Geometry & Applications*, 18(3):251–271, 2008. doi:10.1007/978-3-540-74839-7_13.

16 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.

17 Martin Gronemann. Bitonic *st*-orderings for upward planar graphs. In Yifan Hu and Martin Nöllenburg, editors, *Graph Drawing and Network Visualization (GD'16)*, volume 9801 of *LNCS*, pages 222–235. Springer, 2016. Available at arXiv:1608.08578.

18 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Graph Drawing, 8th International Symposium, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000, Proceedings*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2000. doi:10.1007/3-540-44541-2_8.

19 Michael D. Hutton and Anna Lubiw. Upward planar drawing of single-source acyclic digraphs. *SIAM Journal on Computing*, 25(2):291–311, 1996. doi:10.1137/S0097539792235906.

20 Helen C. Purchase. Metrics for graph drawing aesthetics. *J. Vis. Lang. Comput.*, 13(5):501–516, 2002. doi:10.1006/jvlc.2002.0232.

21 Kozo Sugiyama, Shôjirô Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

22 Roberto Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman & Hall/CRC, 1st edition, 2016.

23 Colin Ware, Helen C. Purchase, Linda Colpoys, and Matthew McGill. Cognitive measurements of graph aesthetics. *Inf. Vis.*, 1(2):103–110, 2002. doi:10.1057/palgrave.ivs.9500013.

RAC Drawings of Graphs with Low Degree

Patrizio Angelini
John Cabot University, Rome, Italy

Michael A. Bekos
Department of Mathematics, University of Ioannina, Ioannina, Greece

Julia Katheder
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Tübingen, Germany

Michael Kaufmann
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Tübingen, Germany

Maximilian Pfister
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Tübingen, Germany

Abstract

Motivated by cognitive experiments providing evidence that large crossing-angles do not impair the readability of a graph drawing, RAC (Right Angle Crossing) drawings were introduced to address the problem of producing readable representations of non-planar graphs by supporting the optimal case in which all crossings form 90° angles.

In this work, we make progress on the problem of finding RAC drawings of graphs of low degree. In this context, a long-standing open question asks whether all degree-3 graphs admit straight-line RAC drawings. This question has been positively answered for the Hamiltonian degree-3 graphs. We improve on this result by extending to the class of 3-edge-colorable degree-3 graphs. When each edge is allowed to have one bend, we prove that degree-4 graphs admit such RAC drawings, a result which was previously known only for degree-3 graphs. Finally, we show that 7-edge-colorable degree-7 graphs admit RAC drawings with two bends per edge. This improves over the previous result on degree-6 graphs.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Graph algorithms

Keywords and phrases Graph Drawing, RAC graphs, Straight-line and bent drawings

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.11

Related Version *Full Version*: `https://arxiv.org/abs/2206.14909`

1 Introduction

In the literature, there is a wealth of approaches to draw planar graphs. Early results date back to Fáry's theorem [22], which guarantees the existence of a planar straight-line drawing for every planar graph; see also [9, 30, 31, 33, 34]. Over the years, several breakthrough results have been proposed, e.g., de Fraysseix, Pach and Pollack [11] in the late 80's devised a linear-time algorithm [10] that additionally guarantees the obtained drawings to be on an integer grid of quadratic size (thus making high-precision arithmetics of previous approaches unnecessary). Planar graph drawings have also been extensively studied in the presence of bends. Here, a fundamental result is by Tamassia [32] in the context of *orthogonal* graph drawings, i.e., drawings in which edges are axis-aligned polylines. In his seminal paper, Tamassia suggested an approach, called *topology-shape-metrics*, to minimize the number of bends of degree-4 plane graphs using flows. For a complete introduction, see [12].

When the input graph is non-planar, however, the available approaches that yield aesthetically pleasing drawings are significantly fewer. The main obstacle here is that the presence of edge-crossings negatively affects the drawing's quality [28] and, on the other hand,

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 11; pp. 11:1–11:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

their minimization turns out to be a computationally difficult problem [23]. In an attempt to overcome these issues, a decade ago, Huang et al. [26] made a crucial observation that gave rise to a new line of research (currently recognized under the term "beyond planarity" [25]): edge crossings do not negatively affect the quality of the drawing too much (and hence the human's ability to read and interpret it), if the angles formed at the crossing points are large. Thus, the focus moved naturally to non-planar graphs and their properties, when different restrictions on the type of edge-crossings are imposed; see [19] for an overview.

Among the many different classes of graphs studied as part of this emerging line of research, one of the most studied ones is the class of *right-angle-crossing graphs* (or *RAC graphs*, for short); see [14] for a survey. These graphs were introduced by Didimo, Eades and Liotta [15, 17] back in 2009 as those admitting straight-line drawings in which the angles formed at the crossings are all 90°. Most notably, these graphs are optimal in terms of the crossing angles, which makes them more readable according to the observation by Huang et al. [26]; moreover, RAC drawings form a natural generalization of orthogonal graph drawings [32], as any crossing between two axis-aligned polylines trivially yields 90° angles.

In the same work [15, 17], Didimo, Eades and Liotta proved that every n-vertex RAC graph is sparse, as it can contain at most $4n - 10$ edges, while in a follow-up work [16] they observed that not all degree-4 graphs are RAC. This gives rise to the following question which has also been independently posed in several subsequent works (see e.g., [3], [18, Problem 6], [14, Problem 9.5], [19, Problem 8]) and arguably forms the most intriguing open problem in the area, as it remains unanswered since more than one decade.

▶ **Question 1.** *Does every graph with degree at most 3 admit a straight-line RAC drawing?*

The most relevant result that is known stems from the related problem of simultaneously embedding two or more graphs on the Euclidean plane, such that the crossings between different graphs form 90° angles. In this setting, Argyriou et al. [4] showed that a cycle and a matching always admit such an embedding, which implies that every *Hamiltonian* degree-3 graph is RAC.

Finally, note that recognizing RAC graphs is hard in the existential theory of the reals [29], which also implies that RAC drawings may require double-exponential area, in contrast to the quadratic area requirement for planar graphs [11].

RAC graphs have also been studied by relaxing the requirement that the edges are straight-line segments, giving rise to the class of k*-bend* RAC graphs (see, e.g, [1, 6, 7, 8, 13]), i.e., those admitting drawings with at most k bends per edge and crossings at 90° angles. It is known that every degree-3 graph is 1-bend RAC and every degree-6 graph is 2-bend RAC [3]. While the flexibility guaranteed by the presence of one or two bends on each edge is not enough to obtain a RAC drawing for every graph (in fact, 1- and 2-bend RAC graphs with n vertices have at most $5.5n - O(1)$ and $72n - O(1)$ edges, respectively [1, 6]), it is known that every graph is 3-bend RAC [13] and fits on a grid of cubic size [21].

Our contribution. We provide several improvements to the state of the art concerning RAC graphs with low degree. In particular, we make an important step towards answering Question 1 by proving that 3-edge-colorable degree-3 graphs are RAC (Theorem 3). This result applies to Hamiltonian 3-regular graphs, to bipartite 3-regular graphs and, with some minor modifications to our approach, to all Hamiltonian degree-3 graphs, thus extending the result in [4]. As a further step towards answering Question 1, we prove that bridgeless 3-regular graphs with oddness at most 2 are RAC (Theorem 8). If their oddness is k, we provide an algorithm to construct a 1-bend RAC drawing where at most k edges have a bend(Theorem 9).

We then focus on RAC drawings with one or two bends per edge. Namely, we prove that all degree-4 graphs admit 1-bend RAC drawings and all 7-edge-colorable degree-7 graphs admit 2-bend RAC drawings (Theorems 10 and 12), which form non-trivial improvements over the state of the art, as the existence of such drawings was previously known only for degree-3 and degree-6 graphs [3]. Due to space constraints, proofs of statements marked with $(*)$ can be found in [2].

2 Preliminaries

Let $G = (V, E)$ be a graph. W.l.o.g. we assume that G is connected, as otherwise we apply our drawing algorithms to each component of G separately. G is called *degree-k* if the maximum degree of G is k. G is called *k-regular* if the degree of each vertex of G is exactly k. A 2-factor of an undirected graph $G = (V, E)$ is a spanning subgraph of G consisting of vertex disjoint cycles. Let F be a 2-factor of G and let $\prec$ be a total order of the vertices such that the vertices of each cycle $C \in F$ appear consecutive in $\prec$ according to some traversal of C. In other words, every two vertices that are adjacent in C are consecutive in $\prec$ except for two particular vertices, which are the first and the last vertices of C in $\prec$. We call the edge between these two vertices the *closing edge* of C. By definition, $\prec$ also induces a total order of the cycles of F. Let $\{u, v\}$ be an edge in $E \setminus F$ and let C and C' be the cycles of F that contain u and v, respectively. If $C = C'$, then $\{u, v\}$ is a *chord* of C. Otherwise, $C \neq C'$. If $u \prec v$, $\{u, v\}$ is called a *forward edge* of u and a *backward edge* of v. The following theorem provides a tool to partition the edges of a bounded degree graph into 2-factors [27].

▶ **Theorem 2** (Eades, Symvonis, Whitesides [20]). *Let $G = (V, E)$ be an n-vertex undirected graph of degree Δ and let $d = \lceil \Delta/2 \rceil$. Then, there exists a directed multi-graph $G' = (V, E')$ with the following properties:*

1. *each vertex of G' has indegree d and outdegree d;*
2. *G is a subgraph of the underlying undirected graph of G'; and*
3. *the edges of G' can be partitioned into d edge-disjoint directed 2-factors.*

Furthermore, the directed graph G' and the d 2-factors can be computed in $\mathcal{O}(\Delta^2 n)$ time.

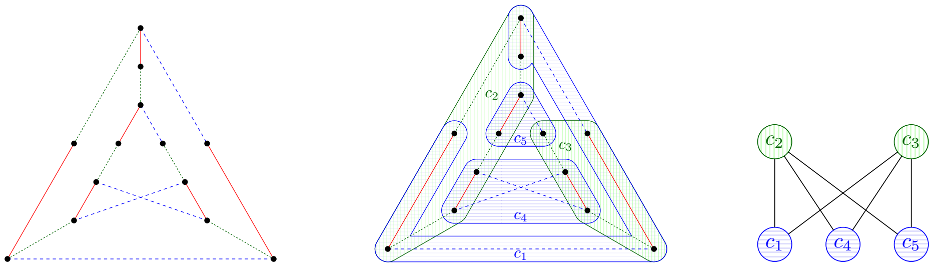
Let Γ be a polyline drawing of G such that the vertices and edge-bends lie on grid points. The area of Γ is determined by the smallest enclosing rectangle. Let $\{u, v\}$ be an edge in Γ. We say that $\{u, v\}$ is using an *orthogonal port* at u if the edge-segment s_u of $\{u, v\}$ that is incident to u is either horizontal or vertical; otherwise it is using an *oblique port* at u. We denote the orthogonal ports at u by N, E, S and W, if s_u is above, to the right, below or to the left of u, respectively. If no edge is using a specific orthogonal port, we say that this port is *free*.

3 RAC drawings of 3-edge-colorable degree-3 graphs

In this section, we prove that 3-edge-colorable degree-3 graphs admit RAC drawings of quadratic area, which can be computed in linear time assuming that the edge coloring is given (testing the existence of such a coloring is NP-complete even for 3-regular graphs [24]).

▶ **Theorem 3.** *Given a 3-edge-colorable degree-3 graph G with n vertices and a 3-edge-coloring of G, it is possible to compute in $O(n)$ time a RAC drawing of G with $O(n^2)$ area.*

We assume w.l.o.g. that G does not contain degree-1 vertices, as otherwise we can replace each such vertex with a 3-cycle while maintaining the 3-edge-colorability of the graph and without asymptotically increasing the size of the graph. Since G is 3-edge-colorable, it can be



(a) G **(b)** $c_1, \dots, c_5$ **(c)** $\mathcal{H}$

Figure 1 (a) A non-planar, non-Hamiltonian, 3-edge-colorable degree-3 example graph G. The matching M_1 is drawn with blue dashed lines, M_2 with red solid lines and M_3 with green dotted lines. (b) The components c_1, c_2 and c_3 of the subgraph H_y induced by $M_1 \cup M_2$ (shaded in blue) and the components c_4 and c_5 of H_x induced by $M_2 \cup M_3$ (shaded in green). (c) The auxiliary graph $\mathcal{H}$ in which the components of H_y and H_x sharing at least one vertex are connected by an edge. In this example, the BFS traversal of the components of $\mathcal{H}$ is c_1, c_2, c_3, c_4, c_5.

decomposed into three matchings M_1, M_2 and M_3. In the produced RAC drawing, the edges in M_1 will be drawn horizontal, those in M_3 vertical, while those in M_2 will be crossing-free, not maintaining a particular slope. Let H_y and H_x be two subgraphs of G induced by $M_1 \cup M_2$ and $M_2 \cup M_3$, respectively. Since every vertex of G has at least two incident edges, which belong to different matchings, each of H_y and H_x spans all vertices of G. Further, any connected component in H_y or H_x is either a path or an even-length cycle, as both H_y and H_x are degree-2 graphs alternating between edges of different matchings.

We define an auxiliary bipartite graph $\mathcal{H}$, whose first (second) part has a vertex for each connected component in H_y (H_x), and there is an edge between two vertices if and only if the corresponding components share at least one vertex; see Fig. 1.

▶ **Property 4.** *The auxiliary graph $\mathcal{H}$ is connected.*

Proof. Suppose for a contradiction that $\mathcal{H}$ is not connected. Let v_c and $v_{c'}$ be two vertices of $\mathcal{H}$ that are in different connected components of $\mathcal{H}$. By definition of $\mathcal{H}$, v_c and $v_{c'}$ correspond to connected components c and c', respectively, of H_y or H_x. W.l.o.g. assume that c belongs to H_y. Let u_0 and u_k be two vertices of G that belong to c and c', respectively. Since G is connected, there is a path $P = (u_0, u_1, u_2, \dots u_k)$ between u_0 and u_k in G, such that no two consecutive edges in P belong to the same matching. Let (u_i, u_{i+1}) be the first edge of P from u_0 to u_k that belongs to M_3, which exists since $c \neq c'$. By construction, this implies that u_i belongs to c and to another component c^* of H_x. By definition of $\mathcal{H}$, v_c and v_{c^*} are connected in $\mathcal{H}$, where v_{c^*} is the corresponding vertex of c^* in $\mathcal{H}$. Repeating this argument until u_k is reached yields a path in $\mathcal{H}$ from v_c to $v_{c'}$, a contradiction. ◀

We now define two total orders $\prec_y$ and $\prec_x$ of the vertices of G, which will then be used to assign their y- and x-coordinates, respectively, in the final RAC drawing of G. Since we seek to draw the edges of M_1 (M_3) horizontal (vertical), we require that the endvertices of any edge in M_1 (M_3) are consecutive in $\prec_y$ ($\prec_x$, respectively). Moreover, for the edges of M_2, we guarantee some properties that will allow us to draw them without crossings.

To construct $\prec_y$ and $\prec_x$, we process the components of H_y and H_x according to a certain BFS traversal of $\mathcal{H}$ and for each visited component of H_y (H_x), we append all its vertices to $\prec_y$ ($\prec_x$) in a certain order.

To select the first vertex of the BFS traversal of H, we consider a vertex u of G belonging to two components c and c' of H_y and H_x, respectively, such that u is the endpoint of c if c is a path; if c is a cycle, we do not impose any constraints on the choice of u. We refer to vertex u as the *origin vertex* of G. Also, let v_c and $v_{c'}$ be the vertices of $\mathcal{H}$ corresponding to c and c', respectively. By definition of $\mathcal{H}$, v_c and $v_{c'}$ are adjacent in $\mathcal{H}$. We start our BFS traversal of $\mathcal{H}$ at v_c and then we move to $v_{c'}$ in the second step (note that this choice is not needed for the definition of $\prec_y$ and $\prec_x$, but it guarantees a structural property that will be useful later). From this point on, we continue the BFS traversal to the remaining vertices of $\mathcal{H}$ without further restrictions. In the following, we describe how to process the components of H_y and H_x in order to guarantee an important property (see Property 5)

Let c be the component of H_y or H_x corresponding to the currently visited vertex in the traversal of $\mathcal{H}$. Since $\mathcal{H}$ is bipartite, no other component of H_y (H_x) shares a vertex with c, if c belongs to H_y (H_x). Hence, no vertex of c already appears in $\prec_y$ ($\prec_x$).

If c is a path, then we append the vertices of c to $\prec_y$ or $\prec_x$ in the natural order defined by a walk from one of its endvertices to the other. Note that if c is the first component in the BFS traversal of $\mathcal{H}$, one of these endvertices is by definition the origin vertex of G, which we choose to start the walk. Hence, in the following we focus on the case that c is a cycle. In this case, the vertices of c will also be appended to $\prec_y$ or $\prec_x$ in the natural order defined by some specific walk of c, such that the so-called *closing edge* connecting the first and the last vertex of c in this order belongs to M_2. Note that an edge might be closing in both orders $\prec_y$ and $\prec_x$.

Suppose first that $c \in H_y$. If c is the first component in the BFS traversal of $\mathcal{H}$, then we append the vertices of c to $\prec_y$ in the order that they appear in the cyclic walk of c starting from the origin vertex of G and following the edge of M_1 incident to it. Otherwise, let v be the first vertex of c in $\prec_x$, which is well defined since there is at least one vertex of c that is part of $\prec_x$, namely, the one that is shared with its parent. We append the vertices of c to $\prec_y$ in the order that they appear in the cyclic walk of c starting from v and following the edge of M_1 incident to v. Hence, v is the first vertex of c in both $\prec_y$ and $\prec_x$. In both cases, it follows that the closing edge of c belongs to M_2.

Suppose now that $c \in H_x$, which implies that c is not the first component in the BFS traversal of $\mathcal{H}$. Let v be the first vertex of c in $\prec_y$, which is again well defined since there is at least one vertex of c that is part of $\prec_y$. We append the vertices of c to $\prec_x$ in the inverse order that they appear in the cyclic walk of c starting from v and following the edge (v, w) of M_3 incident to v (or equivalently, in the order they appear in the cyclic walk of c starting from the neighbor of v different from w and ending at v). Hence, v is the first vertex of c in $\prec_y$ and the last vertex of c in $\prec_x$. Also in this case, the closing edge of c belongs to M_2. See Fig. 2 for an illustration. Note that the closing edge of a component c is contained inside the parent component of c is the BFS traversal. Moreover, by construction, the following property holds.

▶ **Property 5.** *The endvertices of any edge in M_1 (M_3) are consecutive in $\prec_y$ ($\prec_x$). The endvertices of any edge in M_2 are consecutive in $\prec_y$ ($\prec_x$) unless this edge is a closing edge in a component of H_y (of H_x).*

Computing the vertex coordinates. We use $\prec_y$ and $\prec_x$ to specify the y- and the x-coordinates of the vertices, respectively. To do so, we iterate through $\prec_y$ and set the y-coordinate of its first vertex to 1. Let v be the next vertex in the iteration and let u be its predecessor in $\prec_y$. Assume that the y-coordinate of u is i. If $(u, v) \in M_1$, we set the same y-coordinate i to v. Otherwise, either u and v belong to two different components of H_y or $(u, v) \in M_2$ and we set the y-coordinate $i + 1$ to v. Similarly, we iterate through $\prec_x$ and set the x-coordinate of its first vertex to 1. Let v be the next vertex in the iteration and let u be its predecessor in $\prec_x$. Assume that the x-coordinate of u is i. If $(u, v) \in M_3$, we set

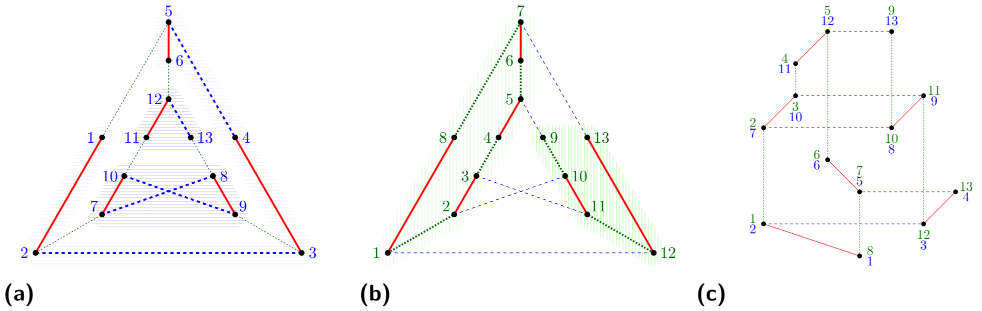


Figure 2 The total orders (a) $\prec_y$ for H_y that consists of the blue and red edges, and (b) $\prec_x$ for H_x that consists of the green and red edges. The final drawing of G is shown in (c).

the x-coordinate of v to i. Otherwise, either u and v belong to two different components of H_x or $(u, v) \in M_2$ and we set the x-coordinate $i + 1$ to v. Hence, no two vertices share the same x- and y-coordinates. We next show that the computed vertex coordinates induce a straight-line RAC drawing Γ of G with the possible exception of the edge of M_2 incident to the origin vertex of G, since this edge would be a closing edge for both c and c' and hence by Property 5 its endpoints would be consecutive in neither $\prec_y$ nor $\prec_x$. If this edge exists, we denote it by e^*, while the graph $G \setminus e^*$ and its drawing in Γ are denoted by G^* and Γ^*, respectively.

▶ **Lemma 6.** *Let e be an edge of G^*. Then, e is drawn horizontally in Γ^* if $e \in M_1$; vertically in Γ^* if $e \in M_3$ and crossing-free in Γ^* if $e \in M_2$.*

Proof. If $e \in M_1$ or $e \in M_3$, the statement follows from Property 5 and the computed vertex coordinates. Hence, let $e = (u, v)$ be an edge of M_2 and let $c_y \in H_y$ and $c_x \in H_x$ be the two components of $\mathcal{H}$ containing e. Suppose to the contrary that there is an edge $e' = (u', v')$ crossing e. If $e' \in M_1$, then both u' and v' belong to the same component $c'_y \in H_y$. If $c'_y \neq c_y$, then e' cannot cross e as the vertices of c_y and c'_y span different intervals of y-coordinates, hence e' belongs to c_y. Similarly, if $e' \in M_3$, then e' belongs to c_x. Finally, if $e' \in M_2$, then u' and v' belong to the same component in both H_y and H_x; thus e' belongs to both c_y and c_x.

1. Edge e is a closing edge for neither c_y nor c_x: The vertices u and v are consecutive in both $\prec_y$ and $\prec_x$ by Property 5. Hence, both their x- and y-coordinate differ by exactly one by construction. Since all vertices have integer coordinates, no horizontal or vertical edge is crossing e, hence $e' \in M_2$. Observe that since the y- (the x-) coordinate of the vertices in c_y (in c_x) are non-decreasing along the walk defining its order in $\prec_y$ (in $\prec_x$), no crossing between e and e' can occur if e' is not a closing edge of c_y or c_x, which will be covered in the next cases (by swapping the roles of e and e').
2. Edge e is a closing edge for c_y but not for c_x: Note that c_y is not the first component in the BFS traversal, since the closing edge of this component is e^*. Further, u and v are consecutive in $\prec_x$, but not in $\prec_y$. We assume that u directly precedes v in $\prec_x$, which implies that u is the first vertex of c_y in $\prec_y$, while v is the last. It follows that their x-coordinates differ by exactly one, hence e' cannot belong to M_3. If e' belongs to M_1, then one of u' or v', say u', has x-coordinate smaller or equal to the one of u. Since u and v are consecutive in $\prec_x$, we have necessarily that u' precedes u in $\prec_x$, which is a contradiction to the choice of u since both u' and v' belong to c_y. In fact, u was chosen as the starting point of the walk, when considering c_y, as the first vertex of c_y in $\prec_x$, hence $e' \in M_2$. Since both endpoints of e' belong to both c_y and c_x, then one of u' or v',

say u', has x-coordinate smaller or equal to the one of u. Since u and v are consecutive in $\prec_x$, we have necessarily that $u' \prec_x u$, which is a contradiction to the choice of u since both u' and v' belong to c_y.

3. Edge e is a closing edge for c_x but not for c_y: This case is analogous to the previous one.
4. Edge e is a closing edge for both c_y and c_x: Observe that neither c_y nor c_x is the first component in the BFS traversal of $\mathcal{H}$, since e^* is the closing edge of this component, which is not part of G^*. Recall that by definition, the vertices v_{c_y} and v_{c_x} corresponding to c_y and c_x in $\mathcal{H}$ are adjacent. Assume that c_y is visited before c_x in the BFS traversal; the other case is symmetric. By our construction rule, when considering c_y, we started the walk from the vertex u that is the first vertex of c_y in $\prec_x$, which means u also belongs to a component c'_x of H_x. Since c_y is a cycle, u is incident to an edge in M_2, which then also belongs to c'_x. Clearly, the edge of M_2 incident to u is the closing edge of c_y and contained in c'_x, which implies that the edge does not belong to c_x, hence this case does not occur in G^*. ◀

By the last case of Lemma 6, it follows that if the edge e^* exists, then it is the only closing edge of two components, which is summarized in the following corollary.

▶ **Corollary 7.** *There is at most one edge in M_2 that is a closing edge for two components.*

We now describe how to add the edge e^* to Γ^* if such an edge exists to obtain the final drawing Γ. Let u and v be the endvertices of e^* with u being the origin vertex of G. By construction, u and v are in the first two components c and c' of the BFS traversal of $\mathcal{H}$. Since u is the first vertex in $\prec_y$, its y-coordinate is 1, i.e., u is the bottommost vertex of Γ^*. Also, since u is the first vertex of c' in $\prec_y$, it is incident to the closing edge of c' and c by definition, in particular, this edge is e^*. Note that this implies that the x-coordinate of v is 1, so v is the leftmost vertex of Γ^* and the first vertex in $\prec_x$. This ensures that v can be moved to the left and u to the bottom in order to draw the edge e^* crossing-free. In particular, moving v by n units to the left and u by n units to the bottom we can guarantee that e^* does not intersect the first quadrant $\mathbb{R}^2_+$, while by construction any other edge (not incident to u or v) lies in $\mathbb{R}^2_+$. Since e^* is the only edge of M_2 incident to u and v, it remains to consider the edges of M_1 and M_3 incident to u or v. Observe that the edge of M_1 incident to v remains horizontal, while the edge of M_3 incident to u remains vertical. Finally, the edge of M_1 incident to u is crossing free in Γ^*, since there is no vertex below it, hence it remains crossing-free after moving u to the bottom. Similarly, the edge of M_3 incident to v is crossing free in Γ^*, since there is not vertex to the left of it, hence it remains crossing-free after moving v to the left. Together with Lemma 6 we obtain that Γ is a RAC drawing of G. We complete the proof of Theorem 3 by discussing the time complexity and the required area. We construct the components of H_y and H_x based on the given edges-coloring using BFS in $\mathcal{O}(n)$ time. To define $\prec_y$ and $\prec_x$, we choose the origin vertex u and the components c and c' for the start of the BFS of $\mathcal{H}$ in linear time. We then traverse every edge of G at most twice. Hence, this step takes $\mathcal{O}(n)$ time in total. Assigning the vertex coordinates, by first iterating through $\prec_y$ and $\prec_x$ and then possibly moving the end-vertices of e^*, clearly takes $\mathcal{O}(n)$ time again, hence the drawing can be computed in linear time.

For the area, we observe that the initial x- and y-coordinates for all the vertices range between 1 and n. Since we possibly move the origin vertex u and its M_2 neighbor v by n units each, the drawing area is at most $2n \times 2n$.

We conclude this section by mentioning two results that form generalizations of our approach of Theorem 3. In this regard, we need the notion of oddness of a bridgeless 3-regular graph, which is defined as the minimum number of odd cycles in any possible 2-factor of it. Theorem 8 is limited to oddness-2, while Theorem 9 provides an upper bound on the number of edges requiring one bend that is linear in the oddness; their proofs are in [2].

▶ **Theorem 8.** **(*)** *Every bridgeless* 3*-regular graph with oddness* 2 *admits a RAC drawing in quadratic area which can be computed in subquadratic time.*

▶ **Theorem 9.** **(*)** *Every bridgeless* 3*-regular graph with oddness* $k \geq 2$ *admits a* 1*-bend RAC drawing in quadratic area where at most* k *edges require one bend.*

4 1-Bend RAC Drawings of Degree-4 graphs

In this section, we focus on degree-4 graphs and show that they admit 1-bend RAC drawings.

▶ **Theorem 10.** *Given a degree-*4 *graph* G *with* n *vertices, it is possible to compute in* $O(n)$ *time a* 1*-bend RAC drawing of* G *with* $O(n^2)$ *area.*

Proof. By Theorem 2, we augment G into a directed 4-regular multigraph G' with edge disjoint 2-factors F_1 and F_2. Let G_s be the graph obtained from G' as follows. For each vertex u of G' with incident edges $(a_1, u), (u, b_1) \in F_1$ and $(a_2, u), (u, b_2) \in F_2$, we add two vertices u_s and u_t to G_s that are incident to the following five edges: u_s is incident to the two incoming edges of u, namely, (a_1, u_s) and (a_2, u_s), while u_t is incident to (u_t, b_1) and (u_t, b_2). Finally, we add the edge (u_s, u_t) to G_s, which we call *split-edge*.

By construction, G_s is 3-regular and 3-edge colorable, since each vertex of it is incident to one edge of F_1, one edge of F_2 and one split-edge. By applying the algorithm of Theorem 3 to G_s, we obtain a RAC drawing Γ_s of G_s, such that the matching M_2 in the algorithm is the one consisting of all the split-edges. To obtain a 1-bend drawing for G', it remains to merge the vertices u_s and u_t for every vertex u in G'. We place u at the position of u_s in Γ_s. We draw each outgoing edge (u, x) of u as a polyline with a bend placed close to the position of u_t in Γ_s (the specific position will be discussed later), which implies that the two segments are close to the edges (u_s, u_t) and (u_t, x) in Γ_s, respectively. This guarantees that any edge has exactly one bend, as any edge is outgoing for exactly one of its endvertices.

We next discuss how we place the bends for the outgoing edges of u. Since each split-edge belongs to M_2, it is drawn in Γ_s either as the diagonal of a 1 × 1 grid box or as a closing edge. Also, since the outgoing edges of u are in M_1 and M_3, they are drawn as horizontal and vertical line-segments.

Assume first that the split-edge of u is the diagonal of a 1 × 1 grid box; see Fig. 3a. If the outgoing edge (u_t, x) belongs to M_1, then we place the bend of (u, x) either half a unit to the right of u_t if x is to the right of u_t in Γ_s, or half a unit to its left otherwise. Symmetrically, if (u_t, x) belongs to M_3, we place the bend half a unit either above or below u_t.

Assume now that the split-edge of u is a closing edge in exactly one of H_y or H_x, say w.l.o.g. of a cycle c in H_x, i.e., it spans the whole x-interval of c. By construction, the outgoing edge of (u_t, x) that belongs to M_3 is a vertical line-segment attached above u, as either u_t or u_s are the first vertex of c in $\prec_y$; in the latter case, u_t is the second vertex of c in $\prec_y$ by construction. If the edge (u_t, x) belongs to M_3, we place the bend exactly at the computed position of u_t. If (u_t, x) belongs to M_1, we place it either half a unit to the right of u_t if x is to the right of u_t in Γ_s, or half a unit to its left otherwise; see Fig. 3b.

Assume last that the split-edge of u is the closing edge e for a H_y and a H_x cycle, which is unique by Corollary 7. As discussed for the analogous case in Section 3 (see the discussion following Corollary 7), one of u_s and u_t, say w.l.o.g. u_s, is the leftmost, while the other u_t is the bottommost vertex in Γ_s. For the placement of the bends, we slightly deviate from our approach above. Let (u_t, x) and (u_t, y) be the two edges of M_1 and M_3 incident to u_t in G_s. Then, it is not difficult to find two grid points b_x and b_y sufficiently below the positions of x and y in Γ_s, such that (u, x) and (u, y)

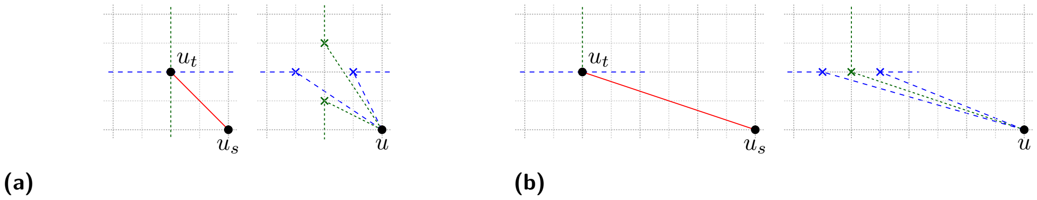


Figure 3 Illustration on how to place the bends in the proof of Theorem 10. To merge the vertices u_s and u_t of a vertex u in G', u is placed at the position of u_s. The bends of the outgoing edges at u are placed close to the position of u_t in the drawing depending on their orientation.

drawn by bending at b_x and b_y do not cross. Since no two bends overlap, no new crossings are introduced and the slopes of the segments involved in crossings are not modified, the obtained drawing Γ' is a 1-bend RAC drawing for G' (and thus for G).

Regarding the time complexity, we observe that we can apply Theorem 2 and the split-operation in $\mathcal{O}(n)$ time. The split operation immediately yields a valid 3-coloring of the edges, hence we can apply the algorithm of Theorem 3 to obtain Γ_s in $\mathcal{O}(n)$ time. Finally, contracting the edges can clearly be done in $\mathcal{O}(n)$ time, as it requires a constant number of operations per edge. For the area, we observe that in order to place the bends, we have to introduce new grid-points, but we at most double the number of points in any dimension, hence we still maintain the asymptotic quadratic area guaranteed by Theorem 3. ◀

The following theorem, whose proof is in [2], provides an alternative construction which additionally guarantees a linear number of edges drawn as straight-line segments.

▶ **Theorem 11.** **(*)** *Given a degree-4 graph G with n vertices and m edges, it is possible to compute in $O(n)$ time a 1-bend RAC drawing of G with $O(n^2)$ area where at least $\frac{m}{8}$ edges are drawn as straight-line segments.*

5 RAC drawings of 7-edge-colorable degree-7 graphs

We prove that 7-edge-colorable degree-7 graphs admit 2-bend RAC drawings by proving the following slightly stronger statement.

▶ **Theorem 12.** *Given a degree-7 graph G decomposed into a degree-6 graph H and a matching M, it is possible to compute in $O(n)$ time a 2-bend RAC drawing of G with $O(n^2)$ area.*

Since H is a degree-6 graph, it admits a decomposition into three disjoint (directed) 2-factors F_1, F_2 and F_3 after applying Theorem 2 and (possibly) augmenting H to a 6-regular (multi)-graph. To distinguish between directed and undirected edges, we write $\{u, v\}$ to denote an undirected edge between u and v, while (u, v) denotes a directed edge from u to v. In the following, we will define two total orders $\prec_x$ and $\prec_y$, which will define the x- and y-coordinates of the vertices of G, respectively. We define $\prec_y$ such that the vertices of each cycle in F_1 will be consecutive in $\prec_y$. Initially, for any cycle of F_1, the specific internal order of its vertices in $\prec_y$ is specified by one of the two traversals of it; however, we note here that this choice may be refined later in order to guarantee an additional property (described in Lemma 13). The definition of $\prec_x$ is more involved and will also be discussed later. Theorem 2 guarantees that the edges of F_2 (F_3) are oriented such that any vertex has at most one incoming and one outgoing edge in F_2 (F_3). Once $\prec_y$ and $\prec_x$ are computed, each vertex u of G will be mapped to point $(8i, 8j)$ of the Euclidean plane provided that u is the i-th vertex in $\prec_x$ and the j-th vertex in $\prec_y$. Each vertex u is associated with a closed box $B(u)$ centered at u of size 8×8. We aim at computing a drawing of G in which

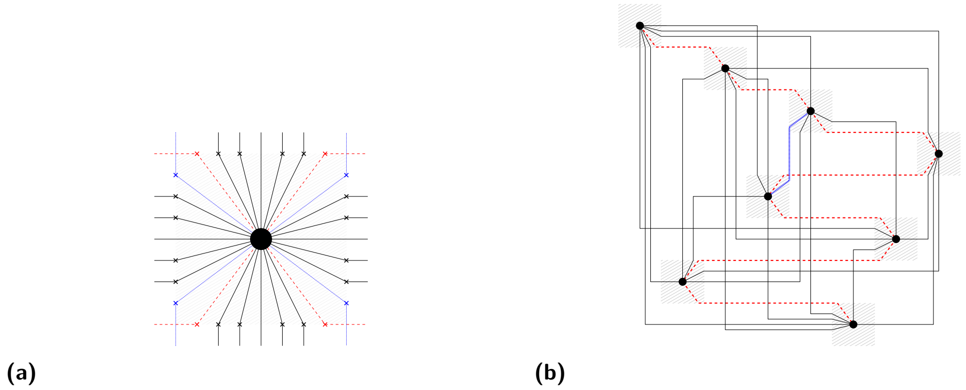


Figure 4 Edge routing in the 8×8 box $B(u)$ of a vertex u (a). Red dashed (blue dotted) ports are reserved for horizontal (vertical) type-2 edges. A 2-bend RAC drawing of K_8 is shown in (b). In this drawing, red dashed edges are horizontal type-2, while the blue dotted one is vertical type-2.

(i) no two boxes overlap, and
(ii) the edges are drawn with two bends each so that only the edge-segments that are incident to u are contained in the interior of $B(u)$, while all the other edge-segments are either vertical or horizontal.

This guarantees that the resulting drawing is 2-bend RAC; see Fig. 4b.

In the final drawing, all edges will be drawn with exactly three segments, out of which either one or two are *oblique*, i.e., they are neither horizontal nor vertical. It follows from (ii) that the bend point between an oblique segment and a vertical (horizontal) segment lies on a horizontal (vertical) side of the box containing the oblique segment. During the algorithm, we will classify the edges as *type*-1 or *type*-2. Type-1 edges will be drawn with one oblique segment, while type-2 edges with two oblique segments. In particular, for a type-1 edge (u, v), we further have that the oblique segment is incident to v, which implies that (u, v) occupies an orthogonal port at u. On the other hand, a type-2 edge (u, v) requires that $B(u)$ and $B(v)$ are *aligned* in y (in x), i.e., there exists a horizontal (vertical) line that is partially contained in both $B(u)$ and $B(v)$, in order to draw the middle segment of (u, v) horizontally (vertically). By construction, this is equivalent to having u and v consecutive in $\prec_y$ ($\prec_x$). These alignments guarantee that if we partition the edges of F_1 into $\bar{F_1}$ and $\hat{F_1}$ containing the closing and non-closing ones, respectively, then it is possible to draw $\hat{F_1}$ as a horizontal type-2 edge (independent of the x-coordinate of its endvertices), as its endvertices are consecutive in $\prec_y$ by construction. Thus, we can put our focus on edges in $\bar{F_1} \cup F_2 \cup F_3$, which we initially classify as type-1 edges (by orienting each edge (u, v) of $\bar{F_1}$ from u to v if $u \prec_y v$). We refine $\prec_y$ using the concept of *critical vertices*. Namely, for a vertex u of G, the direct successors of u in $\bar{F_1} \cup F_2 \cup F_3$ are the critical vertices of u, which are denoted by $c(u)$. Based on the relative position of u to its critical vertices in $\prec_y$, we label u as (α, β), if α vertices $t_1, \dots, t_\alpha$ of $c(u)$ are after u in $\prec_y$ and β vertices $b_1, \dots, b_\beta$ before. We refer to $t_1, \dots, t_\alpha$ ($b_1, \dots, b_\beta$) as the *upper* (*lower*) *critical neighbors* of u. An edge connecting u to an upper (lower) critical neighbor is called *upper critical* (*lower critical*, respectively). More general, the upper and lower critical edges of u are its *critical edges*.

Note that $2 \leq \alpha + \beta \leq 3$ as any vertex has exactly one outgoing edge in F_2 and F_3 and at most one in $\bar{F_1}$, that is, the number of upper and lower critical neighbors of vertex u ranges between 2 and 3. It follows that the label of each vertex of H is in $\{(0, 2), (1, 1), (2, 0), (1, 2), (2, 1), (3, 0)\}$; refer to these labels as the *feasible labels* of the vertex.

Observe that a $(3,0)$-, $(2,1)$- or $(1,2)$-label implies that the vertex is incident to a closing edge of F_1 (hence, each cycle in F_1 has at most one such vertex, which is its first one in $\prec_y$). This step will complete the definition of $\prec_y$.

▶ **Lemma 13.** **(*)** *For each cycle c of F_1, there is an internal ordering of its vertices followed by a possible reorientation of one edge in $F_2 \cup F_3$, such that in the resulting $\prec_y$*

(a) *every vertex of c has a feasible label,*
(b) *no vertex of c has label $(3,0)$, and*
(c) *if there exists a $(1,2)$-labeled vertex in c, then its (only) upper critical neighbor belongs to c.*

Now that $\prec_y$ is completely defined, we orient any edge $(u,v) \in M$ from u to v if and only if $u \prec_y v$. In this case, we further add v as a critical vertex of u. This implies that some vertices can have one more critical upper neighbor, which then gives rise to the new following labels, which we call *tags* for distinguishing: $\{[3,1],[3,0],[2,2],[2,1],[2,0],[1,2],[1,1],[0,2]\}$. In this context, Lemma 13 guarantees the following property.

▶ **Property 14.** *Any cycle c of F_1 has at most one vertex with tag $[\alpha,\beta]$ such that $\alpha+\beta=4$.*

Next, we compute the final drawing satisfying Properties (i) and (ii) by performing two iterations over the vertices of G in reverse $\prec_y$ order. In the first, we specify the final position of each vertex of G in $\prec_x$ and classify its incident edges while maintaining the following Invariant 15. In the second one, we exploit the computed $\prec_x$ to draw all edges of G.

▶ **Invariant 15.** *The endvertices of each vertical type-2 edge are consecutive in $\prec_x$. Further, any vertex is incident to at most one vertical type-2 edge.*

The second part of Invariant 15 implies that the vertical type-2 edges form a set of independent edges. In this regard, we say that a vertex u is a *partner* of a vertex v in G if and only if u and v are connected with an edge in this set.

In the first iteration, we assume that we have processed the first i vertices $v_n,\dots,v_{n-i+1}$ of G in reverse $\prec_y$ order and we have added these vertices to $\prec_x$ together with a classification of their incident edges satisfying Invariant 15. We determine the position of v_{n-i} in $\prec_x$ based on the $\prec_x$ position of its upper critical neighbors. The incident edges of v_{n-i} are classified based on a case analysis on its tag $[\alpha,\beta]$. Recall that unless otherwise specified, every edge is a type-1 edge.

1. The tag of v_{n-i} is $[3,1]$ or $[3,0]$: Let a, b and c be the upper critical neighbors of v_{n-i}, which implies that they were processed before v_{n-i} by the algorithm and are already part of $\prec_x$. W.l.o.g. assume that $a \prec_x b \prec_x c$. By Invariant 15, vertex b is the partner of at most one already processed vertex b', which is consecutive with b in $\prec_x$. If b' exists and $b' \prec_x b$, then we add v_{n-i} immediately after b in $\prec_x$. Symmetrically, if b' exists and $b \prec_x b'$, then we add v_{n-i} immediately before b in $\prec_x$. Otherwise, we add v_{n-i} immediately before b in $\prec_x$. This guarantees that v_{n-i} is placed between a and c in $\prec_x$ and that Invariant 15 is satisfied, since none of the upper critical edges incident to v_{n-i} was classified as a type-2 edge.
2. The tag of v_{n-i} is $[2,1]$,$[2,0]$, $[1,2]$, $[1,1]$ or $[0,2]$: By appending v_{n-i} to $\prec_x$, we maintain Invariant 15, since none of the upper critical edges incident to v_{n-i} was classified as type-2.
3. The tag of v_{n-i} is $[2,2]$: Let a and b be the upper critical neighbors of v_{n-i}, which implies that they were processed before v_{n-i} by the algorithm and are already part of $\prec_x$. W.l.o.g.

assume that $(v_{n-i}, a) \in M$. We classify the edge (v_{n-i}, b) as a vertical type-2 edge and we add v_{n-i} immediately before b in $\prec_x$. To show that Invariant 15 is maintained by this operation it is sufficient to show that b was not incident to a vertical type-2 edge before. Suppose for a contradiction that there is a vertex b' in $\{v_n, \dots, v_{n-i+1}\}$, such that (b, b') or (b', b) is a type-2 edge. As seen in the previous cases, this implies that b or b' has tag $[2, 2]$, respectively. Since in the $[2, 2]$ case the edge classified as type-2 is the one not in M and since any vertex that has tag $[2, 2]$ has label $(1, 2)$, by Lemma 13 it follows that vertical type-2 edges are chords of a cycle. Hence, b or b' would lie in the same cycle as v_{n-i}, which is a contradiction to Property 14, thus Invariant 15 holds.

Orders $\prec_x$ and $\prec_y$ define the placement of the vertices. By iterating over the vertices, we describe how to draw the edges to complete the drawing such that Properties (i) and (ii) are satisfied. We distinguish cases based on the tag of the current vertex v_i.

1. The tag of v_i is $[3, 1]$ or $[3, 0]$: Let $\{a, b, c\}$ be the upper critical neighbors of v_i. The construction of $\prec_x$ ensures that not all of $\{a, b, c\}$ precede or follow v_i in $\prec_x$, w.l.o.g. we can assume that $a \prec_x b, v_i \prec_x c$. Then, we assign the W-port at v_i to (v_i, a), the N-port at v_i to (v_i, b) and the E-port at v_i to (v_i, c). If v_i has a lower critical neighbor, we assign the S-port at v_i for the edge connecting v_i to it.
2. The tag of v_i is $[2, 1]$ or $[2, 0]$: Let $\{a, b\}$ be the upper critical neighbors of v_i. We assign the N-port at v_i to (v_i, a). Note that v_i was appended to $\prec_x$ during its construction. If $b \prec_x v_i$, we assign the W-port at v_i to (v_i, b). Otherwise, we assign the E-port at v_i to (v_i, b). The S-port is assigned to the lower critical edge of v_i, if present.
3. The tag of v_i is $[1, 2]$ or $[0, 2]$: This case is symmetric to the one above by exchanging the roles of upper and lower critical neighbors and N- and S-ports.
4. The tag of v_i is $[1, 1]$: Let a be the upper critical neighbor and b the lower critical neighbor of v_i. Then we assign the N-port to the edge (v_i, a) and the S-port to (v_i, b).
5. The tag of v_i is $[2, 2]$: Let $\{a, b\}$ and $\{c, d\}$ be the upper and lower critical neighbors of v_i. W.l.o.g. let $(v_i, a) \in M$. By Invariant 15 and construction, the edge (v_i, b) is a type-2 edge. The N- and S-ports at v_i are assigned to the edges (v_i, a) and (v_i, c). If $d \prec_x v_i$, we assign the W-port at v_i to (v_i, d). Otherwise, we assign the E-port at v_i to (v_i, d).

We describe how to place the bends of the edges on each side of the box $B(u)$ of an arbitrary vertex u based on the type of the edge that is incident to u, refer to Fig. 4a. We focus on the bottom side of $B(u)$. Let (x_u, y_u) be the position of u that is defined by $\prec_x$ and $\prec_y$. Recall that the box $B(u)$ has size 8×8. Let $e = \{u, v\}$ be an edge incident to u. If e is a horizontal type-2 edge, then we place its bend at $(x_u - 3, y_u - 4)$, if $v \prec_x u$, otherwise we have $u \prec_x v$ and we place the bend at $(x_u + 3, y_u - 4)$. If e is a type-1 edge that uses the S-port of u, then segment of e incident to u passes through point $(x_u, y_u - 4)$. If e is a type-1 edge oriented from v to u such that $v \prec_y u$ and e uses either the W-port or the E-port of v, then we place the bend at $(x_u + i, y_u - 4)$ with $i \in \{-2, -1, 1, 2\}$. Since any vertex has at most four incoming type-1 edges after applying Lemma 13, we can place the bends so that no two overlap. No other edge crosses the bottom side of $B(u)$. The description for the other sides can be obtained by rotating this scheme; for the left and the right side the type-2 edges are the vertical ones.

We now describe how to draw each edge $e = (u, v)$ of G based on the relative position of u and v in $\prec_x$ and $\prec_y$ and the type of e. Refer to Fig. 4b. Suppose first that e is a type-2 edge. If e is a horizontal type-2 edge, then u and v are consecutive in $\prec_y$ and $B(u)$ and $B(v)$ are aligned in y-coordinate, in particular, there is a horizontal line that contains the top side of one box and the bottom side of the other, hence it passes through the two assigned bend-points, which implies that the middle segment is horizontal. Similarly, if e is a vertical type-2 edge, then u and v are consecutive in $\prec_x$ by Invariant 15. Hence, the assigned points for the bends define a vertical middle segment. Suppose now that e is a type-1 edge. The case analysis for the second iteration over the vertices guarantees that for any relative position of v to u, we assigned an appropriate orthogonal port at u which allows to find a point on the first segment, such that the orthogonal middle-segment of the edge e (that is perpendicular to the first) can reach the assigned bend point on the boundary of $B(v)$.

We argue that the constructed drawing is indeed 2-bend RAC as follows. By construction, every edge consists of three segments and no bend overlaps with an edge or with another bend. Each vertical (horizontal) line either crosses only one box or contains the side of exactly two boxes, whose corresponding vertices are consecutive in $\prec_x$ ($\prec_y$). This implies that if a vertical (horizontal) segment of an edge shares a point with the interior of a box, then this box correspond to one of its endvertices. Further, any oblique segment is fully contained inside the box of its endvertex, hence crossings can only happen between a vertical and a horizontal segment which implies that the drawing is RAC.

To complete the proof of Theorem 12, we discuss the time complexity and the required area. We apply Theorem 2 to $G \setminus M$ to obtain F_1, F_2, F_3 in $\mathcal{O}(n)$ time. Computing the labels clearly takes $\mathcal{O}(n)$ time. For each cycle of F_1, the ordering of its internal vertices in Lemma 13 can be done in time linear in the size of the cycle by computing for each vertex the number of forward and backward edges, and of chords. Computing the tags takes $\mathcal{O}(n)$ time. In each of the following two iterations, we perform a constant number of operations per vertex. Hence we can conclude that the drawing can be computed in $\mathcal{O}(n)$ time. For the area, we can observe that the size of the grid defined by the boxes is $8n \times 8n$ and by construction, any vertex and any bend point is placed on a point on the grid.

▶ **Corollary 16.** *Given a 7-edge-colorable degree-7 graph with n vertices and a 7-edge-coloring of it, it is possible to compute in $O(n)$ time a 2-bend RAC drawing of it with $O(n^2)$ area.*

6 Conclusions and Open Problems

We significantly extended the previous work on RAC drawings for low-degree graphs in all reasonable settings derived by restricting the number of bends per edge to 0, 1, and 2. The following open problems are naturally raised by our work.

- Are all 4-edge-colorable degree-3 graphs RAC (refer to Question 1)?
- Are all degree-5 graphs 1-bend RAC? What about degree-6 graphs?
- Is it possible to extend Theorem 12 to all (i.e., not 7-edge-colorable) degree-7 graphs or even to (subclasses of) graphs of higher degree, e.g. Hamiltonian degree-8 graphs?
- While recognizing graphs that admit a (straight-line) RAC drawing is NP-hard [5], the complexity of the recognition problem in the 1- and 2-bend setting is still unknown.

References

1 Patrizio Angelini, Michael A. Bekos, Henry Förster, and Michael Kaufmann. On RAC drawings of graphs with one bend per edge. *Theor. Comput. Sci.*, 828-829:42–54, 2020. doi:10.1016/j.tcs.2020.04.018.

2 Patrizio Angelini, Michael A. Bekos, Julia Katheder, Michael Kaufmann, and Maximilian Pfister. RAC drawings of graphs with low degree. *CoRR*, abs/2206.14909, 2022. arXiv:2206.14909.

3 Patrizio Angelini, Luca Cittadini, Walter Didimo, Fabrizio Frati, Giuseppe Di Battista, Michael Kaufmann, and Antonios Symvonis. On the perspectives opened by right angle crossing drawings. *J. Graph Algorithms Appl.*, 15(1):53–78, 2011. doi:10.7155/jgaa.00217.

4 Evmorfia N. Argyriou, Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis. Geometric RAC simultaneous drawings of graphs. *J. Graph Algorithms Appl.*, 17(1):11–34, 2013. doi:10.7155/jgaa.00282.

5 Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis. The straight-line RAC drawing problem is NP-hard. *J. Graph Algorithms Appl.*, 16(2):569–597, 2012. doi:10.7155/jgaa.00274.

6 Karin Arikushi, Radoslav Fulek, Balázs Keszegh, Filip Moric, and Csaba D. Tóth. Graphs that admit right angle crossing drawings. *Comput. Geom.*, 45(4):169–177, 2012. doi:10.1016/j.comgeo.2011.11.008.

7 Michael A. Bekos, Walter Didimo, Giuseppe Liotta, Saeed Mehrabi, and Fabrizio Montecchiani. On RAC drawings of 1-planar graphs. *Theor. Comput. Sci.*, 689:48–57, 2017. doi:10.1016/j.tcs.2017.05.039.

8 Steven Chaplick, Fabian Lipp, Alexander Wolff, and Johannes Zink. Compact drawings of 1-planar graphs with right-angle crossings and few bends. *Comput. Geom.*, 84:50–68, 2019. doi:10.1016/j.comgeo.2019.07.006.

9 Norishige Chiba, Kazunori Onoguchi, and Takao Nishizeki. Drawing planar graphs nicely. *Acta Inform.*, 22:187–201, 1985. doi:10.1007/BF00264230.

10 Marek Chrobak and Thomas H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Inf. Process. Lett.*, 54(4):241–246, 1995. doi:10.1016/0020-0190(95)00020-D.

11 Hubert de Fraysseix, János Pach, and Richard Pollack. Small sets supporting Fáry embeddings of planar graphs. In Janos Simon, editor, *Symposium on the Theory of Computing*, pages 426–433. ACM, 1988. doi:10.1145/62212.62254.

12 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

13 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Henk Meijer. Area, curve complexity, and crossing resolution of non-planar graph drawings. *Theory Comput. Syst.*, 49(3):565–575, 2011. doi:10.1007/s00224-010-9275-6.

14 Walter Didimo. Right angle crossing drawings of graphs. In Seok-Hee Hong and Takeshi Tokuyama, editors, *Beyond Planar Graphs*, pages 149–169. Springer, 2020. doi:10.1007/978-981-15-6533-5_9.

15 Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Workshop on Algorithms and Data Structures*, volume 5664 of *LNCS*, pages 206–217. Springer, 2009. doi:10.1007/978-3-642-03367-4_19.

16 Walter Didimo, Peter Eades, and Giuseppe Liotta. A characterization of complete bipartite RAC graphs. *Inf. Process. Lett.*, 110(16):687–691, 2010. doi:10.1016/j.ipl.2010.05.023.

17 Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. *Theor. Comput. Sci.*, 412(39):5156–5166, 2011. doi:10.1016/j.tcs.2011.05.025.

18 Walter Didimo and Giuseppe Liotta. The crossing-angle resolution in graph drawing. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 167–184. Springer, 2013.

19 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

20 Peter Eades, Antonios Symvonis, and Sue Whitesides. Three-dimensional orthogonal graph drawing algorithms. *Discret. Appl. Math.*, 103(1-3):55–87, 2000. doi:10.1016/S0166-218X(00)00172-4.

21 Henry Förster and Michael Kaufmann. On compact RAC drawings. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *European Symposium on Algorithms*, volume 173 of *LIPIcs*, pages 53:1–53:21. Schloss Dagstuhl, 2020. doi:10.4230/LIPIcs.ESA.2020.53.

22 István Fáry. On straight lines representation of planar graphs. *Acta Sci. Math. (Szeged)*, 11:229–233, 1948.

23 M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983. doi:10.1137/0604033.

24 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.

25 Seok-Hee Hong and Takeshi Tokuyama, editors. *Beyond Planar Graphs*. Springer, 2020. doi:10.1007/978-981-15-6533-5.

26 Weidong Huang, Peter Eades, and Seok-Hee Hong. Larger crossing angles make graphs easier to read. *J. Vis. Lang. Comput.*, 25(4):452–465, 2014. doi:10.1016/j.jvlc.2014.03.001.

27 Julius Petersen. Die Theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891. doi:10.1007/BF02392606.

28 Helen C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interact. Comput.*, 13(2):147–162, 2000. doi:10.1016/S0953-5438(00)00032-1.

29 Marcus Schaefer. Rac-drawability is ∃ R-Complete. In Helen C. Purchase and Ignaz Rutter, editors, *Graph Drawing and Network Visualization*, volume 12868 of *LNCS*, pages 72–86. Springer, 2021. doi:10.1007/978-3-030-92931-2_5.

30 Sherman K. Stein. Convex maps. *Proc. American Math. Soc.*, 2(3):464–466, 1951.

31 E. Steinitz and H. Rademacher. *Vorlesungen über die Theorie der Polyeder*. Julius Springer, Berlin, Germany, 1934.

32 Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.

33 William Thomas Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13:743–768, 1963.

34 Klaus Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

Polynomial Time Algorithm for ARRIVAL on Tree-Like Multigraphs

David Auger
DAVID Lab.,UVSQ, Université Paris Saclay, 45 avenue des Etats-Unis, 78000, Versailles, France

Pierre Coucheney
DAVID Lab.,UVSQ, Université Paris Saclay, 45 avenue des Etats-Unis, 78000, Versailles, France

Loric Duhazé
DAVID Lab., UVSQ, Université Paris Saclay, 45 avenue des Etats-Unis, 78000, Versailles, France
LISN, Université Paris Saclay, France

Abstract

A *rotor walk* in a directed graph can be thought of as a deterministic version of a Markov Chain, where a pebble moves from vertex to vertex following a simple rule until a terminal vertex, or sink, has been reached. The *ARRIVAL problem*, as defined by Dohrau et al. [8], consists in determining which sink will be reached. While the walk itself can take an exponential number of steps, this problem belongs to the complexity class NP ∩ co-NP without being known to be in P. In this work, we define a class of directed graphs, namely *tree-like multigraphs*, which are multigraphs having the global shape of an undirected tree. We prove that in this class, ARRIVAL can be solved in almost linear time, while the number of steps of a rotor walk can still be exponential. Then, we give an application of this result to solve some deterministic analogs of stochastic models (e.g., Markovian decision processes, Stochastic Games).

2012 ACM Subject Classification Mathematics of computing

Keywords and phrases Rotor-routing, Rotor Walk, Reachability Problem, Game Theory, Tree-like Multigraph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.12

Related Version *Full Version*: `https://arxiv.org/abs/2204.13151`

Funding *Loric Duhazé*: This research was partially supported by Labex DigiCosme (project ANR11LABEX0045DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex ParisSaclay (ANR11IDEX000302).

Acknowledgements We want to thank Johanne Cohen for her advises and attentive review.

1 Introduction

The *rotor routing*, or *rotor walk model*, has been studied under different names: *eulerian walkers* [17, 16] and *patrolling algorithm* [19]. It shares many properties with a more algebraic focused model: *abelian sandpiles* [4, 15]. We can cite [12] and [15] as general introductions to this cellular automaton.

Let us explain briefly how a rotor walk works. Consider a directed graph and for each vertex v, if v has k outgoing arcs, number these arcs from 1 to k. Then, we place a pebble on a starting vertex and proceed to the walk. On the initial vertex, the pebble moves to the next vertex according to arc 1. It does the same on the second vertex and so on. But the second time that a vertex is reached, the pebble will move according to arc 2, and so on until arc k has been used, and then we start again with arc 1.

In this work, we fix a set of vertices that we call sinks and stop the walking process when a sink is reached. The problem of determining, for a starting configuration (numbering) of arcs and an initial vertex, which sink will be reached first is the ARRIVAL problem. It

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 12; pp. 12:1–12:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is defined in [8] together with a proof that the problem belongs to the complexity class NP ∩ co-NP, but it is not known to be in P. It has then been shown in [10] that the problem is in the smaller complexity class UP ∩ co-UP, and a subexponential algorithm has been proposed in [11], based on computing a Tarski fixed point. This algorithm is even polynomial if the graph is almost acyclic (in a certain sense). A direct application of the rotor-routing automaton is that several structural properties of Markov chains can be approximated or bounded by rotor walks see [6, 7, 9]. It seems natural to extend these results to one and two player variants and define rotor analogs for *Markov decision processes* and *stochastic games* [18]. It is proved in [18] that deciding if a player can ensure some value is NP-complete for the one-player version and PSPACE-complete for the two-player version.

Contributions and Organization of the Paper

We define the class of *tree-like multigraphs*, and prove that while the number of steps needed to complete a rotor walk can still be exponential, the tree-like structure helps to efficiently solve ARRIVAL in linear time. We also extend our results to one-player and two-player variants. It is to be noted that tree-like multigraphs are not almost acyclic in the sense of [11], thus their algorithm does not run in polynomial time in our case.

In Section 2, we first give some standard definitions for multigraphs and proceed to define rotor walks in this context, together with different rotor-routing notions (*exit pattern, cycle pushing*, etc.). Next, in Section 3, we define tree-like multigraphs and our main tool to study ARRIVAL on these graphs, namely the *return flow.* Then, in Section 4, we sketch a polynomial algorithm that solves ARRIVAL and finally we show that this algorithm can be used to efficiently solve both the one and two player case.

The following table summarizes our results (bold), i.e. time complexity of computing the sink (or the optimal sink, for one-player and two-player) reached by a particle starting on a particular vertex in a graph $G = (V, A)$. Results on simple tree-like multigraphs depicted here and quickly presented in this paper are detailed in our extended version [1]. The first column states the complexity of the natural algorithm to solve these problems, namely simulating the *rotor walk.*

	Rotor Walk	0 player	1 player	2 players
General digraph	*exponential*	NP ∩ coNP	NP-complete	PSPACE-complete
Tree-like multigraph	**exponential**	$\boldsymbol{O(\lvert A\rvert)^\dagger}$	$\boldsymbol{O(\lvert A\rvert)}$	$\boldsymbol{O(\lvert A\rvert)}$
Simple Tree-like multigraph	$\boldsymbol{O(\lvert V\rvert^2)}$	$\boldsymbol{O(\lvert V\rvert)^\dagger}$	$\boldsymbol{O(\lvert V\rvert)^\dagger}$	$\boldsymbol{O(\lvert V\rvert)^\dagger}$

The † indicates the cases where we can solve the problem for all vertices of the graph at the same time with this complexity. Note that all proofs that do not appear directly in the document are detailed in our extended version [1].

2 Basic Definitions

2.1 Directed Multigraphs

In this paper, unless stated otherwise, we always consider a directed multigraph $G = (V, \mathcal{A}, h, t)$ where V is a finite set of *vertices*, $\mathcal{A}$ is a finite set of *arcs*, and h (for *head*) and t (for *tail*) are two maps from $\mathcal{A}$ to V defining incidence between arcs and vertices. For sake of clarity, we only consider graphs without arcs of the form $h(a) = t(a)$ (i.e. loops). All our complexity results would remain true if we authorized them. Note that multigraphs can have multiple arcs with the same head and tail. Let $u \in V$ be a vertex, we denote by $\mathcal{A}^+(u)$ (resp. $\mathcal{A}^-(u)$) the subset of arcs $a \in \mathcal{A}$ with tail u (resp. with head u).

Let $\Gamma^+(u)$ (resp. $\Gamma^-(u)$) be the subset of vertices $v \in V$ such that there is an arc $a \in \mathcal{A}$ with $h(a) = v$ and $t(a) = u$ (resp.$h(a) = u$ and $t(a) = v$). A graph such that for all $u \in V$ we have $|\mathcal{A}^+(u)| = |\Gamma^+(u)|$ is called *simple.* A vertex u for which $|\Gamma^+(u) \cup \Gamma^-(u)| = 1$ is called a *leaf.*

2.2 Rotor Routing Mechanics

Rotor Graphs

Let $G = (V, \mathcal{A}, h, t)$ be a multigraph.

▶ **Definition 1** (Rotor Order). *We define a* rotor order *at $u \in V$ as an operator denoted by θ_u such that:*

- $\theta_u : \mathcal{A}^+(u) \to \mathcal{A}^+(u)$
- *for all $a \in \mathcal{A}^+(u)$, the orbit $\{a, \theta_u(a), \theta_u^2(a), ..., \theta_u^{|\mathcal{A}^+(u)|-1}(a)\}$ of arc a under θ_u is equal to $\mathcal{A}^+(u)$, where $\theta_u^k(a)$ is the composition of θ_u applied to arc a exactly k times.*

Observe that each arc of $\mathcal{A}^+(u)$ appears exactly once in any orbit of θ_u. Now, we will integrate the operator θ_u to our graph structure as follows.

▶ **Definition 2** (Rotor Graph). *A* rotor graph *G is a (multi)graph G together with:*

- *a partition $V = V_0 \cup S_0$ of vertices, where $S_0 \neq \emptyset$ is a particular set of vertices called sinks, and V_0 is the rest of the vertices;*
- *a rotor order θ_u at each $u \in V_0$.*

In this document, unless stated otherwise, all the graphs we consider are rotor graphs with $G = (V_0, S_0, \mathcal{A}, h, t, \theta)$.

▶ **Definition 3** (Rotor Configuration). *A* rotor configuration *(or simply configuration) of a rotor graph G is a mapping ρ from V_0 to $\mathcal{A}$ such that $\rho(u) \in \mathcal{A}^+(u)$ for all $u \in V_0$. We denote by $\mathcal{C}(G)$ the set of all rotor configurations on the rotor graph G.*

What will be called a *particle* in the remaining of this paper is a pebble which will move from one vertex to another; hence the position of the particle is characterized by a single vertex. This movement, called rotor walk, follows specific rules that we detail after.

▶ **Definition 4** (Rotor-particle configuration). *A* rotor-particle configuration *is a couple (ρ, u) where ρ is a rotor configuration and $u \in V$ denotes the position of a particle.*

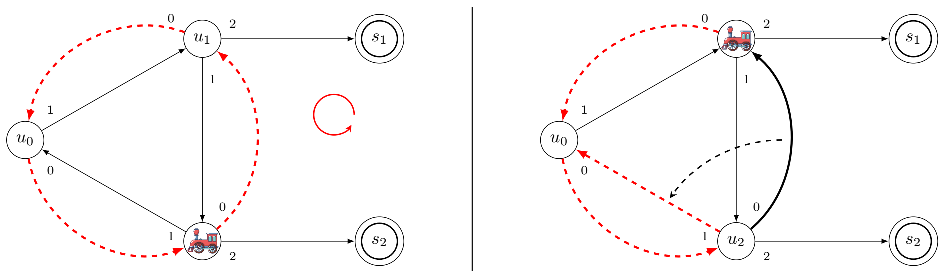
Rotor Walk

▶ **Definition 5.** *Let us define two mappings on $\mathcal{C}(G) \times V_0$:*

- **turn***, with values in $\mathcal{C}(G)$, is defined by* **turn**$(\rho, u) = \rho'$ *where ρ' is equal to ρ except at u where $\rho'(u) = \theta_u(\rho(u))$.*
- **move***, with values in V, is defined by* **move**$(\rho, u) = h(\rho(u))$.

By composing those mappings, we are now ready to define the *routing of a particle* which is a single step of a rotor walk.

▶ **Definition 6** (Routing of a Particle). *The* ***routing*** *of a particle (illustrated in Figure 1) from a rotor-particle configuration (ρ, u) is a mapping:* **routing** $: \mathcal{C}(G) \times V_0 \longrightarrow \mathcal{C}(G) \times V$ *defined by* **routing**$(\rho, u) = (\rho', v)$*, with $\rho' =$* **turn**(ρ, u) *and $v =$* **move**(ρ, u)*. This can be viewed as the particle first travelling through $\rho(u)$ and then $\rho(u)$ is replaced by $\theta_u(\rho(u))$.*



(a) Let ρ be the rotor configuration depicted by the red arcs in dashes.

(b) The red rotor configuration in dashes is obtained by processing the operation: $\mathbf{routing}(\rho, u_2)$.

Figure 1 A rotor-routing where the particle is depicted by a train and starts on u_2. The sink-vertices are s_1 and s_2. The red arcs in dashes represent the current rotor configuration. The rotor orders on the different vertices are anticlockwise, i.e. they are: θ_{u_0}:(u_0,u_2),(u_0,u_1); θ_{u_1}: (u_1,u_0),(u_1,u_2),(u_1,s_1); θ_{u_2}: (u_2,u_1),(u_2,u_0),(u_2,s_2). These orders are also depicted by the numbers around each vertex.

▶ Remark 7. Our routing rule (move, then turn) is slightly different than the one defined in [17] which is mostly used in the literature (turn, then move) but is more convenient to study ARRIVAL. The two rules are equivalent up to applying **turn** mapping on all vertices.

A rotor-routing is in fact a single step of a rotor walk.

▶ **Definition 8** (Rotor Walk). *A* rotor walk *is a (finite or infinite) sequence of rotor-particle configurations* $(\rho_i, u_i)_{i\geq 0}$*, which is recursively defined by* $(\rho_{i+1}, u_{i+1}) = \mathbf{routing}(\rho_i, u_i)$ *as long as* $u_i \in V_0$*.*

One can check that the sequence of vertices in the rotor walk starting from the rotor-particle configuration depicted on Figure 1(b) until a sink is reached is $u_1, u_0, u_2, u_0, u_1, u_2, s_2$.

Routing to Sinks

Now that we have defined our version of rotor-walk, we proceed to the corresponding version of the problem *ARRIVAL* similar to the one stated in [8].

▶ **Definition 9** (Maximal Rotor Walk). *A maximal rotor walk is a rotor walk such that in the case where it is finite, the last vertex must be a sink vertex* $s \in S_0 = V \setminus V_0$*.*

▶ **Definition 10** (Stopping Rotor Graph). *If, for any vertex* $u \in V$*, there exists a directed path from* u *to a sink* $s \in S_0$*, the graph is said to be* stopping.

The next lemma is a classical result on rotor walks (cf Lemma 16 in [12]).

▶ **Lemma 11** (Finite number of steps). *If* G *is stopping then any rotor walk in* G *is finite.*

The main objective of our work is to study the sink that will be reached by a maximal rotor walk from a rotor-particle configuration, if the rotor walk is finite.

▶ **Definition 12** (Exit Sink). *Let* $u \in V$*, let* ρ *be a configuration, if the maximal rotor walk starting from* (ρ, u) *is finite in* G*, then the sink reached by such a rotor walk is denoted by* $\mathbf{S}_G(\rho, u)$ *and called* exit sink *of* u *for the rotor configuration* ρ *in* G*.*

▶ **Definition 13** (Exit Pattern). *For a rotor configuration* ρ *on a stopping rotor graph* G*, the* exit pattern *is the mapping that associates to each vertex* $u \in V$*, its exit sink* $\mathbf{S}_G(\rho, u)$*.*

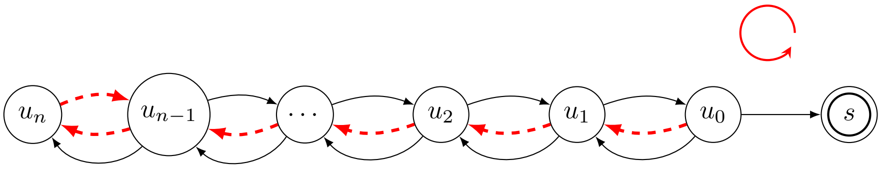


Figure 2 Family of path-like multigraphs where maximal routing can take an exponential number of steps in the number of vertices, here equal to $n+2$. The interior vertices (u_0 to u_{n-1}) have two arcs going left and one going right. Routing a particle from u_0 to sink s with the initial configuration ρ drawn with red arcs in dashes takes a non-polynomial time considering the anticlockwise rotor ordering on each vertex, depicted by the curved arc in red. One can check that the number of times a particle starting from u_0 will travel from u_i to u_{i+1} before reaching s is 2^{i+1}.

2.3 ARRIVAL and Complexity Issues

With our notations, ARRIVAL (see [8]) can be expressed as the following decision problem:

In a stopping rotor graph G,
given (ρ, u) and $s \in S_0$
does $\mathbf{S}_G(\rho, u) = s$?

This problem belongs to NP ∩ co-NP for simple graphs as shown in [8], but there is still no polynomial algorithm known to solve it. The case of eulerian simple graphs can be solved in time $O(|V+\mathcal{A}|^3)$, since a finite maximal rotor walk from (ρ, v) ends in at most $O(|V+\mathcal{A}|^3)$ routings (see [19]).

In the case of multigraphs, ARRIVAL still belongs to NP ∩ co-NP, since the polynomial certificate used for simple graphs in [8] remains valid. Our goal in this paper is to solve the problem in polynomial time for a particular class of multigraphs. Despite that, even for a path multigraph, the routing can be exponential, as shown on Figure 2.

2.4 Cycle Pushing

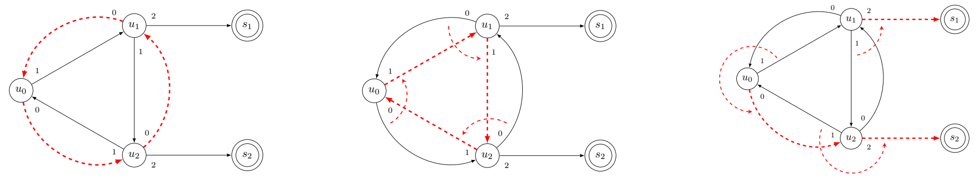
In order to speed-up the rotor walk process, a simple tool to avoid computing every step of the walk is the use of *cycle pushing*. We keep the terminology of *cycle pushing* used in [12] – even if what is called a cycle in this terminology is usually called a directed cycle or circuit in graph theory.

▶ **Definition 14** (Cycle *Pushing*). *Let ρ be a rotor configuration on G containing a directed cycle $C = (u_1, u_2, u_3, ..., u_k)$. We call* cycle push *the operation on ρ that leads to a configuration ρ' such that:*
- *for all $u_i \in C$, $\rho'(u_i) = \theta_{u_i}(\rho(u_i))$;*
- $\forall u \in V_0 \setminus C, \rho'(u) = \rho(u)$

*i.e. we process a **turn** on all $u_i \in C$.*

Note that a cycle push on a cycle C can also be computed by putting a particle on a vertex u of C and routing the particle until it comes back to u for the first time. Hence, cycle pushing is in a sense a shortcut on the rotor walk process, so in a manner similar to Lemma 11 it follows that in a stopping rotor graph, any sequence of cycle pushes is finite. This is a well known result in rotor walk studies (see [15]). It implies that, by processing a long enough sequence of successive cycle pushes, the resulting configuration contains no directed cycles. Such a sequence of cycle pushes is called *maximal.*

The two following results can be found in [12].



(a) Red Rotor Configuration ρ in dashes with rotor order on each vertex described by increasing numbers.

(b) The red configuration in dashes is obtained from a Cycle Push on the cycle $\{(u_0,u_2),(u_2,u_1),(u_1,u_0)\}$.

(c) Destination Forest (depicted by red arcs in dashes) computed by two successive cycle pushes on ρ.

Figure 3 Computation of the Destination Forest by successive cycle pushing.

▶ **Lemma 15** (Exit Pattern conservation for Cycle Push). *If G is a stopping rotor graph, for any rotor configuration ρ and configuration ρ' obtained from ρ by a cycle push, the exit pattern for ρ and ρ' is the same.*

▶ **Lemma 16** (Commutativity of Cycle Push). *In a stopping rotor graph, any maximal sequence of cycle pushes starting from a given rotor configuration ρ leads to the same acyclic configuration ρ' (ρ' does not contain a cycle).*

▶ **Definition 17** (Destination Forest). *We call the configuration obtained by a maximal cycle push sequence on ρ the* Destination Forest *of ρ, denoted by $D(\rho)$.*

The destination forest has a simple interpretation in terms of rotor walks: start a rotor walk by putting a particle on any vertex of a stopping graph G; consider a vertex $u \in V_0$; if the particle ever reaches u, it will leave u by arc $D(\rho)(u)$ on the last time it enters u.

In an acyclic configuration like $D(\rho)$, finding the exit pattern is simple, precisely:

▶ **Lemma 18** (Path to a sink). *If there is a directed path between $u \in V$ and $s \in S_0$ in ρ then $\mathbf{S}_G(\rho, u) = s$. It follows that from $D(\rho)$ one can compute the exit pattern of ρ in time complexity $O(|\mathcal{A}|)$.*

This gives us a new approach, since computing the exit pattern of a configuration ρ can be done by computing its Destination Forest $D(\rho)$. Note that by doing this we are solving a problem harder than ARRIVAL because we compute the exit sink of all vertices simultaneously.

▶ Remark 19. The strategy consisting in pushing cycles until the Destination Forest is reached can take an exponential number of steps. This is the case in the example drawn in Figure 2.

3 Tree-Like Multigraphs: Return Flow Definition

3.1 Tree-Like Multigraphs

To a directed multigraph $G = (V, \mathcal{A}, h, t)$ we associate:

- A simple directed graph $\hat{G} = (V, \hat{\mathcal{A}})$ such that, for $u, v \in V$ there is an arc from u to v in $\hat{G}$ if there is at least one arc $a \in \mathcal{A}$ with $t(a) = u$ and $h(a) = v$. Please note that even if there are multiple arcs a that satisfy this property, there is only one arc with tail u and head v in $\hat{\mathcal{A}}$. As it is unique, an arc from u to v in $\hat{\mathcal{A}}$ will simply be denoted by (u, v).
- A simple undirected graph $\overline{G} = (V, E)$ such that, for $u, v \in V$ there is an edge between u and v in $\overline{G}$ if and only if there is at least one arc $a \in \mathcal{A}$ such that $t(a) = u$ and $h(a) = v$ or $h(a) = u$ and $t(a) = v$.

▶ **Definition 20** (Tree-Like Rotor Multigraph and Tree-Like Multigraph). *A rotor multigraph $G = (V_0, S_0, \mathcal{A}, h, t, \theta)$ is tree-like if $\overline{G}$ is a tree and its set of leaves contains S_0. In that case, we say that $G = (V, \mathcal{A}, h, t)$ is a* tree-like multigraph.

Without loss of generality, in order to avoid some complexity in the notation and proofs, we will only study stopping tree-like rotor multigraphs.

▶ **Definition 21** (Sink Component). *A sink component is a strongly connected component in G that does not contain a sink vertex and such that there is no arc leaving the component.*

Note that all sink components can be computed in linear time.

▶ **Lemma 22** (Lemma 36 in [1]). *Consider a configuration ρ on a (not necessarily stopping) tree-like rotor multigraph $G = (V_0, S_0, \mathcal{A}, h, t, \theta)$. Consider a configuration ρ' on the* stopping *tree-like rotor multigraph $G' = (V_0', S_0', \mathcal{A}', h, t, \theta)$ where G' is obtained from G by replacing each sink component by a unique sink, and where $\rho'(u) = \rho(u)$ for each $u \in V_0'$. For any $u \in V$, finding the exit sink of u (if any) or the sink component reached by u in G for the configuration ρ can be directly determined by solving ARRIVAL for the configuration ρ' in G'.*

Note that, after replacing sink components by sinks, the multigraph G' may no longer be a tree-like multigraph but a forest-like multigraph. However we can split the study of ARRIVAL in each tree-like component of this forest since a particle cannot travel between those trees in a rotor walk.

3.2 Return Flows

Let us consider the simple example depicted on Figure 4 to motivate the introduction of (u, v)-subtrees and return flows, which is our main tool. In this figure, consider the routing of a particle starting at u:

- the particle moves from u to v_1, and stays for a while in the subtree T_1 – where it either reaches a sink or comes back to u. Suppose it comes back to u. Then:
- the particle moves from u to v_3, and either reaches a sink in T_3 or comes back to u. Suppose it comes back to u once again;
- the rotor walk goes on, in T_3, then in $T_2, T_1, T_1, T_3, \ldots$
- until finally the particle ends in a sink in one of the subtrees, say T_2.

Now consider only the relative movement that the particle had in T_2: it went from u into T_2 and back to u a number of times, before it ended in a sink. If we were to replace T_1 and T_3 by single arc leading back automatically to u, the relative movement in T_2 would have been exactly the same. The return flow will be a quantity that counts exactly the ability of each subtree to bounce back the particle to u. During the process described above, every time the particle enters a subtree and comes back to u, we can think of it as consuming a single unit of return flow in this subtree. The first time that a particle enters a subtree that has exactly one unit of return flow left, then the particle must end in a sink of that subtree.

▶ **Definition 23** ((u, v)-subtree). *Let $(u, v) \in \hat{\mathcal{A}}$. The (u, v)-subtree $T_{(u,v)}$ is a sub(multi)graph of G:*

- *whose vertices are all the vertices of the connected component of $\bar{G} \setminus \{u\}$ that contains v, together with u;*
- *whose arcs are all the arcs of G that link the vertices above, excepted in u where we remove all arcs of $\mathcal{A}^+(u)$ but a single arc a with head v. Such an arc a always exists because $(u, v) \in \hat{\mathcal{A}}$;*
- *whose rotor orders are unchanged except at u where $\theta_u(a) = a$.*

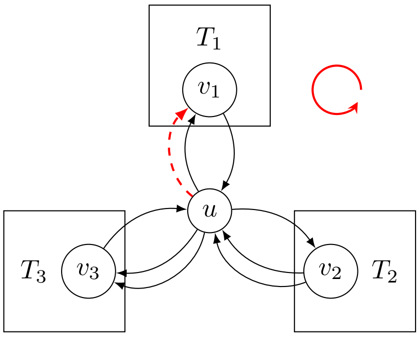


Figure 4 We sketch a stopping tree-like rotor multigraph as follows: a vertex u, its neighbours v_1, v_2, v_3 (that might be sinks), respectively belonging to T_1, T_2, T_3, the three connected components of $\bar{G} \setminus \{u\}$. In particular, we have $\hat{\mathcal{A}}^+(u) = \{(u, v_1); (u, v_2); (u, v_3)\}$. We consider the rotor configuration in red on u and θ_u is the anticlockwise order on the arcs of $\mathcal{A}^+(u)$.

Such a subtree is a (not necessarily stopping) tree-like rotor multigraph. A rotor configuration ρ in G can be thought of as a rotor configuration ρ' in $T_{(u,v)}$ by defining that $\rho'(u) = a$ and $\rho'(w) = \rho(w)$ for all $w \in T_{(u,v)}$.

We define a notion of *flow* for a particular starting vertex.

▶ **Definition 24** (Flow of (u, v))**.** *We define the* flow *on arc $(u, v) \in \hat{\mathcal{A}}$ for configuration ρ, denoted by $F_\rho(u, v)$, the number of times (possibly infinite) that an arc with tail u and head v is visited during the maximal rotor walk of a particle starting from the rotor-particle configuration (ρ, u). We denote by $F_\rho(u)$ the flow vector of (u, v) for every $v \in \Gamma^+(u)$.*

▶ **Definition 25** (Return flow)**.** *The* return flow *of arc $(u, v) \in \hat{\mathcal{A}}$ for configuration ρ, denoted by $r_\rho(u, v)$, is the flow on (u, v) in the (u, v)-subtree $T_{(u,v)}$.*

By definition of return flow, if $u \in S_0$, then $r_\rho(u, v) = 0$, and if $v \in S_0$, or if $(v, u) \notin \hat{\mathcal{A}}$ then $r_\rho(u, v) = 1$. Remark also that, even if the tree-like multigraph is stopping, it is not necessarily the case of any (u, v)-subtree: this is for instance the case of a leaf v which is not a sink such that $(u, v) \in \hat{\mathcal{A}}$. Finiteness of the return flow characterizes the subtrees that are stopping as stated in Lemma 26.

▶ **Lemma 26** (Lemma 40 in [1])**.** *Given a stopping tree-like multigraph G and an arc $(u, v) \in \hat{\mathcal{A}}$, the (u, v)-subtree $T_{(u,v)}$ is stopping if and only if for any rotor configuration on G, the return flow of (u, v) is finite.*

We give a bound on the maximal value of the return flow in a multigraph as it will be used to express our complexity results later.

▶ **Lemma 27** (Return flow bound, Lemma 41 in [1])**.** *Return flows can be written in at most $O(|\mathcal{A}|)$ bits.*

Return flows and flows are linked by the following result:

▶ **Lemma 28** (Lemma 42 in [1])**.** *Given a stopping tree-like rotor multigraph G, consider $u \in V_0$ and suppose that $h(D(\rho)(u)) = v$. Then:*

- *$F_\rho(u, v) = r_\rho(u, v)$;*
- *for all $w \in \Gamma^+(u) \setminus \{v\}$, $\quad F_\rho(u, w) < r_\rho(u, w)$;*
- *for all $w \in \Gamma^+(u) \cap \Gamma^-(u) \setminus \{v\}, r_\rho(w, u) = F_\rho(u, w) + 1$.*

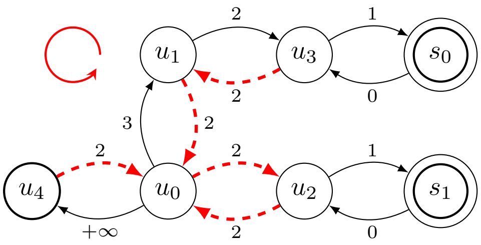


Figure 5 Examples of return flows in a simple graph. The rotor configuration is depicted by red arcs in dashes, with $S_0 = \{s_0, s_1\}$, and θ_{u_i} is the anticlockwise order on every vertex. We write the return flow of all arcs of $\hat{\mathcal{A}}$ next to their corresponding arc in $\mathcal{A}$. As a tutorial example, we detail the computation of $r_\rho(u_1, u_0)$ and $r_\rho(u_0, u_1)$. In the (u_1, u_0)-subtree, the particle will visit the following sequence of vertices $u_1, u_0, u_2, u_0, u_1, u_0, u_4, u_0, u_2, s_1$, where it crosses (u_1, u_0) twice, thus $r_\rho(u_1, u_0) = 2$. For the (u_0, u_1)-subtree, the sequence of vertices visited by the particle is $u_0, u_1, u_0, u_1, u_3, u_1, u_0, u_1, u_3, s_0$ hence $r_\rho(u_0, u_1) = 3$.

Thus, to compute $D(\rho)(u)$, we need to compute the flow of all arcs (u, w) with $w \in \Gamma^+(u)$ and compare it to $r_\rho(u, w)$. Theorem 29 gives the time complexity of such operation where $c(a, b)$ is the time complexity to divide an integer a by an integer b.

▶ **Theorem 29** (Theorem 43 in [1]). *For any vertex $u \in V_0$, given the return flows of all arcs $(u, v) \in \hat{\mathcal{A}}$ with $v \in \Gamma^+(u)$, one can compute $D(\rho)(u)$ and the flow on each arc (u, v) in time $O(|\Gamma^+(u)| \cdot c(r_{\max}, |\mathcal{A}^+(u)|))$ with r_{max} being the maximum (finite) value of $r_\rho(u, v)$ for all $v \in \Gamma^+(u)$.*

But in the case of simple graphs, we can improve this computation (see Lemma 30).

▶ **Lemma 30** (Lemma 61 in [1]). *If the graph is simple, $D(\rho)(u)$ is the first arc with respect to order θ_u, starting at $\rho(u)$, among all arcs of $\mathcal{A}^+(u)$ with minimal return flow. The flow on all arcs $(u, v) \in \hat{\mathcal{A}}$ can be computed in time $O(|\Gamma^+(u)|)$.*

4 ARRIVAL for Tree-Like Multigraphs

In this section we show that, for a given configuration ρ, we can compute the Destination Forest $D(\rho)$ in time complexity $O(|\mathcal{A}| \cdot c(r_{\max}, |\mathcal{A}|))$, hence solve the ARRIVAL problem for every vertex at the same time. To achieve this, we recursively compute return flows for all arcs in $\hat{\mathcal{A}}$ and then use these flows to compute the destination forest.

▶ **Theorem 31** (Complete Destination Algorithm, Theorem 47 in [1]). *The configuration $D(\rho)$ can be computed in time $O(|\mathcal{A}|)$ for a stopping tree-like rotor multigraph in a model where arithmetic operations can be made in constant time, or alternatively in $O(|\mathcal{A}| \cdot c(r_{\max}, |\mathcal{A}|))$ on a Turing Machine.*

Sketch of the Proof. Consider an arbitrary vertex x. We proceed with a Breadth-First Search (BFS) to compute an order such that all necessary return flows are already computed when we use Theorem 29. The algorithm is split into two phases:

1. Computation of the return flows of arcs directed from x towards leaves, starting from the arcs closest to the leaves and recursively coming back to x, using Theorem 29.
2. Computation of the return flows of all remaining arcs, starting from the arcs closest to x and recursively coming back to the leaves. We use Theorem 29 only twice for each vertex u to compute the return flows of all arcs (u, v) with (u, v) being directed from the leaves towards x.

All in all, we use Theorem 29 three times for each vertex, hence the time complexity is $O(\sum_{u \in V} (|\hat{\mathcal{A}}^+(u)| \cdot c(r_{\max}, |\mathcal{A}|)))$ which amounts to $O(|\mathcal{A}| \cdot c(r_{\max}, |\mathcal{A}|))$. ◀

We showed in Lemma 27 that return flows could be written in at most $O(|\mathcal{A}|)$ bits which gives an upper bound for $c(r_{\max}, |\mathcal{A}|)$ of $k|\mathcal{A}|\log(|\mathcal{A}|)$ for some constant $k > 0$. It is proved in [14] that the multiplication of two n bits integers can be done in time $O(n \log(n))$ and as the complexity of the division is equivalent to the complexity of multiplication (see [5]), the bound follows. Thus the complexity of our algorithm is $O(|\mathcal{A}|^2 \log(|\mathcal{A}|))$ in this context.

Simple Graph

In the case of simple graphs, this algorithm has a better complexity as we can use Lemma 30 instead of Theorem 29. Nevertheless, the algorithm remains the same and the complexity sums up to $O(|A|)$ which is also $O(|V|)$ as the graph is simple (details can be found in Theorem 63 in [1]).

5 One and Two player Variants

Problem ARRIVAL can be seen as a zero-player game where the winning condition is that the particle reaches a particular sink (or set of sinks). The one and two players variants of ARRIVAL (i.e. deterministic analogs of *Markov decision processes* and *Stochastic games*) we address in this section are inspired from [18], but differ by the choice of the set of strategies (see the discussion hereafter).

We specifically consider a game with a single player that controls a subset of vertices $V_{\mathcal{MAX}}$ of V_0. Given a rotor configuration on the rest of the vertices of V_0, a starting vertex and a set of targets among the sinks, his goal is to wisely choose the initial rotor configuration of the vertices he controls (his strategy) such that the particle reaches one of the targets.

Our definition of the way a player controls his vertices is somewhat different from the one in the seminal paper [18], where a strategy consists in choosing an outgoing-arc each time the particle is on a vertex controlled by the player. In this sense the set of strategies that we allow (which seems to us a very natural extension of the zero player case) is a finite subset of the version from [18] and could seem easier, but the latter can in fact be solved by some slight modifications of our algorithm. Moreover, the proof of NP-completeness from [18] is still valid in our case, despite the fact that we reduce the number of available strategies.

▶ **Definition 32** (Partial Configuration). *Let V' be a subset of V_0, a* partial rotor configuration *on V' is a mapping ρ' from V' to $\mathcal{A}$ such that $\rho'(u) \in \mathcal{A}^+(u)$ for all $u \in V'$.*

Given the disjoint sets of vertices V_r, $V_{\mathcal{MAX}}$ and S_0, a one-player rotor game (resp. one-player tree-like rotor game) is defined by $(V_r, V_{\mathcal{MAX}}, S_0, \mathcal{A}, h, t, \theta, \mathrm{val}, \rho)$ where:

- $(V_0, S_0, \mathcal{A}, h, t, \theta)$ is a rotor graph (resp. tree-like rotor graph) with $V_0 = V_r \cup V_{\mathcal{MAX}}$;
- val is a map from S_0 to $\{0, 1\}$ defining the target sinks (sinks of value 1);
- ρ is a partial configuration on V_r, i.e. on the vertices not controlled by the player.

The tree-like rotor game is *stopping* if and only if the induced rotor graph $(V_0, S_0, \mathcal{A}, h, t, \theta)$ is stopping. The player is called $\mathcal{MAX}$, and a *strategy* for $\mathcal{MAX}$ is a partial rotor configuration on $V_{\mathcal{MAX}}$. We denote by $\Sigma_{\mathcal{MAX}}$ the finite set of strategies for this player.

Consider a partial rotor configuration ρ on V_r together with strategy σ and denote by (ρ, σ) the rotor configuration where we apply the partial configuration ρ or σ depending on whether the vertex is in V_r or $V_{\mathcal{MAX}}$. The *value of the game* for strategy σ and starting

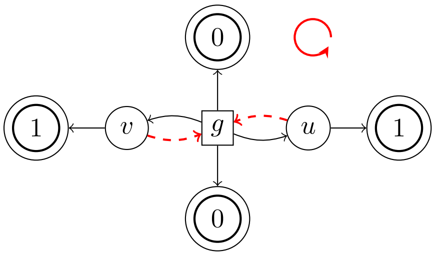


Figure 6 Simple graph where the optimal strategy depends on the starting vertex u_0, with $V_{\mathcal{MAX}} = \{g\}$, with $u, v \in V_0$ and with all other vertices being sinks. As in previous examples, the starting configuration is depicted by red arcs in dashes, and the rotor order on all vertices is an anticlockwise order on their outgoing arcs. In the case $u_0 = v$, the only optimal strategy is $\sigma(g) = (g, v)$ and the game has value 1. In the case $u_0 = u$, the only optimal strategy is $\sigma(g) = (g, u)$ and the game has value 1.

vertex u_0 is denoted by $\mathrm{val}_\sigma(u_0)$ and is equal to $\mathrm{val}(s)$ where s is the sink reached by a maximal rotor walk from the rotor particle configuration $((\rho, \sigma), u_0)$ if any, and 0 otherwise. As in the zero-player framework, up to computing strongly connected components that do not contain sinks and replacing each of them with a sink of value 0, we can suppose that the tree-like rotor game is stopping. In the following, all rotor games we consider are tree-like and stopping unless stated otherwise.

When u_0 is fixed, the maximal value of $\mathrm{val}_\sigma(u_0)$ over strategies $\sigma \in \Sigma_{\mathcal{MAX}}$ is called the *optimal value* of the game with starting vertex u_0 and is denoted by $\mathrm{val}^*(u_0)$. Any strategy $\sigma \in \Sigma_{\mathcal{MAX}}$ such that $\mathrm{val}_\sigma(u_0) = \mathrm{val}^*(u_0)$ is called an *optimal strategy* for the game starting in u_0. Observe that optimal strategies may depend on the choice of u_0 (see Figure 6).

The one-player ARRIVAL problem consists in computing the optimal value of a given starting vertex in a one-player rotor game.

Recall that in the tree-like rotor graph, $T_{(u,v)}$ denotes the (u, v)-subtree. We extend this notation to denote the one-player, not necessarily stopping, game played on the (u, v)-subtree where we restrict $V_{\mathcal{MAX}}$ and V_r to the subtree. For this game, we only consider the case where the starting vertex is u.

▶ **Definition 33** (Value under strategy). *Let (u, v) be an arc of $\hat{\mathcal{A}}$. For a strategy σ on the (u, v)-subtree, we denote by $val_\sigma(u, v)$ (resp. $val^*(u, v)$) the value of the game under strategy σ (resp. under an optimal strategy) in $T_{(u,v)}$. This is called the value (resp. the optimal value) of arc (u, v) for strategy σ.*

▶ **Definition 34** (Optimal return flow r^*). *Let (u, v) be an arc of $\hat{\mathcal{A}}$. If $val^*(u, v) = 0$, then $r^*(u, v)$ is defined as the maximum of $r_\sigma(u, v)$ over all strategies σ on $T_{(u,v)}$, otherwise it is the minimum of $r_{\sigma^*}(u, v)$ among* optimal *strategies σ^* on $T_{(u,v)}$.*

Lemma 35 connects the value $\mathrm{val}_\sigma(u)$ with the value of the last outgoing arc of vertex u while processing a maximal rotor walk from the rotor-particle configuration $((\rho, \sigma), u)$.

▶ **Lemma 35.** *Let a be an arc of $\mathcal{A}^+(u)$ such that $D(\rho, \sigma)(u) = a$ with $h(a) = v$. We have $val_\sigma(u) = val_\sigma(u, v)$.*

To recursively compute an optimal strategy, we need a stronger notion of optimality, namely a *subtree optimal strategy.*

▶ **Definition 36** (Subtree optimal strategy). *A strategy σ^* is* subtree optimal *at u_0 if it is optimal at u_0 and, moreover, $val_{\sigma^*}(u, v) = val^*(u, v)$ and $r_{\sigma^*}(u, v) = r^*(u, v)$ for every (u, v)-subtree such that (u, v) is directed from u_0 towards the leaves.*

Instead of recursively computing only the return flow as in the zero-player game, we now propagate both the optimal value and the optimal return flow to construct a subtree optimal strategy (details in the proof of Theorem 54 in [1]) and give an equivalent of Theorem 31 for the one-player game.

▶ **Theorem 37** (Computation of $\mathrm{val}^*(u_0)$, Theorem 54 in [1])**.** *The optimal value $val^*(u_0)$ can be computed in the same time complexity as the computation of $D(\rho)$ in the zero-player game (see Theorem 31).*

Some remarks are in order here. (i) The algorithm used to compute $\mathrm{val}^*(u_0)$ (Algorithm 3 in [1]) provides the optimal value of u_0 as well as a subtree optimal strategy at u_0 for every decisional vertex. A tutorial example is given in [1] (Figure 11). (ii) This algorithm can also be adapted to compute optimal values for games with more general type of strategies, particularly those in [18]. (iii) In the case of a simple graph, one can compute $\mathrm{val}^*(u_0)$ for every vertex u_0 with the same time complexity as in Theorem 37 as detailed in our extended version [1]. (iv) If we consider a game with integer values on the sinks, one can use a dichotomic process to solve it in $\log(|S|)$ steps where a step consists in solving a binary game.

Two-player Game

In the two-player game, each player controls the initial configuration on disjoint sets of vertices, and the second player wants to minimize the value of the game. In a general graph, there may not exist an equilibrium in pure strategies (see Figure 7 where we define a game equivalent to *matching pennies*). However, in the case of tree-like rotor multigraphs, we can show that there always is an equilibrium in pure strategies and compute it (see [1]). As in the one-player case, if we consider integer values, we can simply proceed using a dichotomy technique.

Conclusion and Future Work

Concerning ARRIVAL, one remaining fundamental question is to determine whether there exists a polynomial algorithm to solve the zero-player game. Similarly, problems such as *simple stochastic games, parity games* and *mean-payoff games* are also in NP ∩ co-NP, and there are no polynomial algorithm known to solve them (see [13]). For those different problems, considering subclasses of graphs where we can find polynomial algorithms is a fruitful approach (see [2] and [3]). This paper is a first step in this direction.

Thus, we would like to study more general classes of graphs. To begin with, even graphs that are well-studied in terms of the *sandpile group* such that ladders or grids remain now an open problem for ARRIVAL. The problem of finding the destination of multiple particles at the same time is also an important open problem in nearly all cases – we solve it for the (simple) path graph in our extended version (see [1]).

Now that the Tree-like multigraph case is settled, it seems natural to try and extend these results to classes of graphs with bounded width (e.g. treewidth, pathwidth, etc.). However, this extension is not direct and requires further work. If we were to replace a single node by a bag of nodes, defining an analog of the rotor ordering, the routine and return flows is much more complicated since a particle can enter and leave the bag in different ways. We need to compute and store all the exit arcs for all entering arcs in the bag for every rotor configuration, and combine these informations for neighbouring bags in order to find locally the destination forest. We intend to finish this work in a near future.

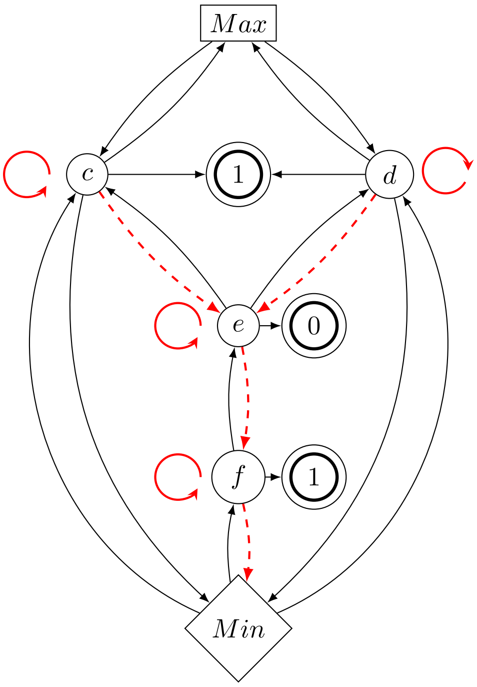


Figure 7 This example is a simple undirected graph where each edge is replaced by two arcs. We have $V_{\mathcal{MAX}} = \{Max\}$, $V_{\mathcal{MIN}} = \{Min\}$ and $V_0 = \{c, d, e, f\}$ and S_0 is the rest of the vertices with their value written inside them. The particle starts on the vertex Max. In this game, the only optimal strategy for $\mathcal{MAX}$ when the strategy for $\mathcal{MIN}$ is the arc (Min, x) with $x \in \{c, d\}$ is (Max, x). On the other hand the only optimal strategy for $\mathcal{MIN}$ when the strategy for $\mathcal{MAX}$ is the arc (Max, c) (resp. (Max, d)) is (Min, d) (resp. (Min, c)). The situation is like the classical matching pennies game where one player tries to match the strategy of the opponent whereas the other player has the opposite objective. It is known that such game does not admit a Nash equilibrium in pure strategies.

References

1 David Auger, Pierre Coucheney, and Loric Duhaze. Polynomial time algorithm for arrival on tree-like multigraphs. *arXiv preprint arXiv:2204.13151*, 2022.

2 David Auger, Pierre Coucheney, and Yann Strozecki. Finding optimal strategies of almost acyclic simple stochastic games. In *International Conference on Theory and Applications of Models of Computation*, pages 67–85. Springer, 2014.

3 David Auger, Pierre Coucheney, and Yann Strozecki. Solving simple stochastic games with few random nodes faster using bland's rule. In *36th International Symposium on Theoretical Aspects of Computer Science*, 2019.

4 Anders Björner, László Lovász, and Peter W Shor. Chip-firing games on graphs. *European Journal of Combinatorics*, 12(4):283–291, 1991.

5 Richard P Brent and Paul Zimmermann. *Modern computer arithmetic*, volume 18. Cambridge University Press, 2010.

6 Swee Hong Chan, Lila Greco, Lionel Levine, and Peter Li. Random walks with local memory. *Journal of Statistical Physics*, 184(1):1–28, 2021.

7 Joshua N Cooper and Joel Spencer. Simulating a random walk with constant error. *Combinatorics, Probability and Computing*, 15(6):815–822, 2006.

8 Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. Arrival: A zero-player graph game in NP ∩ coNP. In *A journey through discrete mathematics*, pages 367–374. Springer, 2017.

9 Tobias Friedrich and Thomas Sauerwald. The cover time of deterministic random walks. In *International Computing and Combinatorics Conference*, pages 130–139. Springer, 2010.

10 Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubácek, Karel Král, Hagar Mosaad, and Veronika Slívová. Arrival: Next stop in cls. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

11 Bernd Gärtner, Sebastian Haslebacher, and Hung P. Hoang. A Subexponential Algorithm for ARRIVAL. In *ICALP 2021*, volume 198, pages 69:1–69:14, 2021.

12 Giuliano Pezzolo Giacaglia, Lionel Levine, James Propp, and Linda Zayas-Palmer. Local-to-global principles for rotor walk. *arXiv preprint arXiv:1107.4442*, 2011.

13 Nir Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007.

14 David Harvey and Joris Van Der Hoeven. Integer multiplication in time O(n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

15 Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuyal Peres, James Propp, and David B. Wilson. *Chip-Firing and Rotor-Routing on Directed Graphs*, pages 331–364. Springer, 2008.

16 AM Povolotsky, VB Priezzhev, and RR Shcherbakov. Dynamics of eulerian walkers. *Physical review E*, 58(5):5449, 1998.

17 Vyatcheslav B Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Physical Review Letters*, 77(25):5079, 1996.

18 Rahul Savani, Matthias Mnich, Martin Gairing, and John Fearnley. Reachability switching games. *Logical Methods in Computer Science*, 17, 2021.

19 Vladimir Yanovski, Israel A Wagner, and Alfred M Bruckstein. A distributed ant algorithm for protect efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.

Graph Realization of Distance Sets

Amotz Bar-Noy ✉
City University of New York (CUNY), NY, USA

David Peleg ✉
Weizmann Institute of Science, Rehovot, Israel

Mor Perry ✉
The Academic College of Tel-Aviv-Yaffo, Israel

Dror Rawitz ✉
Bar Ilan University, Ramat-Gan, Israel

Abstract

The Distance Realization problem is defined as follows. Given an $n \times n$ matrix D of nonnegative integers, interpreted as inter-vertex distances, find an n-vertex weighted or unweighted graph G realizing D, i.e., whose inter-vertex distances satisfy $dist_G(i,j) = D_{i,j}$ for every $1 \le i < j \le n$, or decide that no such realizing graph exists. The problem was studied for general weighted and unweighted graphs, as well as for cases where the realizing graph is restricted to a specific family of graphs (e.g., trees or bipartite graphs). An extension of Distance Realization that was studied in the past is where each entry in the matrix D may contain a *range* of consecutive permissible values. We refer to this extension as Range Distance Realization (or Range-DR). Restricting each range to at most k values yields the problem k-Range Distance Realization (or k-Range-DR). The current paper introduces a new extension of Distance Realization, in which each entry $D_{i,j}$ of the matrix may contain an arbitrary set of acceptable values for the distance between i and j, for every $1 \le i < j \le n$. We refer to this extension as Set Distance Realization (Set-DR), and to the restricted problem where each entry may contain at most k values as k-Set Distance Realization (or k-Set-DR).

We first show that 2-Range-DR is NP-hard for unweighted graphs (implying the same for 2-Set-DR). Next we prove that 2-Set-DR is NP-hard for unweighted and weighted trees. We then explore Set-DR where the realization is restricted to the families of stars, paths, or cycles. For the weighted case, our positive results are that for each of these families there exists a polynomial time algorithm for 2-Set-DR. On the hardness side, we prove that 6-Set-DR is NP-hard for stars and 5-Set-DR is NP-hard for paths and cycles. For the unweighted case, our results are the same, except for the case of unweighted stars, for which k-Set-DR is polynomially solvable for any k.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph Realization, distance realization, network design

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.13

Funding Supported in part by a US-Israel BSF grant (2018043).

1 Introduction

Background. *Network realization* problems are fundamental graph-algorithmic questions in which one is asked to construct a network conforming to some predefined requirements. Given a *specification* (or *information profile*) that consists of constraints on some network parameters, such as the vertex degrees, distances, or connectivity, one is required to construct a network conforming to the given specification, i.e., satisfying the requirements, or to determine that no such network exists. The motivation for network realization problems stems from both "exploratory" contexts where one attempts to reconstruct an *existing* network of unknown structure based on the outcomes of experimental measurements, and engineering contexts related to network design.

In the Distance Realization problem, the given profile is an $n \times n$ matrix D such that each entry $D_{i,j} \in \mathbb{N} \cup \{\infty\}$, for $1 \leq i < j \leq n$, and $D_{i,i} = \{0\}$, for every $1 \leq i \leq n$. We view $D_{i,j}$ as specifying the required distance between the vertices i and j in the network. A graph $G = (V, E)$ is a *realization* of D if $dist_G(i,j) = D_{i,j}$, for every $1 \leq i < j \leq n$, where $dist_G(i,j)$ denotes the distance between i and j in G. Generally, we may be interested in two types of realizing graphs. In unweighted Distance Realization it is assumed that each edge of the realizing graph is of length 1. In weighted Distance Realization the edges of the realizing graph may have any positive integral lengths.

Observe that an unweighted realizing graph is fully determined by D: the edge (i,j) exists in the graph if and only if $D_{i,j} = 1$. It follows that there is only one graph G_D that may serve as a candidate realizing graph. This was observed by Hakimi and Yau [12], who provided a characterization for distance realization by unweighted graphs, implying also a polynomial-time algorithm for unweighted Distance Realization. Notice also that in the case where the realization is required to be a specific graph H, one can solve unweighted Distance Realization by deciding whether H and G_D are isomorphic. This Graph Isomorphism problem is computationally easy when H belongs to certain graph types, such as stars, paths, and cycles, and therefore the problem of distance realization by such graphs can be solved in polynomial time.

Hakimi and Yau [12] also studied weighted Distance Realization. They proved that the necessary and sufficient condition for the realizability of a given martix D is that D is a metric. Furthermore, they gave a polynomial-time algorithm that computes a realization for any given metric distance matrix. More specifically, their algorithm constructs a minimum-edge realizing graph whose edges are necessary in every realization of D.

Patrinos and Hakimi [14] considered the case where weights can be negative. They showed that any symmetric matrix (with zero diagonal) is a distance matrix of some graph G. They gave necessary and sufficient conditions for realizing such a matrix by a tree, and they showed that if a tree realization exists it is unique. Distance Realization in *weighted trees* was considered in [2], which presented a characterization for realizability. For unweighted trees, there is a straightforward realization algorithm, based on the algorithm of [12] for general unweighted graphs, and on the fact that the realization, if exists, is unique. Distance realization restricted to bipartite graphs was studied in [4], where it was observed that it is sufficient to check the unique realization in the unweighted case or the minimal realization in the weighted case.

A natural extension of Distance Realization is when each entry in the distance matrix may contain a *range* of consecutive values instead of a single value. Range specifications may arise, for example, when D reflects the properties of an unknown network, and its values are obtained by imprecise measurements, or alternatively, when D represents a design specification for a planned network in a setting where distance constraints are not rigid and allow some flexibility. Formally, we are given two values $D^-_{i,j}$ and $D^+_{i,j}$ for every i, j and the realizing G must satisfy $D^-_{i,j} \leq dist_G(i,j) \leq D^+_{i,j}$. We refer to this extended version of the problem as Range Distance Realization (or Range-DR).

Tamura et al. [18] obtained necessary and sufficient conditions for the realizability of a range distance matrix by weighted graphs, generalizing the result of [12] from precise to range specifications. A polynomial-time algorithm for weighted Range-DR was given in [15]. The unweighted version of Range-DR was shown to be NP-hard in [4], where it was also shown that if the realizing graph is required to be a tree, then both the unweighted and weighted versions of Range-DR are NP-hard.

Realization with Distance Sets. In this paper we introduce a novel extension of Range Distance Realization, called Set Distance Realization (Set-DR). Instead of a *range*, we assume that each entry $D_{i,j}$ in the distance matrix specifies a *set* of acceptable values for the distance between i and j, for every $1 \le i < j \le n$. More formally, consider an $n \times n$ matrix D, such that each entry $D_{i,j} \subseteq \mathbb{N} \cup \{\infty\}$, for $1 \le i < j \le n$, is a non-empty set, and $D_{i,i} = \{0\}$, for every $1 \le i \le n$. We view $D_{i,j}$ as specifying a list of acceptable values of the distance between i and j, where i and j are vertices in some network. A graph $G = (V, E)$ is a *realization* of the D if $dist_G(i, j) \in D_{i,j}$, for every $1 \le i < j \le n$.

One of the main questions studied in this paper involves the effect of limiting the number of values in each entry of the matrix D. This question is equally interesting for Set-DR and Range-DR. Given an integer k, we say that the matrix D is a k*-set* distance matrix if $|D_{i,j}| \le k$ for every $1 \le i < j \le n$. A distance matrix D is a k*-range* distance matrix, if $D_{i,j}$ is a range that contains at most k consecutive values for every $1 \le i < j \le n$. A 1-set distance profile is called *precise*. Restricting the Set Distance Realization problem to k-set distance matrices yields the problem k-Set Distance Realization (or k-Set-DR). Similarly, restricting the Range Distance Realization problem to k-range distance matrices yields the problem k-Range Distance Realization (or k-Range-DR).

Henceforth, we assume that entry $D_{i,j}$ in a 2-set distance matrix D consists of two integers, $D_{i,j} = \left\{d^0_{i,j}, d^1_{i,j}\right\}$, which need not be distinct.

Our Results. In this paper we study the computational complexity of k-Set-DR and k-Range-DR, as a function of k, in various graph families.

Inspecting the proof given in [4] for the hardness results for Range-DR by trees and unweighted graphs reveals that 3-Range-DR is already NP-hard over these graph families, implying that 3-Set-DR is NP-hard as well. We modify the reductions from [4] to show that already 2-Range-DR is NP-hard for general unweighted graphs, where precise realization is known to be polynomial [12]. For general weighted graphs, it is known that Range-DR is computationally easy [15]. We note that the algorithm from [15] does not work for Set-DR, since it relies on the continuity of the given ranges. In fact, Set-DR for general weighted graphs remains an open problem. We show that both unweighted 2-Set-DR and weighted 2-Set-DR are NP-hard for trees. Thus, we obtain a dichotomy between 2-set distance realization and precise realization for trees, since precise realization is known to be solvable in polynomial time [12, 2].

Next, we show that 2-Set-DR is polynomial time solvable for stars, paths, and cycles. Our realization algorithms are based on a reduction to the 2-SAT problem (satisfiability of a 2-CNF formula), which can be solved in linear time [10]. The idea is to use one vertex i_0 as a point of reference for all other vertices. Thus, a Boolean variable b_j is associated with each vertex j and determines which of the two values of $D_{i_0,j}$ should be used. The 2-CNF formula is constructed according to the rest of the entries of D. Applying this approach for stars is rather straightforward, but it becomes more complicated for paths, and especially for cycles. In addition, we prove that there exists a polynomial time algorithm for k-Set-DR on unweighted stars, for any k. For weighted stars, we present a polynomial time realization algorithm for Range-DR when the range values are polynomially bounded. This algorithm is based on a reduction to the feasibility problem of linear integer programs with at most two variables per constraint, which can be solved efficiently [5].

On the hardness side, we show that Set-DR is NP-hard for weighted stars already with 6-set distance profiles. We obtain slightly tighter results for paths and cycles, for which both unweighted Set-DR and weighted Set-DR are already NP-hard already with 5-set

Table 1 Results for realization with unweighted graphs.

Graph family	Range-DR	Set-DR
General	2-Range-DR is NP-hard (Thm 1) 1-Range-DR is polynomial [12]	2-Set-DR is NP-hard (Thm 1) 1-Set-DR is polynomial [12]
Tree	3-Range-DR is NP-hard [4] 1-Range-DR is polynomial [12]	2-Set-DR is NP-hard (Thm 2) 1-Set-DR is polynomial [12]
Star	Range-DR is polynomial (Thm 4)	Set-DR is polynomial (Thm 4)
Path	2-Range-DR is polynomial (Thm 8) Range-DR is NP-hard (Thm 9)	2-Set-DR is polynomial (Thm 8) 5-Set-DR is NP-hard (Thm 10)
Cycle	2-Range-DR is polynomial (Thm 12) Range-DR is NP-hard (Thm 14)	2-Set-DR is polynomial (Thm 12) 5-Set-DR is NP-hard (Thm 15)

Table 2 Results for realization with weighted graphs.

Graph family	Range-DR	Set-DR
General	Range-DR is polynomial [15]	Open problem
Tree	3-Range-DR is NP-hard [4] 1-Range-DR is polynomial [2]	2-Set-DR is NP-hard (Thm 2) 1-Set-DR is polynomial [2]
Star	Range-DR is polynomial[1](Thm 6)	2-Set-DR is polynomial (Thm 3) 6-Set-DR is NP-hard (Thm 5)
Path	2-Range-DR is polynomial (Thm 7) Range-DR is NP-hard (Thm 9)	2-Set-DR is polynomial (Thm 7) 5-Set-DR is NP-hard (Thm 10)
Cycle	2-Range-DR is polynomial (Thm 13) Range-DR is NP-hard (Thm 14)	2-Set-DR is polynomial (Thm 13) 5-Set-DR is NP-hard (Thm 15)

distance profiles. Our hardness results are based on reductions from the 3-colorability problem. However, the reductions are not similar. Specifically, in the case of weighted stars, the possible colors of a vertex are encoded in the distance matrix by possible edge weights, while in the case of weighted and unweighted paths, the colors are encoded by vertices and their location on the path. The hardness result for 5-Set-DR on the cycle is obtained by a reduction from 5-Set-DR on the path.

Tables 1 and 2 summarize our results.

Related Work. An optimization variant of Distance Realization problem was introduced in [12]. In this problem, a distance matrix D is given over a set S of n vertices, and the goal is to find a graph G including S, with possibly *auxiliary* vertices, that realizes the given distance matrix for S. Necessary and sufficient conditions are given for a matrix to be realizable by a weighted or an unweighted graph. It is shown in [9] that an optimal realization can have at most n^4 vertices, and therefore, there is a finite (but exponential) time algorithm to find an optimal realization. In [1] it is shown that finding optimal realizations of distance matrices with integral entries is NP-complete, and evidence to the difficulties in approximating the optimal realization is provided in [6]. Over the years, various heuristics for optimal realizations were considered [13, 16, 17, 19]. Since optimal realization seems hard even to approximate, special cases and other variations have been studied [6, 11].

[1] This result requires that the entries of D are polynomially bounded.

Special attention has been given to the optimal distance realization problem where the realizing graph is a tree. In [12], a procedure is given for finding a tree realization of D if exists. It is also shown therein that a tree realization, if exists, is unique and is the optimum realization of D. Necessary and sufficient conditions for a distance matrix to be realizable by a tree were given in several papers [3, 8, 17]. Finally, an $O(n^2)$ time algorithm for optimal tree-realization is described in [7].

2 Realizations by Trees and Unweighted Graphs

In this section we consider unweighted realizations in general graphs and both unweighted and weighted realizations in trees.

2.1 Realizations by Unweighted Graphs

For general graphs, recall that the range realization problem in weighted graphs and the precise realization problem in unweighted graphs both have a polynomial time algorithm. We provide an NP-hardness result for Range-DR by unweighted graphs, even for 2-range distance profiles.

▶ **Theorem 1.** *2-Range-DR is NP-hard in unweighted graphs.*

Proof. We prove the theorem using a reduction from the 3-coloring problem.

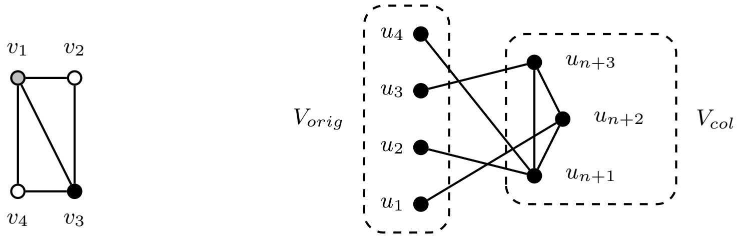
Consider an instance G of the 3-coloring problem. We construct a 2-range distance matrix D for $n+3$ vertices, i.e., for the vertex set $\{u_1, \ldots, u_{n+3}\}$. Intuitively, we think of the first n vertices, $V_{orig} = \{u_1, \ldots, u_n\}$, as representing the *original* vertices of the given graph G, and of additional 3 vertices of D, $V_{col} = \{u_{n+1}, u_{n+2}, u_{n+3}\}$, as representing the three *colors*. Let

$$D_{i,j} = \begin{cases} \{1\} & i = n+1, \ldots, n+3, \quad j = n+1, \ldots, n+3, \\ \{1,2\} & i = 1, \ldots, n, \quad j = n+1, \ldots, n+3, \\ \{2,3\} & 1 \le i < j \le n, \quad (v_i, v_j) \notin E(G), \\ \{3\} & 1 \le i < j \le n, \quad (v_i, v_j) \in E(G). \end{cases}$$

We now prove that the input G is 3-colorable if and only if D is realizable by an unweighted graph. (See Figure 1 for an illustration.)

Suppose G is 3-colorable. Let $\chi : V(G) \mapsto \{1, \ldots, 3\}$ be the coloring function. For the matrix D defined from G, construct a realizing graph $\mathcal{G}$ as follows. Start with a triangle containing the color vertices $u_{n+1}, u_{n+2}, u_{n+3}$. Connect each original vertex u_i to the color vertex $u_{n+\chi(i)}$. It is easy to verify that $\mathcal{G}$ realizes D (see Figure 1b for an example).

Suppose there exists an unweighted graph $\mathcal{G}$ which realizes the matrix D. Consider two original vertices u_i and u_j. Since $1 \notin D_{i,j}$, it follows that u_i and u_j are not connected by an edge. Therefore, every original vertex u_i must be connected to at least one of the color vertices. Define a coloring function for G as follows. For every original vertex u_i, let $n+c$ be some color vertex connected to u_i, and let $\chi(v_i) = c$. Since $1, 2 \notin D_{i,j}$, if two vertices v_i and v_j are connected by an edge in G, then their distance in $\mathcal{G}$ must be at least 3. This ensures that none of the color vertices are connected to both u_i and u_j (as this would make their distance 2). It follows that if $(v_i, v_j) \in E(G)$, then v_i and v_j are assigned different colors. ◀



(a) A 3-colorable graph. **(b)** A realization of D.

Figure 1 An example of the reduction in the proofs of Theorem 1 for $n = 4$. White, gray, and black correspond to nodes $n + 1$, $n + 2$, and $n + 3$, respectively.

2.2 Tree Realizations

Next, we show that distance realization in unweighted or weighted trees is hard even for 2-set distance profiles. The reduction we use is almost identical to a reduction from [4] that (implicitly) shows hardness for 3-range distance profiles. The proof is ommited for lack of space.

▶ **Theorem 2.** *2-Set-DR is NP-hard for both unweighted trees and weighted trees.*

3 Star Realizations

In this section we study Set-DR in stars. We show that there exists a polynomial time algorithm that solves weighted 2-Set-DR in stars. On the other hand, we show that 5-Set-DR on weighted stars is NP-hard. Furthermore, we present a polynomial time algorithm that solves k-Range-DR on weighted stars, for any k, provided that matrix entries are polynomially bounded.

To put these results in context, it may be useful to observe that the *unweighted* case in stars is easier: unweighted k-Set-DR in stars can be solved in polynomial time for any k.

3.1 2-Set-DR on Stars is Easy

We show that the 2-Set-DR problem in stars can be solved efficiently.

▶ **Theorem 3.** *There exists a polynomial time algorithm for 2-Set-DR on stars.*

Proof. Assume that i is the center of the star. It follows that the weight of any edge (i, j), for $j \neq i$ can be either $d^0_{i,j}$ or $d^1_{i,j}$. Define a Boolean variable x_j, where $x_j = \text{FALSE}$, if the weight of the edge (i, j) is $d^0_{i,j}$, and $x_j = \text{TRUE}$, if the weight of the edge (i, j) is $d^1_{i,j}$. The rest of the entries of D are used to create a 2-CNF formula that is satisfiable if and only if there exists a star realization of D in which i is the center.

Consider two vertices $j, k \neq i$. Since there are two possible weights for the edges (i, j) and (i, k), it follows that there are four possible distances from j to k:

1. $d^0_{i,j} + d^0_{i,k}$,
2. $d^0_{i,j} + d^1_{i,k}$,
3. $d^1_{i,j} + d^0_{i,k}$, and
4. $d^1_{i,j} + d^1_{i,k}$.

For each one of the above four options, check whether it equals $d^0_{j,k}$ or $d^1_{j,k}$. This induces a truth table on the variable x_j and x_k that can be represented by at most two 2-CNF clauses.[2] Doing this for all pairs of vertices creates a 2-CNF formula that contains at most $O(n^2)$ clauses by concatenating all the above mentioned clauses.

Suppose that there exists a star realization P of D with i as a center. This induces an assignment to the variables Boolean variables that satisfies the 2-CNF formula. On the other hand, assume that the 2-CNF formula that is obtained by assuming that i is the center is satisfiable. A satisfying assignment induces a star, which complies with the profile.

Since there are n candidates for the center vertex, we need to run the above process n times. It follows that the total running time of the algorithm is $O(n^3)$. ◀

We remark that the running time for unweighted case can be improved to $O(n^2)$.

▶ **Theorem 4.** *There exists a polynomial time algorithm for k-SET-DR on unweighted stars, for any k.*

Proof. Since all distances in an unweighted star are either 1 or 2, one may assume that $D_{i,j} \subseteq \{1, 2\}$, for every $i \neq j$. The theorem follows due to Theorem 3. ◀

3.2 6-Set-DR on Weighted Stars is Hard

We show that the star realization problem is NP-hard even on 6-set distance profiles.

▶ **Theorem 5.** *6-SET-DR is NP-hard in weighted stars.*

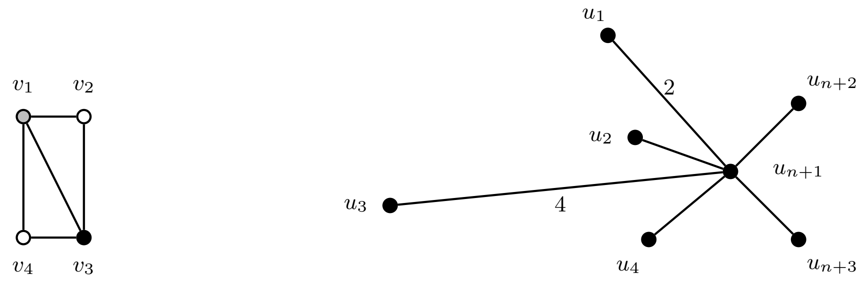
Proof. We prove the lemma using a reduction from the 3-COLORING problem.

Consider a graph G, where $V(G) = \{v_1, \ldots, v_n\}$. We construct a distance matrix D on $n + 3$ vertices, denoted by $\{u_1, \ldots, u_{n+3}\}$. Informally, the distance matrix is defined to force u_{n+1} to be the center of the star while u_{n+2} and u_{n+3} must be two of the leaves whose distance from the center is 1. The rest of the vertices, $u_1, \ldots, u_n$ that are associated with the n vertices $v_1, \ldots, v_n$ of G, are leaves whose distance from the center is either 1, 2, or 4. The idea is that distance 2^c is associated with the color c (0, 1, or 2). Finally, the distances between these n leaves is defined to guarantee that the two endpoints of any edge of G are associated with different colors, if a realization exists. More formally, the 6-set distance matrix D is defined as follows for any two indices $1 \leq k < \ell \leq n + 3$:

$$D_{k,\ell} = \begin{cases} \{3,5,6\} & k, \ell \leq n, (v_k, v_\ell) \in E(G), \\ \{2,3,4,5,6,8\} & k, \ell \leq n, (v_k, v_\ell) \notin E(G), \\ \{1,2,4\} & k \leq n, \ell = n+1, \\ \{2,3,5\} & k \leq n, \ell = n+2, n+3, \\ \{1\} & k = n+1, \ell = n+2, n+3, \\ \{2\} & k = n+2, \ell = n+3 \ . \end{cases}$$

We show that G is 3-colorable if and only if D is realizable by weighted star.

[2] For example, let $D_{i,j} = \{2, 3\}$, $D_{i,k} = \{3, 4\}$, and $D_{j,k} = \{5, 7\}$. There are two possible weight assignments: either $w(i, j) = d^0_{i,j}$ and $w(i, k) = d^0_{i,k}$ or $w(i, j) = d^1_{i,j}$ and $w(i, k) = d^1_{i,k}$ This can be represented by the clause $(\neg x_j \vee \neg x_k) \wedge (x_j \vee x_k)$.



(a) A 3-colorable graph. **(b)** A realization of D.

Figure 2 An example of the reduction in the proof of Theorem 5 for $n = 4$. White, gray, and black correspond to the weights of the edges of the star. Edges without a label are of weight 1.

Assume that G is 3-colorable and $\chi : V \mapsto \{0, 1, 2\}$ is a 3-coloring of G. We describe a star realization S. First, let u_{n+1} be the center of the star. Vertices u_{n+2} and u_{n+3} are leaves such that $w(u_{n+1}, w_{n+2}) = w(u_{n+1}, w_{n+2}) = 1$. Next, for every $i \in \{1, \dots, n\}$, if $\chi(v_i) = c$, then $w(u_{n+1}, u_i) = 2^{\chi(v_i)}$. It is straightforward to verify that S realizes D. In particular, we observe that the first requirement of D is satisfied, since χ is a 3-coloring.

For the other direction, suppose that S is a star realization of D. First, notice that the above distance matrix makes sure that $u_{n+2}, u_{n+1}, u_{n+3}$ form a path with two edges of weight 1. Hence, u_{n+1} must be the center of the star. We define a coloring χ of $V(G)$ according to the weights of the edges to the center: $\chi(v_i) = \log_2 w(u_{n+1}, u_i)$. χ is a proper 3-coloring, since the first requirement of D ensures that $w(u_{n+1}, u_i) \neq w(u_{n+1}, u_j)$ if $(v_i, v_j) \in E$. This is because $2 = 1 + 1$, $4 = 2 + 2$, and $8 = 4 + 4$ are not members of $\{3, 5, 6\}$ which are the possible distances between u_i and u_j. (See Figure 2.) ◀

3.3 Range-DR on Weighted Stars

In [4] it was shown that there exists a polynomial time algorithm that solves the Range-DR problem on a given fixed weighted tree, assuming that non-integral edges weights are allowed. Let D be a range distance matrix, where $D_{ij} = \left\{D^-_{ij}, \dots, D^+_{ij}\right\}$. Also, let $T = (V, E_T)$ be a known tree, and let $P_{k,\ell} = (k = i_0, i_1, \dots, i_{t_{k,\ell}} = \ell)$ be the unique path from vertex k to vertex ℓ in T. For each edge $(i, j) \in E_T$, let $w_{i,j}$ be the variable that denotes the weight of this edge. Consider the following linear program:

$$\sum_{j=0}^{j=t_{k,\ell}-1} w_{i_j,i_{j+1}} \geq D^-_{k,\ell} \qquad \forall P_{k,\ell}$$

$$\sum_{j=0}^{j=t_{k,\ell}-1} w_{i_j,i_{j+1}} \leq D^+_{k,\ell} \qquad \forall P_{k,\ell}$$

The algorithm from [4] finds a realization by obtaining a feasible solution to the above program.

As mentioned above this approach may obtain a realization with non-integral edge weights and distances. Moreover, it may be the case that there exists a realization with non-integral edge weights, while a realization with integral edge lengths does not exist. For example, consider the following distance matrix:

$$D = \begin{pmatrix} \{0\} & \{3\} & \{3\} & \{1,2\} \\ \{3\} & \{0\} & \{3\} & \{1,2\} \\ \{3\} & \{3\} & \{0\} & \{1,2\} \\ \{1,2\} & \{1,2\} & \{1,2\} & \{0\} \end{pmatrix}$$

D admits no star realization with integral weights, but if one allows edge lengths and distances of 1.5, then a realization exists (where vertex v_4 is the center, and all edge weights are 1.5).

We show that the realization problem with integral edge weights is solvable on stars, assuming that the entries of D are polynomially bounded.

▶ **Theorem 6.** *Assume that the entries of D are polynomially bounded. Then there exists a polynomial time algorithm for* RANGE-DR *on weighted stars.*

Proof. Fix the center of the star, and consider the above LP. Since all paths are of length one or two, the resulting integer linear program contains at most two variables per inequality. Since we require integral weights, we add the following integrality constraints: $w_i \in \mathbb{N}$, for every i. The feasibility problem of integer programs with at most two variables per constraint is solvable in $O(mU)$, where m is the number of constraints and U is the range of the variables [5]. In this case $U = \max_{i,j}(D_{ij})$. ◀

Notice that the above mentioned NP-hardness for 6-SET-DR applies to profiles in which each entry is composed of at most 6 constants, which means that it is polynomially bounded.

4 Path Realizations

In this section we study the realization of distance profiles by paths. We first show that if each entry of the distance matrix consists of at most two different values then a realization by a path (if exists) can be found by a polynomial time algorithm. On the other hand, we show that SET-DR on paths is NP-hard, even on 5-set distance profiles. Both results hold both for weighted and unweighted paths.

4.1 Algorithm for 2-Set-DR on Paths

A path P realizes the 2-set distance profile D if either $dist_P(i,j) = d^1_{i,j}$ or $dist_P(i,j) = d^2_{i,j}$ holds for every i and j.

We show that 2-SET-DR on weighted paths can be solved by a reduction to 2-CNF satisfiability, which can be solved in $O(m)$ time [10], where m is the number of clauses.

▶ **Theorem 7.** *There exists a polynomial time algorithm for 2-*SET*-DR on weighted paths.*

Proof. Assume that the weighted path is embedded on the real line where the vertices are located at integers and the distance between any two vertices is their distance on the line. Furthermore, assume that the leftmost vertex is vertex i and it is located at 0.

Any realization of D in which i is the left-most node, implies that vertex $j \neq i$ is located either at $d^0_{i,j}$ or at $d^1_{i,j}$. Define a Boolean variable x_j, where $x_j =$ FALSE represents that j is located at $d^0_{i,j}$ and $x_j =$ TRUE represents that j is located at $d^1_{i,j}$. The rest of the entries of D will create a 2-CNF formula that is satisfiable if and only if there exists a realization of D on the path in which i is the left-most vertex.

Consider the possible placements of two vertices $j, k \neq i$. Since each one of them has two possible placements, it follows that there are four possible placements of j and k:

1. j at $d^0_{i,j}$ and k at $d^0_{i,k}$,
2. j at $d^0_{i,j}$ and k at $d^1_{i,k}$,
3. j at $d^1_{i,j}$ and k at $d^0_{i,k}$, and
4. j at $d^1_{i,j}$ and k at $d^1_{i,k}$.

For each one of the above four options, check whether it complies with $d^0_{j,k}$ or with $d^1_{j,k}$. This induces a truth table on the variable x_j and x_k that can be represented by at most two 2-CNF clauses. Doing this for all pairs of vertices creates a 2-CNF formula that contains at most $O(n^2)$ clauses by concatenating all the above mentioned clauses.

If there exists a path realization P of the profile, then embed it on the real line such that the left-most vertex i of P is located at 0. Then, consider the formula that we get by assuming that i is the left-most vertex of P, and assign values to the Boolean variables according to the distances from i. Since P realizes the profile, the 2-CNF formula must be satisfied. On the other hand, assume that the 2-CNF formula that is obtained by assuming that i is placed on 0 is satisfiable. A satisfying assignment induces a placement of the vertices on the real line, and this implies a realization of the profile.

Since there are n candidates for the left-most vertex we need to run the above process n times. It follows that the total running time of the algorithm is $O(n^3)$. ◀

The same proof works for the unweighted case. One only need to notice that in this case all the distances are integers in the range $\{1, \ldots, n-1\}$ and therefore the placement of the vertices is a bijection from $\{1, \ldots, n\}$ to $\{0, \ldots, n-1\}$.

▶ **Theorem 8.** *There exists a polynomial time algorithm for 2-Set-DR on unweighted paths.*

4.2 Hardness Result

Range-DR in weighted paths was shown to be NP-hard in [4] using a reduction from the Linear Arrangement problem. This result also applies to unweighted paths. It is important to note that the reduction constructs a matrix with unlimited ranges.

We start the section with an alternative and simpler proof that also requires unlimited ranges.

▶ **Theorem 9.** *Range-DR is NP-hard in both weighted and unweighted paths.*

Proof. We prove the theorem using a reduction from Hamiltonian Path. Given a graph G, construct the following distance matrix:

$$D_{i,j} = \begin{cases} \{1, \ldots, n-1\} & (v_i, v_j) \in E(G) \ , \\ \{2, \ldots, n-1\} & (v_i, v_j) \notin E(G) \ . \end{cases}$$

If G has a Hamiltonian path, then this path induces a realization of D. On the other hand, a realization of D corresponds to a Hamiltonian path in G. ◀

Next we show that the Set-DR problem on paths is NP-hard even on 5-set distance profiles.

▶ **Theorem 10.** *5-Set-DR is NP-hard in unweighted and weighted paths.*

Proof. We prove the lemma using a reduction from the 3-coloring problem.

Consider a graph G, where $V(G) = \{v_1, \ldots, v_n\}$. We construct a distance matrix D on $3n+2$ vertices, denoted $\{u_0, \ldots, u_{3n+1}\}$. Intuitively, the vertices u_{3i-2}, u_{3i-1} and u_{3i} represent vertex v_i in the original graph, and the location of u_{3i-2} encoded the color of v_i.

The vertices u_0 and u_{3n+1} are the end-points of the path, and they serve as preference points. More formally, for every x, we define $\bar{x} \triangleq \lceil x/3 \rceil$. The matrix is defined as follows for any two indices $0 \le k < \ell \le 3n+1$:

$$D_{k,\ell} = \begin{cases} \{3n+1\} & k=0, \ell=3n+1, \\ \{3\bar{\ell}-2, 3\bar{\ell}-1, 3\bar{\ell}\} & k=0, \bar{\ell} \in \{1,\dots,n\}, \\ \{3n-3\bar{k}+1, 3n-3\bar{k}+2, 3n-3\bar{k}+3\} & \bar{k} \in \{1,\dots,n\}, \ell = 3n+1, \\ \{1,2\} & \bar{k} = \bar{\ell}, \\ \{3(\bar{\ell}-\bar{k})+\Delta : \Delta \in \{-2,-1,0,1,2\}\} & \bar{k} < \bar{\ell}, \\ & k \bmod 3 \neq 1 \text{ or } \ell \bmod 3 \neq 1 \\ & \text{ or } (v_{\bar{k}}, v_{\bar{\ell}}) \notin E(G), \\ \{3(\bar{\ell}-\bar{k})+\Delta : \Delta \in \{-2,-1,1,2\}\} & \bar{k} < \bar{\ell}, \\ & k, \ell \bmod 3 = 1, (v_{\bar{k}}, v_{\bar{\ell}}) \in E(G) . \end{cases}$$

Observe that D is a 5-set distance matrix. Also, notice that only the last requirement is not a range, and it consists of a union of two ranges of size 2.

We show that G is 3-colorable if and only if D is realizable using an unweighted path.

Assume that G is 3-colorable and $\chi : V \mapsto \{0,1,2\}$ is a 3-coloring of G. We describe a path realization as a placement of the vertices on integral points from 0 to $3n+1$. First, u_0 is placed on 0 and u_{3n+1} is placed on $3n+1$. Next, for every $i \in \{1,\dots,n\}$, if $\chi(v_i) = c$, then u_{3i-2} is placed at location $3i-2+c$. The vertices u_{3i-1} and u_{3i} are placed at the two remaining free locations from $\{3i-2, 3i-1, 3i\}$. It is straightforward to verify that P realizes D. We observe that the last requirement of D is satisfied, since χ is a 3-coloring. See example in Figure 3.

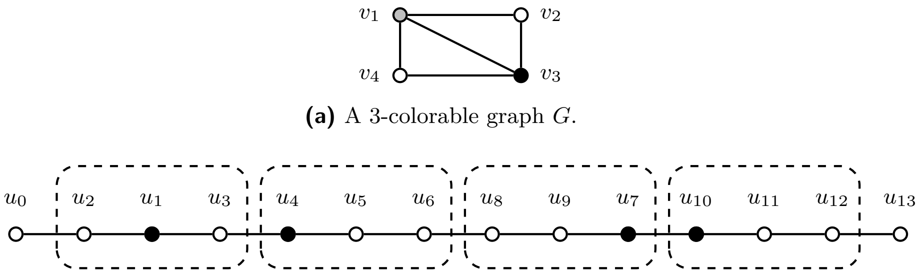
Now suppose that P is a path realization of D. First, notice that the above distance matrix makes sure that the distance between u_0 and u_{3n+1} must be $3n+1$. Moreover, all the other distances are strictly less than $3n+1$. Hence, if a path realization exists, then we may assume without loss of generality that u_0 is placed on 0 and u_{3n+1} is placed on $3n+1$. In the weighted case, it follows that all other vertices are located at $\{1,\dots,3n\}$, which means that all edges are of unit length. Since the distances between u_{3i-2}, u_{3i-1} and u_{3i} are 1 or 2, for every $i \in \{1,\dots,n\}$, these three nodes are forced to appear as a sub-path of P consisting of two edges. Moreover, the required distances from u_0 and u_{3n+1} forces u_{3i-2}, u_{3i-1} and u_{3i} to be assigned to the three consecutive positions $3i-2, 3i-1, 3i$ on the path. We define a coloring χ according to the positions of $\{u_{3i-2} : i \in \{1,\dots,n\}\}$. More specifically, $\chi(v_i) = c$ if u_{3i-2} is located at $3i-2+c$. χ is a 3-coloring, since the last requirement of D ensures that $\chi(v_i) \neq \chi(v_j)$ if $(v_i, v_j) \in E$. ◀

The above proof implies an even stronger result, that we will need in the sequel for the hardness of 5-Set-DR in cycles.

▶ **Theorem 11.** *5-Set-DR is NP-hard in unweighted and weighted paths, even when the required end-points of the path are given in the input.*

5 Cycle Realizations

As was the case with the path, we first show that for 2-set distance profiles a realization by a cycle, if exists, can be found with a polynomial time algorithm. On the other hand, we show that the Set-DR problem on cycles is NP-hard even on 5-set distance profiles, by a reduction from the path realization problem.



(a) A 3-colorable graph G.

(b) A realization of D.

Figure 3 An example of the reduction in the proof of Theorem 10. White, gray, and black are colors 0, 1, and 2, respectively. The location of full nodes correspond to the chosen colors.

5.1 Realization of 2-set distance Profiles by Cycles

A cycle C realizes the 2-set distance profile D if either $dist_C(i,j) = d^1_{i,j}$ or $dist_C(i,j) = d^2_{i,j}$ holds for every i and j.

The proofs of theorems 12 and 13 are omitted due to the page restriction.

▶ **Theorem 12.** *There exists a polynomial time algorithm for 2-Set-DR on unweighted cycles.*

▶ **Theorem 13.** *There exists a polynomial time algorithm for 2-Set-DR on weighted cycles.*

5.2 Hardness Result

We start the section with a hardness proof that requires unlimited ranges.

▶ **Theorem 14.** *Range-DR is NP-hard in both weighted and unweighted cycles.*

Proof. We prove the theorem using a reduction from Hamiltonian Cycle. Given a graph G, construct the following distance matrix:

$$D_{i,j} = \begin{cases} \{1, \dots, \lfloor \frac{n}{2} \rfloor\} & (v_i, v_j) \in E(G) \ , \\ \{2, \dots, \lfloor \frac{n}{2} \rfloor\} & (v_i, v_j) \notin E(G) \ . \end{cases}$$

If G has a Hamiltonian cycle, then this cycle induces a realization of D. On the other hand, a realization of D corresponds to a Hamiltonian cycle in G. ◀

Next we show that Set-DR on cycles is NP-hard even on 5-set distance profiles using reductions from the problem on paths, where the required end-points of the paths are given in the input.

▶ **Theorem 15.** *5-Set-DR is NP-hard in unweighted and weighted cycles.*

Proof. We use a reduction from the Set-DR in unweighted paths, where D is a 5-set distance matrix and the required end-points of the path are given in the input, which was shown to be NP-hard in Theorem 11.

Intuitively, we add $3n$ vertices $n+1, \dots, 4n$, unit weight edges between i and $i+1$, for $i \in \{n, \dots, 4n-1\}$, and the unit weight edge $(4n, 1)$. Formally, given a matrix $D \in \mathbb{N}^{n \times n}$ we construct a matrix $D' \in \mathbb{N}^{4n \times 4n}$ as follows:

$$D'_{k,\ell} = \begin{cases} D_{k,\ell} & 1 \le k, \ell \le n, \\ \min\{(\ell - k), (4n - \ell + k)\} & n < k < \ell \le 4n, \\ \{\min\{(\ell - \delta - 1), (4n + 1 - \ell + \delta)\} : \delta \in D_{k,1}\} & 1 \le k \le n, n < \ell \le 4n, \end{cases}$$

Note that we assume without loss of generality that $\delta \in D_{k,1}$ if and only if $n - 1 - \delta \in D_{k,n}$.

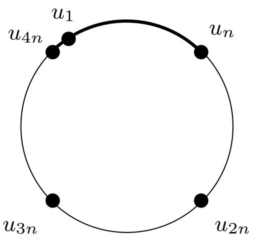


Figure 4 Depiction of the reduction in the proof of Theorem 15, from path realization to cycle realization. The thick line between u_1 and u_n corresponds to the location of the original vertices.

Suppose that D is realizable. In this case, D' is realizable by using a cycle of total length $4n$, where the vertices $u_1, \ldots, u_n$ are placed at positions $1, .., n$ as they are placed in the path realization. Vertex u_i, for $i > n$ is placed at i. (See Figure 4.)

On the other hand, assume that D' is realizable, and assume that u_1 and u_n are placed at locations 1 and n on the cycle. It follows that u_i is located at i, for every $i > n$. In this case, D can be realized by the arc from 1 to n. ◀

6 Summary and Open Problems

This paper introduces the parametric Set Distance Realization (Set-DR) problem, which is an extension of the Range Distance Realization (Range-DR) problem. We study the computational complexity of k-Set-DR and k-Range-DR, as a function of k, in various graph families.

Several questions remain open, including the following.

- Range-DR in weighted general graphs can be solved in polynomial time, but the status of Set-DR is currently unclear.
- For trees, 3-Range-DR and 2-Set-DR are NP-hard, but the status of the 2-Range-DR problem remains unsettled.
- For stars, the hardness of the k-Set-DR problem is unsettled for $k = 3, 4, 5$.
- For paths and cycles the k-Set-DR problem is unsettled for $k = 3, 4$.
- The status of Range-DR for paths and cycles is an open problem.

References

1 Ingo Althöfer. On optimal realizations of finite metric spaces by graphs. *Discret. Comput. Geom.*, 3:103–122, 1988.

2 Agnese Baldisserri. Buneman's theorem for trees with exactly n vertices. *CoRR*, 2014. `arXiv:1407.0048`.

3 Hans-Jürgen Bandelt. Recognition of tree metrics. *SIAM J. Discret. Math.*, 3(1):1–6, 1990.

4 Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Composed degree-distance realizations of graphs. In *32nd IWOCA*, volume 12757 of *LNCS*, pages 63–77. Springer, 2021.

5 Reuven Bar-Yehuda and Dror Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, 29(4):595–609, 2001.

6 Fan R. K. Chung, Mark W. Garrett, Ronald L. Graham, and David Shallcross. Distance realization problems with applications to internet tomography. *J. Comput. Syst. Sci.*, 63(3):432–448, 2001. `doi:10.1006/jcss.2001.1785`.

7 Joseph C. Culberson and Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Inf. Process. Lett.*, 30(4):215–220, 1989.

8 Elias Dahlhaus. Fast parallel recognition of ultrametrics and tree metrics. *SIAM J. Discret. Math.*, 6(4):523–532, 1993.

9 Andreas W. M. Dress. Trees, tight extensions of metric spaces, and the cohomological dimension of certain groups: a note on combinatorial properties of metric spaces. *Adv. in Math.*, 53:321–402, 1984.

10 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

11 Tomás Feder, Adam Meyerson, Rajeev Motwani, Liadan O'Callaghan, and Rina Panigrahy. Representing graph metrics with fewest edges. In *20th STACS*, pages 355–366. Springer, 2003.

12 S. Louis Hakimi and S. S. Yau. Distance matrix of a graph and its realizability. *Quart. Appl. Math.*, 22:305–317, 1965.

13 Juhani Nieminen. Realizing the distance matrix of a graph. *J. Inf. Process. Cybern.*, 12(1/2):29–31, 1976.

14 A. N. Patrinos and S. L. Hakimi. The distance matrix of a graph nand its tree realizability. *Quarterly of Applied Math.*, 255, 1972.

15 Elena Rubei. Weighted graphs with distances in given ranges. *J. Classif.*, 33:282–297, 2016.

16 J. M. S. Simões-Pereira. An optimality criterion for graph embeddings of metrics. *SIAM J. Discret. Math.*, 1(2):223–229, 1988.

17 J. M. S. Simões-Pereira. An algorithm and its role in the study of optimal graph realizations of distance matrices. *Discret. Math.*, 79(3):299–312, 1990.

18 Hiroshi Tamura, Masakazu Sengoku, Shoji Shinoda, and Takeo Abe. Realization of a network from the upper and lower bounds of the distances (or capacities) between vertices. In *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pages 2545–2548. IEEE, 1993.

19 Sacha C. Varone. A constructive algorithm for realizing a distance matrix. *Eur. J. Oper. Res.*, 174:102–111, 2006.

On the Role of the High-Low Partition in Realizing a Degree Sequence by a Bipartite Graph

Amotz Bar-Noy
City University of New York (CUNY), NY, USA

Toni Böhnlein
Weizmann Institute of Science, Rehovot, Israel

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Dror Rawitz
Bar Ilan University, Ramat-Gan, Israel

Abstract

We consider the problem of characterizing degree sequences that can be realized by a bipartite graph. If a partition of the sequence into the two sides of the bipartite graph is given as part of the input, then a complete characterization has been established over 60 years ago. However, the general question, in which a partition and a realizing graph need to be determined, is still open. We investigate the role of an important class of special partitions, called *High-Low partitions*, which separate the degrees of a sequence into two groups, the high degrees and the low degrees. We show that when the High-Low partition exists and satisfies some natural properties, analysing the High-Low partition resolves the bigraphic realization problem. For sequences that are known to be not realizable by a bipartite graph or that are undecided, we provide approximate realizations based on the High-Low partition.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Graph Realization, Bipartite Graphs, Degree Sequences, Graphic Sequences, Bigraphic Sequences, Approximate Realization, Multigraph Realization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.14

Funding This work was supported by US-Israel BSF grant 2018043.

1 Introduction

1.1 Background and Motivation

Graphic degree sequences are among the most well-researched objects in the domain of graph realizations, studied extensively for over 60 years. A sequence $d = (d_1, \ldots, d_n)$ of non-negative integers is said to be a *graphic degree sequence* if there exists an n-vertex simple graph G such that $\deg(G) = d$, where $\deg(G)$ denotes the sequence of vertex degrees of G. The *graphic degree realization (GDR)* problem requires, given a sequence d, to decide whether d is graphic, and if so, to construct a graph G realizing it. Erdös and Gallai [18] gave a complete characterization for graphic degree sequences. However, their method does not provide a realizing graph. Havel and Hakimi [21, 24] gave an algorithm that, given a sequence d, generates a realizing graph, or proves that the sequence is not graphic, in time $\mathcal{O}(\sum_i d_i)$ which is optimal.

In this paper we consider the natural variant of the graphic degree realization problem, referred to as the *bigraphic degree realization (BDR)* problem, where the realizing graph is required to be bipartite. A sequence admitting a bipartite realizing graph is called a *bigraphic degree sequence*. This problem was mentioned in [33] as an open problem over 40 years ago, but we are unaware of any attempt to solve it.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 14; pp. 14:1–14:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The literature does contain, however, a sizeable amount of research on the simpler variant, hereafter referred to as the *given partition* version of the bigraphic degree realization problem, BDR^P, where the partition of d is given as part of the input. More explicitly, the input consists of a *partition*, namely, *two* sequences $a = (a_1, \ldots, a_p)$ and $b = (b_1, \ldots, b_q)$, and the question is to decide whether there exists a bipartite graph $G = (A, B, E)$ such that $|A| = p$, $|B| = q$, and the sequences of degrees of the vertices of A and B are equal to a and b, respectively. We refer to such a pair (a, b) as a *bigraphic degree partition.*

Interestingly, the history of bigraphic degree partitions is as ancient as that of graphic degree sequences. In 1957, Gale and Ryser [19, 35] gave a well-known complete characterization, known as the Gale-Ryser conditions, for a pair of sequences (a, b) to be a bigraphic degree partition. These conditions imply also a polynomial time decision algorithm for BDR^P (by applying a variant of the Havel-Hakimi construction procedure).

Given the Gale-Ryser characterization, the fact that the well-known PARTITION problem is pseudo-polynomial (cf. [9, 16]) would appear to suggest a plausible approach for attacking the (plain) bigraphic degree realization problem BDR, by searching for a bigraphic degree partition for the given sequence d. This approach makes sense because $d_1 < n$ is a necessary condition for a sequence d to be graphic (or bigraphic), and for such d, finding one of the partitions (if one exists) is achievable in polynomial time. However, this approach encounters several immediate obstacles. First, it is possible that some partitions of d are bigraphic while other partitions are not[1]. Second, there may be exponentially many different partitions for a given sequence[2]. Other approaches may be attempted, based on the special structure required by a bigraphic degree partition. So far, however, the bigraphic degree realization remains unresolved: There is no characterization for the class of bigraphic degree sequences, and it is unknown whether the BDR problem is NP-complete.

Towards attacking the bigraphic degree realization problem we study a specific and significant type of partitions that separate the degrees of a sequence by their size, i.e., into two blocks, a block of high degrees and a block of low ones. We refer to such partitions as *High-Low partitions.* These partitions represent an extreme approach, striving to maximize the difference between the degrees on the two sides of the partition. (An opposite extreme approach would be to try to make the two sides as *similar* as possible; we study the role of such partitions, referred to as *equal* partitions, in [8].) High-Low partitions are thus interesting in their own right, and can also be viewed as bipartite counterparts of other partition types considered in the literature, such as *core-periphery* partitions, which commonly occur in social networks, see [3, 4, 5, 11, 34, 46].

1.2 Our Contribution

Our key observation concerning the role of High-Low partitions is that there are special instances where deciding the realizability of the High-Low partition resolves also the BDR problem, i.e., decides whether the given sequence is bigraphic or not (and if so, provides an *exact* realization). Moreover, when the BDR problem is decided in the negative or is unsolved, we are able to generate *approximate* realizations for the given sequence based on its High-Low partition.

[1] Consider the sequence $(6, 6, 4, 4, 2, 2, 2, 2, 2, 2)$ which has three partitions: *(i.)* $(6, 6, 4)(4, 2, 2, 2, 2, 2, 2)$, *(ii.)* $(6, 4, 2, 2, 2)(6, 4, 2, 2, 2)$, and *(iii.)* $(6, 6, 2, 2)(4, 4, 2, 2, 2, 2)$. However, only the last partition is bigraphic.

[2] Consider the sequence $d = (n, n, n-1, n-1, \ldots, 2, 2, 1, 1)$ of length $2n$, for n divisible by 4. Split d into subsequences $B_j = (x, x, x+1, x+1, x+2, x+2, x+3, x+3)$, for $x = 4(j-1)+1$, noting that each B_j has three partitions: *(i.)* $(x, x+1, x+2, x+3)(x, x+1, x+2, x+3)$, *(ii.)* $(x, x, x+3, x+3)(x+1, x+1, x+2, x+2)$, and *(iii.)* $(x+1, x+1, x+2, x+2)(x, x, x+3, x+3)$. This yields $3^{n/4}$ different partitions for d.

Apart from being easier to generate, approximate realizations are desirable for several other reasons. In many applications it is required to design a network given some partial specifications, and returning that there is no suitable network (even provably) is not satisfactory. Additionally, specifications often rely on imprecise data making exact realizations unattractive. Bar-Noy et al. [6] survey different applications and types of approximate network realizations. Our approximate realizations are either

(i) bipartite *multigraphs* (namely, graphs that allow *parallel edges*) with low *maximum* multiplicity of parallel edges or

(ii) *super-realizations* which are bipartite (plain) graphs where a subset of the vertices adheres to the given degree sequence and with a small number of additional vertices and edges.

The *High-Low partition* of a non-increasing sequence $d = (d_1, \ldots, d_n)$ has the form $H = (d_1, \ldots, d_k)$ and $L = (d_{k+1}, \ldots, d_n)$ for some k. The partition (H, L) is *balanced* if $\sum_{i=1}^{k} d_i = \sum_{i=k+1}^{n} d_i$. Clearly, a bigraphic degree partition is necessarily balanced.

Well-behaved High-Low Partition. It turns out that for High-Low partitions, the first Gale-Ryser conditions are of paramount significance. These conditions stipulate that the largest degree on each side must not exceed the number of vertices on the other side (or formally, $d_1 \leq n - k$ and $d_{k+1} \leq k$). We refer to a balanced High-Low partition (H, L) that satisfies the first Gale-Ryser conditions as a *well-behaved* High-Low partition.

Being well-behaved does not, in itself, ensure that the partition is bigraphic[3]. It does, however, enable us to resolve the BDR problem: as we show in Section 3 that the BDR problem is solvable for a non-increasing sequence d that admits a well-behaved High-Low partition (H, L). Specifically, when d admits a well-behaved High-Low partition (H, L), it suffices to test the entire collection of Gale-Ryser conditions on (H, L). If all the conditions are met, then (H, L) is a bigraphic degree partition, hence d is a bigraphic degree sequence. If, on the other hand, one or more of the Gale-Ryser conditions is violated for (H, L), then *every* partition of d must violate one Gale-Ryser condition and d has *no* bigraphic degree partition. It follows that d itself is not a bigraphic degree sequence. On the positive side, we show in Section 4 that even in case a well-behaved High-Low partition fails to be bigraphic, it is still what we call 2-*bigraphic*, namely, it has a realizing bipartite *multigraph* whose maximum edge multiplicity is 2.

Multigraph Realizations. Next, we look at sequences that have a balanced High-Low partition that is *not* well-behaved, i.e., the first Gale-Ryser conditions are violated. Based on the High-Low partition we provide approximate realizations by bipartite *multigraphs* (without loops) measuring their quality by the maximum multiplicity of edges. We consider this notion for general graphs and bipartite graphs. Let r, t be positive integers. A sequence d of non-negative integers is said to be r-graphic if there exists a multigraph G such that $\deg(G) = d$ and the maximum multiplicity of an edge in G is at most r. Similarly, a partition (a, b) is t-bigraphic if there exists a bipartite multigraph $G(A, B, E)$ such that the maximum multiplicity of an edge in G is at most t and the sequences of degrees of the vertices of A and B are equal to a and b, respectively.

[3] Consider the sequence $((6m)^m, (2m)^{5m+1}, 1^{2m})$ (superscripts denote multiplicities of degrees) which has a well-behaved High-Low partition $H = ((6m)^m, (2m)^{m+1})$, $L = ((2m)^{4m}, 1^{2m})$, but it is not bigraphic.

In Section 4, we show that the balanced High-Low partition (H, L) of an r-graphic sequence is t-bigraphic where $t = \max\{t(d), 2r\}$ and $t(d)$ is a parameter indicating the extent to which (H, L) violates the first Gale-Ryser conditions.

Super-Graph Realizations. In Section 5, we deal with sequences where the High-Low partition is not balanced, and study the *High-Low near-partition* obtained by taking $H = \{d_1, \ldots, d_k\}$ and $L = \{d_{k+1}, \ldots, d_n\}$ for the smallest k such that $\sum_{i=1}^{k} d_i \geq \sum_{i=k+1}^{n} d_i$. For sequences where (H, L) satisfies the first Gale-Ryser conditions (called *quasi-well-behaved*), we come close to resolving its realizability status: Either we provide a bipartite super-realization, i.e., a bipartite graph G where $\deg(G) = d'$, and d is a subsequence of d' such that $|d'| - |d| \leq 2(d_k - 1)$ or we decide that d is not bigraphic.

Finally, in Section 6 we show how to combine the results on the two different types of (single-criterion) approximate realizations to yield bi-criterion approximate realizations, i.e., realizations by bipartite super-multigraphs.

1.3 Related Work

Next to characterizing graphic degree sequences, several related questions were considered: Given a degree sequence, find all the (non-isomorphic) graphs that realize it, count all its (non-isomorphic) realizing graphs, and uniformly sample a random realization. These questions are well-studied, cf. [13, 18, 21, 24, 26, 37, 39, 40, 44, 45], and have important applications in network design, randomized algorithms, social networks [10, 15, 17, 29] and chemical networks [38]. Miller [30] recapitulates reduced criteria for a sequence to be graphic. For surveys on graphic sequences, see [41, 42, 43].

Extensive literature exists on finding realizations having certain properties. A degree sequence is *potentially* P-graphic if it has a realization with property P where P is some graph theoretic property. Rao [33] surveys results (see references therein) on several properties including k-edge/k-vertex connected, hamiltonian and tournament. Characterizing potentially bipartite sequences, i.e., the BDR problem, is mentioned as an open problem.

Additional results include a characterization for trees (cf. [20]). The existing results on *planar* graphs are restricted to k*-sequences*, in which the difference between $\max d_i$ and $\min d_i$ is at most k, for $k = 0, 1, 2$ [1, 36]. Degree sequences of *split* graphs (see [23]), *threshold* graphs (see [22]), *matrogenic* graphs (see [28]) and *difference* graphs (see [22]) are fully characterized. Moreover, Degree sequences of *chordal*, *interval*, and *perfect* graphs were studied in [12].

Realizations by multigraphs were considered by Owens and Trent [32] showing how to realize degree sequences with minimum total number of parallel edges or loops (see [31, 27] for improved algorithms). Interestingly, computing a realization with maximum total number of parallel edges is known to be NP-hard, see [25].

2 Preliminaries and Definitions

Let $G = (V, E)$ be a multigraph without loops. Denote by $E_G(v, w)$ the multiset of edges connecting $v, w \in V$. The *maximum multiplicity* of G is $\mathsf{MaxMult}(G) = \max_{(v,w)\in E}(|E_G(v, w)|)$.

2.1 Degree Sequences of Graphs and Multigraphs

Let $d = (d_1, d_2, \ldots, d_n)$ be a sequence of nonnegative integers in nonincreasing order. The *volume* of d is $\sum d = \sum_{i=1}^{n} d_i$. We call a sequence with even volume a *degree sequence*.

We present the characterization of Erdös and Gallai [18] for graphic degree sequences.

▶ **Theorem 1** (Erdös-Gallai [18]). *A degree sequence $d = (d_1, d_2, \dots, d_n)$ is graphic if and only if, for $\ell = 1, \dots, n$,*

$$\sum_{i=1}^{\ell} d_i \le \ell(\ell-1) + \sum_{i=\ell+1}^{n} \min\{\ell, d_i\}. \tag{1}$$

We refer to Equation (1) as the *ℓ-th Erdös-Gallai inequality* EG_ℓ. The Erdös-Gallai characterization allows us to check if a sequence is graphic or not in polynomial time.

A degree sequence d is *r-graphic*, for a positive integer r, if there is a multigraph G such that $\deg(G) = d$ and $\mathsf{MaxMult}(G) \le r$. The Erdös-Gallai inequalities were extended to characterize r-graphic sequences.

▶ **Theorem 2** (Chungphaisan [14]). *Let r be a positive integer. The degree sequence $d = (d_1, d_2, \dots, d_n)$ is r-graphic if and only if, for $\ell = 1, \dots, n$,*

$$\sum_{i=1}^{\ell} d_i \le r\ell(\ell-1) + \sum_{i=\ell+1}^{n} \min\{r\ell, d_i\}, \tag{2}$$

Note that, for a given sequence d, the minimum r such that d is r-graphic can be determined in polynomial time.

2.2 Degree Sequences of Bipartite Graphs and Multigraphs

Let d be a degree sequence for which $\sum d = 2m$ for some integer m. A *block* of d is a subsequence a such that $\sum a = m$. Define $B(d) := \{a \subset d \mid \sum a = m\}$ as the set of all blocks of sequence d. For each $a \in B(d)$ there is a disjoint $b \in B(d)$ that completes it to form a partition of d (so that merging them in sorted order yields d). We call such a pair $a, b \in B(d)$ a *(balanced) partition* of d since $\sum a = \sum b$. Denote the set of all degree partitions of d by $\mathsf{BP}(d) = \{\{a, b\} \mid a, b \in B(d),\ d \setminus a = b\}$.

The Gale-Ryser theorem characterizes bigraphic degree partitions.

▶ **Theorem 3** (Gale-Ryser [19, 35]). *Let d be a degree sequence and partition $(a, b) \in \mathsf{BP}(d)$ where $a = (a_1, a_2, \dots, a_p)$ and $b = (b_1, b_2, \dots, b_q)$. The partition (a, b) is bigraphic if and only if*

$$\sum_{i=1}^{\ell} a_i \le \sum_{i=1}^{q} \min\{\ell, b_i\}, \tag{3}$$

for $\ell = 1, \dots, p$.

We refer to Equation (3) as the *ℓ-th Gale-Ryser inequality* GR^L_ℓ on the left. By symmetry, the partition (a, b) is bigraphic if and only if $\sum_{i=1}^{\ell} b_i \le \sum_{i=1}^{p} \min\{\ell, a_i\}$, for $\ell = 1, \dots, q$. We refer to this equation as the *ℓ-th Gale-Ryser inequality* GR^R_ℓ on the right.

Let t be a positive integer. A degree sequence d is t-bigraphic if d has a partition $(a, b) \in \mathsf{BP}(d)$ such that there is a bipartite multigraph $G = (A, B, E)$ such that $\mathsf{MaxMult}(G) \le t$, $|A| = |a|$, $|B| = |b|$, and the sequences of degrees of the vertices of A and B are equal to a and b, respectively. We also say that partition (a, b) is t-bigraphic. Miller [30] cites the following result of Berge characterizing t-bigraphic partitions.

▶ **Theorem 4** (Berge [30]). *Consider a positive integer t, a degree sequence d and a partition $(a,b) \in BP(d)$ where $a = (a_1,\dots,a_p)$ and $b = (b_1,\dots,b_q)$. The partition (a,b) is t-bigraphic if and only if*

$$\sum_{i=1}^{\ell} a_i \le \sum_{i=1}^{q} \min\{\ell t, b_i\}, \tag{4}$$

for $\ell = 1,\dots,p$.

2.3 Ferrers Diagrams, Conjugate Sequences and Majorization

Ferrers diagrams (cf. [2]) are instrumental in illustrating integer sequences and partitions graphically (see Figure 1).

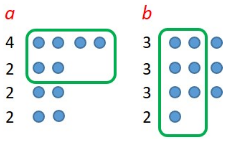


Figure 1 The Ferrers diagram of the partition $a = (4,2,2,2)$ and $b = (3,3,3,1)$.

Conjugate sequences provide us with a convenient alternative way to represent the Gale-Ryser conditions. The *prefix-sum* of a sequence d up to index i is $\sum(d[i]) = \sum_{j=1}^{i} d_j$. Let the sequences a,b have the same length, i.e., $p = q$. (If two sequences are not of the same length, the shorter sequence can be padded with 0's.) Given a degree sequence d, its *conjugate* sequence $\widetilde{d} = (\widetilde{d}_1, \widetilde{d}_2, \dots, \widetilde{d}_{d_1})$ is defined by $\widetilde{d}_k = |\{i \mid d_i \ge k\}|$.

The duality between $\sum a[i]$ and $\sum \widetilde{b}[i]$ is captured graphically in the Ferrers diagram: a_i is the i-th row on the left, and $\widetilde{b}_i$ is the i-th column on the right. Consequently, $\sum a[i]$ is the sum of the first i rows on the left, whereas $\sum \widetilde{b}[i]$ is the sum of the first i columns on the right. (In the figure, the green ovals capture $\sum a[2]$ and $\sum \widetilde{b}[2]$.)

Majorization is a partial order on degree sequences: *a majorizes b* ($b \trianglelefteq a$) if and only if $\sum(b[i]) \le \sum(a[i])$ for every $i \in [1,q]$.

Observe that if $a \trianglerighteq b$, then $\widetilde{b} \trianglerighteq \widetilde{a}$. The Gale-Ryser theorem can now be reformulated using majorization and conjugates, noting that $\sum_{i=1}^{q} \min\{\ell, b_i\} = \sum_{i=1}^{\ell} \widetilde{b}_i$.

▶ **Theorem 5** (Gale-Ryser [19, 35], conjugate representation). *Let d be a degree sequence and $(a,b) \in BP(d)$. The partition (a,b) is bigraphic if and only if $a \trianglelefteq \widetilde{b}$.*

Furthermore, a graphic description of the ℓ-th Gale-Ryser condition on the left is that the sum of the first ℓ *rows* on the left Ferrers diagram (representing a), must be no greater than the sum of the first ℓ *columns* on the right Ferrers diagram (representing b). (The sequence is bigraphic if and only if these conditions hold for every $\ell \le p$.) As the sequences a and b (or equivalently their Ferrers diagrams) can switch sides, it is clear that the Gale-Ryser conditions are symmetric, i.e., $a \trianglelefteq \widetilde{b}$ if and only if $b \trianglelefteq \widetilde{a}$.

3 Well-Behaved High-Low Partitions

In this section, we study sequences that have a well-behaved High-Low partition. Let $d = (d_1,\dots,d_n)$ be a degree sequence with a High-Low partition $HL(d) = (H,L)$ where $H = (d_1,\dots,d_k)$ and $L = (d_{k+1},\dots,d_n)$, for some positive integer $k < n$. We assume that $d_1 \le n-k$ and that $d_{k+1} \le k$, i.e., (H,L) is well-behaved.

In the following, we suppose that (H, L) is not bigraphic implying that at least one Gale-Ryser condition on H and L, respectively, is not satisfied. Let x be the index of the first violated Gale-Ryser condition on L, i.e., such that $\sum(L[i]) \leq \sum(\widetilde{H}[i])$, for $i < x$, and

$$\sum(L[x]) > \sum(\widetilde{H}[x]). \tag{5}$$

▶ **Observation 6.** *For a well-behaved $HL(d) = (H, L)$, $d_k < x$.*

Proof. For $i \leq d_k$, we have that $\sum(\widetilde{H}[i]) = i \cdot k$, and $\sum(L[i]) \leq i \cdot d_{k+1}$ (see Figure 2). Since (H, L) is well-behaved, $d_{k+1} \leq k$, and it follows that $\sum(L[i]) \leq \sum(\widetilde{H}[i])$, hence the ith Gale-Ryser condition on L holds, for $i \leq d_k$. Consequently, $x > d_k$. ◀

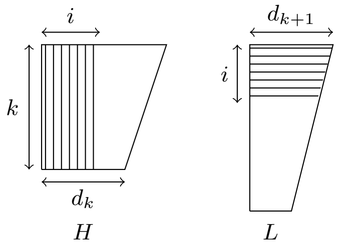


Figure 2 $\sum \tilde{H}[i]$ vs. $\sum L[i]$.

▶ **Observation 7.** *For a well-behaved $HL(d) = (H, L)$, $x > k$.*

Proof. If $d_k \geq k$, then the claim follows from Observation 6. Now suppose $d_k < k$. We need to show that $\sum(\widetilde{H}[j]) \geq \sum(L[j])$ for every $d_k < j \leq k$. This follows because

$$\sum(\widetilde{H}[j]) \geq \sum(\widetilde{H}[d_k]) = k \cdot d_k \geq k \cdot d_{k+1} \geq \sum(L[k]) \geq \sum(L[j]).$$ ◀

Our main goal is to prove the following result.

▶ **Theorem 8.** *Consider a degree sequence d with a well-behaved High-Low partition $HL(d) = (H, L)$. If (H, L) is not bigraphic, then d is not bigraphic (i.e., no partition of d is bigraphic).*

First, we prove a weaker statement. For some other partition $(A, B) \in \mathtt{BP}(d)$, define

$$H^m = H \setminus A, \qquad L^m = L \setminus B, \qquad H^s = H \cap A, \text{ and } \qquad L^s = L \cap B.$$

Moreover, we denote $h = |H^m|$ and $\ell = |L^m|$. Hence, the partition (A, B) can be viewed as obtained from (H, L) by *moving* a subset H^m from the left to the right, and a subset L^m from the right to the left. The subsets H^s and L^s *stay* on their respective sides. Note that, keeping A and B sorted in nonincreasing order, the elements of H^m appear at the beginning of B and the elements of L^m appear at the end of A. Figure 3 illustrates this transformation.

For an index i and an ordered set of integers S, denote the first i elements of S by $S[i]$. Moreover, we use $\sum(S) = \sum_{z \in S} z$ to denote the sum of elements in S. In line with our previous definition, $\sum(S[i])$ is the prefix-sum of S up to index i.

▶ **Definition 9.** *Call the partition (A, B)* benign *if L^m contains at most h of the top x elements of L, i.e., $|L[x] \cap L^m| \leq h$ or equivalently $|L[x] \setminus L^m| \geq x - h$.*

▶ **Lemma 10.** *Consider a degree sequence d with well-behaved High-Low partition $HL(d) = (H, L)$. If (H, L) is not bigraphic, then no benign partition of d is bigraphic.*

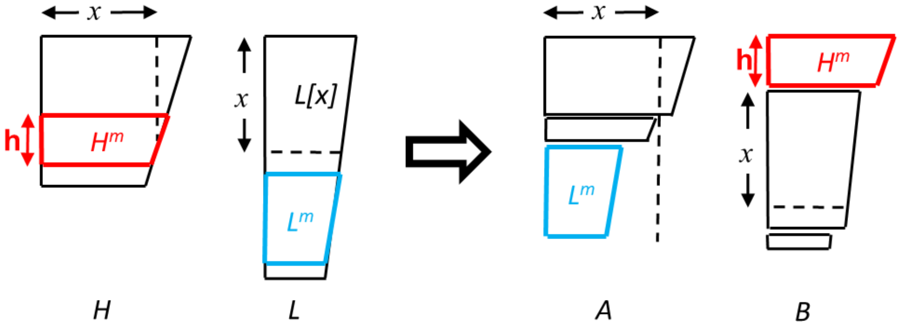


Figure 3 Illustration of the transformation from (H, L) to (A, B). Partition (A, B) is benign since $L[x] \cap L^m$ is empty.

Proof. Let d and $HL(d) = (H, L)$ be as in the lemma, and let (A, B) be some benign partition of d. To show the lemma, we show that $\sum(B[x]) > \sum(\widetilde{A}[x])$ holds, i.e., the partition (A, B) violates GR^R_x. First, verify that

$$\sum(B[x]) = \sum(H^m) + \sum(L^s[x-h]),$$

and that

$$\sum(\widetilde{A}[x]) = \sum(L^m) + \sum(\widetilde{H^s}[x]) \tag{6}$$

since $x > d_{k+1}$ by Observation 6. We need to show that $\sum(L^s[x-h]) > \sum(\widetilde{H^s}[x])$ since $\sum(H^m) = \sum(L^m)$. By Equation (5), it follows that

$$\sum(L[x]) - \sum(L^s[x-h]) + \sum(L^s[x-h]) > \sum(\widetilde{H}[x]) = \sum(\widetilde{H^s}[x]) + \sum(\widetilde{H^m}[x]).$$

To finish the proof, we argue that $\sum(\widetilde{H^m}[x]) \geq \sum(L[x]) - \sum(L^s[x-h])$. Since (A, B) is benign, $L^s[x-h]$ contains at least $x - h$ rows of $L[x]$, or equivalently $L[x] \setminus L^s[x-h]$ are at most h rows of L. It follows that

$$\sum(L[x]) - \sum(L^s[x-h]) \leq h \cdot d_k \leq \sum(\widetilde{H^m}[x])$$

where the last inequality holds due to Observation 6. ◀

Using the symmetry of the Gale-Ryser conditions we prove the following corollary. We introduce another notation, $s = |L[x] \cap L^m|$.

▶ **Corollary 11.** *Consider a degree sequence d with well-behaved High-Low partition $HL(d) = (H, L)$. If (H, L) is not bigraphic, then no partition of d where $x - s \leq k - h$ is bigraphic.*

Proof. The result is obtained from switching the definitions of the moving and staying parts. Partition (B, A) can be viewed as obtained from (H, L) by moving H^s from H to L and L^s from L to H. Note that $|H^s| = k - h$ and $|L^s \cap L[x]| = x - s$. The condition $x - s \leq k - h$ implies that (B, A) is benign, and the corollary follows with Lemma 10. ◀

Another benign case is when the largest element of H^m or H^s is smaller or equal to x. Denote $\lambda = \min\{\max\{H^m\}, \max\{H^s\}\}$.

▶ **Lemma 12.** *Consider a degree sequence d with well-behaved High-Low partition $HL(d) = (H, L)$. If (H, L) is not bigraphic, then no partition of d where $\lambda \leq x$ is bigraphic.*

Proof. Consider d and (H, L) as in the lemma. First, we verify the claim for $\lambda = \max\{H^m\}$. To that end, let (A, B) be a partition of d such that $\max\{H^m\} \leq x$. To prove the claim, we show that $\sum(B[x]) > \sum(\widetilde{A}[x])$ holds. With Equations (5) and (6) it follows that

$$\sum(L[x]) > \sum(\widetilde{H}[x]) = \sum(\widetilde{H^s}[x]) + \sum(\widetilde{H^m}[x]) \overset{(\star)}{=} \sum(\widetilde{H^s}[x]) + \sum(H^m) = \sum(\widetilde{A}[x]),$$

where $(\star)$ holds since $\max\{H^m\} \leq x$. Because (H, L) is a High-Low partition, B majorizes L and, in particular, $\sum(B[x]) \geq \sum(L[x]) > \sum(\widetilde{A}[x])$ holds.

With the arguments used to prove Corollary 11, the claim also holds in case $\lambda = \max\{H^s\}$, and the lemma follows. ◀

Now we are ready to prove Theorem 8.

Proof of Theorem 8. Consider d and $HL(d) = (H, L)$ as in the lemma, and let (A, B) be some partition of d. Towards a contradiction suppose that (A, B) is bigraphic. It follows that

(a) $\sum(B[x]) \leq \sum(\widetilde{A}[x])$,
(b) $\sum(A[x]) \leq \sum(\widetilde{B}[x])$,
(c) $\lambda \leq \sum(B[1]) \leq \sum(\widetilde{A}[1]) = k - h + \ell$,
(d) $\lambda \leq \sum(A[1]) \leq \sum(\widetilde{B}[1]) = n - k + h - \ell$,
(e) $x < \lambda$,
(f) $s > h$, and
(g) $s < x - k + h$.

Equations (a), (b), (c), (d) are Gale-Ryser conditions (see Theorem 3), and the upper bounds on λ hold by definition. As (A, B) is bigraphic, Equation (e) is implied by Lemma 12. Moreover, Equations (f) and (g) are implied by Lemma 10 and Corollary 11, respectively.

Recall that Equation (5) reads

$$\sum(L[x]) > \sum(\widetilde{H}[x]) = \sum(\widetilde{H^s}[x]) + \sum(\widetilde{H^m}[x]).$$

Note that (d) and (e) imply $x < \lambda \leq n - k + h - \ell = |B|$ (i.e., $B[x]$ is a proper subset of B). With Equation (a) we have that

$$\sum(\widetilde{H^s}[x]) + \sum(L^m) = \sum(\widetilde{A}[x]) \geq \sum(B[x]) = \sum(L^s[x-h]) + \sum(H^m).$$

Since $\sum(L^m) = \sum(H^m)$ we get a lower bound on $\sum(\widetilde{H^s}[x])$:

$$\sum(\widetilde{H^s}[x]) \geq \sum(L^s[x-h]). \tag{7}$$

Note that (c) and (e) imply $x < \lambda \leq k - h + \ell = |A|$ (i.e., $A[x]$ is a proper subset of A). Since $k - h < x$ and $d_{k+1} < x$, it follows from Observations 6 and 7 that

$$\sum(L^s) + \sum(\widetilde{H^m}[x]) = \sum(\widetilde{B}[x]) \geq \sum(A[x]) = \sum(H^s) + \sum(L^m[x-k+h]).$$

Since $\sum(L^s) = \sum(H^s)$ we get a lower bound on $\sum(\widetilde{H^m}[x])$:

$$\sum(\widetilde{H^m}[x]) \geq \sum(L^m[x-k+h]). \tag{8}$$

Equation (5) together with Equations (7) and (8) yields

$$\sum(L[x]) > \sum(L^s[x-h]) + \sum(L^m[x-k+h]).$$

Observe that $L^s[x-h]$ contains $L[x] \setminus L^m$ as $s > h$. Since $s < x - k + h$, $L^m[x-k+h]$ contains $L[x] \cap L^m$. It follows that $\sum(L^s[x-h]) \cup \sum(L^m[x-k+h])$ contains $L[x]$, and so

$$\sum(L^s[x-h]) + \sum(L^m[x-k+h]) \geq \sum(L[x])$$

holds contradicting the previous equation. Consequently, (A, B) cannot be bigraphic. ◀

Theorem 8 implies that the bigraphic realizability status can be fully resolved for a degree sequence with a well-behaved High-Low partition.

▶ **Theorem 13.** *Let d be a degree sequence with a well-behaved High-Low partition. It can be decided in polynomial time whether d is bigraphic or not. If d happens to be bigraphic, a bipartite graph realizing d can be computed in polynomial time.*

Proof. Let d be as in the theorem. Computing the High-Low partition $HL(d) = (H, L)$ is straight forward. Using the Gale-Ryser theorem (Theorem 3), we decide if (H, L) is bigraphic or not, and due to Theorem 8 if d is bigraphic or not. If (H, L) is bigraphic, a bipartite graph realizing d can be computed by applying the Havel-Hakimi algorithm to one side of the partition (see [7] for details). All steps are performed in polynomial time. ◀

4 Realizations by Bipartite Multigraphs

Our goal is to provide realizations based on multigraphs without loops where the maximum multiplicity of parallel edges is used to measure their quality. We examine degree sequences that have a balanced High-Low partition but are not necessarily bigraphic. Let r be a positive integer. In the following, we consider an r-graphic degree sequence d with High-Low partition $HL(d) = (H, L)$ where $H = (d_1, \ldots, d_k)$ and $L = (d_{k+1}, \ldots, d_n)$, for some integer $k \in [1, n-1]$. We do not assume that d is well-behaved, and quantify the violation of the first Gale-Ryser conditions with the following definitions. Let

$$t_H(d) = \left\lceil \frac{d_1}{n-k} \right\rceil \qquad \text{and} \qquad t_L(d) = \left\lceil \frac{d_{k+1}}{k} \right\rceil ,$$

and define $t(d) = \max\{t_H(d), t_L(d)\}$. (Note that sequence d has a well-behaved High-Low partition if $t(d) = 1$.) First, we observe that $t_H(d)$ is bounded for r-graphic sequences.

▶ **Lemma 14.** *Let d be an r-graphic degree sequence with High-Low partition $HL(d) = (H, L)$. Then, $t_H(d) \leq 2r$.*

Proof. Let d be as in the lemma with $H = (d_1, \ldots, d_k)$ and $L = (d_{k+1}, \ldots, d_n)$. Since (H, L) is a High-Low partition, we have that $k \leq n-k$. Moreover, $d_1 \leq r(n-1)$ as d is r-graphic. It follows that

$$t_H(d) = \left\lceil \frac{d_1}{n-k} \right\rceil \leq \left\lceil \frac{r(n-1)}{n-k} \right\rceil \leq \left\lceil \frac{r(k+n-k)}{n-k} \right\rceil \leq 2r. \qquad ◀$$

The main result of this section is the next theorem. Its proof is omitted in the conference version of the paper.

▶ **Theorem 15.** *Let d be an r-graphic degree sequence with High-Low partition $HL(d) = (H, L)$ and let $t = \max\{t(d), 2r\}$. Then, (H, L) is t-bigraphic.*

We remark that the conclusion of Theorem 15 does not hold if the degree sequence d is not r-graphic. To see this, consider the non-graphic sequence $d = ((9m)^{m-1}, 6m+1, (3m)^{3m-1}, 1^1)$ for some positive integer m. (We use superscripts to denote the multiplicities of degrees.) Verify that d has a High-Low partition (H, L) where $H = ((9m)^{m-1}, 6m+1)$ and $L = ((3m)^{3m-1}, 1^1)$. We have $t_H(d) = t_L(d) = 3$, but the conditions of Theorem 4 for 3-bigraphic degree sequences are violated. Specifically, the condition for index $m-1$ requires $9m(m-1) \leq (3m-1) \cdot 3 \cdot (m-1) + 1$, which is false.

We complement Theorem 15 by providing an existential lower bound.

▶ **Lemma 16.** *There are degree sequences d with High-Low partition $HL(d)$ such that $t(d) > 1$, and d is not t'-bigraphic for any $t' < t(d)$.*

Proof. Let t be a positive integer, and let p be some prime number. Set $x \geq t(p+1)$ such that p is not a prime factor of x, and consider the sequence $d = (x^{p+2}, p^x)$. Verify that sequence d has a High-Low partition $HL(d) = (H, L)$ with blocks $H = (x^{p+1})$ and $L = (x, p^x)$. By the choice of x and p, d does not have other partitions. We have that $t(d) = t_L(d) \geq t$. It follows that d is not t'-bigraphic for any $t' \leq t$. ◀

For graphic degree sequences, we get the following result.

▶ **Corollary 17.** *Let d be a graphic degree sequence with High-Low partition $HL(d) = (H, L)$ and $t = t(d)$.*

(i) *If $t = 1$, then (H, L) is 2-bigraphic.*
(ii) *If $t > 1$, then (H, L) is t-bigraphic.*

If there is a well-behaved High-Low partition, Theorem 8 implies the following result.

▶ **Corollary 18.** *Let d be a graphic degree sequence with well-behaved High-Low partition $HL(d) = (H, L)$. Then, either*

(i) *(H, L) is bigraphic, or*
(ii) *d is not bigraphic and (H, L) is 2-bigraphic.*

Combinatorial Bounds. In the last part of this section, we show bounds on $t_L(d)$ and $t_H(d)$ in case the degree sequence d is r-graphic or bigraphic. The proofs of the following two theorems are omitted in the conference version of the paper.

We start with r-graphic degree sequences. Lemma 14 provides a bound on $t_H(d)$. The next theorem establishes a bound on $t_L(d)$.

▶ **Theorem 19.** *Let d be an r-graphic sequence with High-Low partition $HL(d) = (H, L)$. Then,*

$$t_L(d) \leq \left\lceil \frac{r(k+1)}{2} \right\rceil.$$

We note that the bound of Theorem 19 is tight and that for graphic sequences, $t_L(d) < t_H(d)$ as well as $t_H(d) < t_L(d)$ can occur. To see this, consider the following two examples.

(1) The graphic sequence $d = (6, 3, 3, 3, 3, 3, 3)$ has exactly one (High-Low) partition (H, L) with blocks $H = (6, 3, 3)$ and $L = (3, 3, 3, 3)$. Verify that $t_L(d) = 2$ and $t_H(d) = 1$.
(2) The degree sequence $d' = ((\frac{k(k+1)}{2})^{k+1}, 1^{\frac{k(k+1)}{2}(k-1)})$, for a positive integer k, is graphic (to see this, observe that $\sum d'$ is even, and that the $(k+1)$th-EG inequality holds; for such a block sequence this is sufficient, see, e.g., [30]). The $(k+1)$th-EG inequality reads $(k(k+1)/2) \cdot (k+1) \leq k(k+1) + (k(k+1)/2) \cdot (k-1)$, which trivially holds. Moreover, $HL(d') = (H', L')$ where $H' = ((\frac{k(k+1)}{2})^k)$ and $L' = ((\frac{k(k+1)}{2}), 1^{\frac{k(k+1)}{2}(k-1)})$. Hence, $|H'| = k$ and $d_{k+1} = \frac{k(k+1)}{2}$.

The next result improves the bounds on $t_L(d)$ and $t_H(d)$ for bigraphic degree sequences.

▶ **Theorem 20.** *Let d be a bigraphic sequence with High-Low partition $HL(d)$. Then,*

$$t_H(d) \leq 1, \quad \textit{and} \qquad t_L(d) \leq \left\lceil \frac{k+2+1/k}{4} \right\rceil.$$

5 Realizations by Bipartite Super-Graphs

We next focus on the situation when the given sequence does not admit a balanced High-Low partition (with $\sum H = \sum L$). In this case, we consider the *High-Low near-partition (HLnP)* (H, L) of d, obtained by taking $H = \{d_1, \ldots, d_k\}$ and $L = \{d_{k+1}, \ldots, d_n\}$ for the smallest k such that $\sum H \geq \sum L$ (i.e., $\sum H - d_k < \sum L + d_k$).

Define the *imbalance gap* of the sequence d to be $IG(d) = \sum H - \sum L$. Rearranging the above two inequalities, we get the following.

▶ **Observation 21.** $0 \leq IG(d) < 2d_k$.

In the remainder of this section we assume that $IG(d) > 0$. We refer to such sequences d as *High-Low-imbalanced*. We call the High-Low near-partition (H, L) *quasi-well-behaved* if it satisfies the first Gale-Ryser condition on both sides.

The main result of this section is that when the given sequence d is High-Low-imbalanced but enjoys a quasi-well-behaved High-Low near-partition, it is possible to come close to resolving its realizability status, in the sense that there is a poly-time algorithm that either decides that d is not bigraphic, or constructs a bipartite super-realization for d with a small number of new vertices and edges. We use the operator $\circ$ to merge two sequences.

▶ **Definition 22.** *A bipartite (n', m') super-realization of an n-integer sequence d, for $n' > n$ and $m' > \sum d$, is a bipartite graph $G(A, B, E)$ such that $|A \cup B| = n + n'$, $|E| = \sum d + m'$, and $deg(A \cup B) = d \circ d'$ for some sequence d' of n' integers.*

Our algorithm hinges on the idea of completing a High-Low near-partition of a given imbalanced sequence d into a (balanced) High-Low partition of a larger sequence in a suitable way. Consider a family D of quasi-well-behaved n-integer nonincreasing sequences. A mapping $\varphi : D \mapsto D'$ is said to be a *valid completion mapping* for D if for every $d \in D$ such that $HLnP(d) = (H, L)$ and $k = |H|$, the generated n'-integer sequence $d' = \varphi(d)$ satisfies the following properties.

(P1) $(H, L \circ d')$ is a well-behaved High-Low partition of $d \circ d'$.

(P2) If d is bigraphic then $d \circ d'$ is bigraphic as well.

The generated sequence $d' = \varphi(d)$ is referred to as the *valid completion* of d.

We now describe a generic Algorithm $A(d, d')$ that, given a High-Low-imbalanced n-integer nonincreasing sequence d with $IG(d) = t$ and a valid completion d' of p integers for d, generates a bipartite (p, t) super-realization for d. The algorithm operates as follows.

1. Construct the High-Low near-partition $HLnP(d) = (H, L)$ for the given sequence d.
2. Let $t \leftarrow IG(d) = \sum H - \sum L$.
3. Set $L' \leftarrow L \circ d'$ (sorted in nonincreasing order).
4. Test all Gale-Ryser conditions on (H, L').
5. If (H, L') is bigraphic, then construct and return a realizing bipartite graph G' for it.
6. Otherwise (* (H, L') is not bigraphic *) return "d is not bigraphic".

▶ **Lemma 23.** *Consider a sequence d and let d' be a valid completion for d. If Algorithm $A(d, d')$ returns a graph G' (Step 5), then it is a bipartite (p, t) super-realization for d. If the algorithm returns a negative response (Step 6), then d is indeed not bigraphic.*

Proof. The first claim follows immediately by the definition of p and t and the fact that G' is a bipartite realization of (H, L').

To prove the second claim, suppose (H, L') is non-bigraphic. As d' is a valid completion of d, property (P1) implies that (H, L') is a well-behaved High-Low partition of $d \circ d'$. It follows from Theorem 8 that $d \circ d'$ is also non-bigraphic. This, in turn, implies by property (P2) that d is not bigraphic. ◀

▶ **Lemma 24.** *Consider a quasi-well-behaved n-integer nonincreasing sequence d with $t = IG(d)$. The sequence $d' = (1^t)$ is a valid completion for d.*

Proof. To see that (P1) holds, observe that

(1) $(H, L \circ d')$ is a High-Low partition of $d \circ d'$ because the elements of d' are no greater than the elements of H.

(2) it is balanced because $\sum H = \sum(L \circ d')$ by the choice of t.

(3) it satisfies the first Gale-Ryser condition on the left since (H, L) is quasi-well-behaved, so $d_1 \leq n - k \leq n + t - k$.

(4) it satisfies the first Gale-Ryser condition on the right since d_{k+1} satisfies $d_{k+1} \leq k$ by the fact that (H, L) is quasi-well-behaved, and $d'_1 = 1 \leq k$, so $\max(L \circ d') \leq k$.

To see that (P2) holds, suppose d is bigraphic, and let G be a bipartite graph realizing it. Noting that $IG(d)$ must be even (as it is the difference of two integers whose sum is even), let M be a matching consisting of $t/2$ edges. Then $G \cup M$ is a bipartite realization of $d \circ d'$. ◀

We conclude the following.

▶ **Theorem 25.** *Consider a quasi-well-behaved n-integer nonincreasing sequence d and let $d' = (1^{IG(d)})$. Then Algorithm $A(d, d')$, in poly-time, either yields a bipartite $(IG(d), IG(d))$ super-realization for d or decides that d is not bigraphic.*

Our goal is to find a poly-time algorithm that constructs a bipartite super-realization for d with less new vertices. We continue this analysis in the journal version of the paper.

6 Realizations by Super-Multigraphs

We consider super-multigraph realizations based on the High-Low near-partition. Together, Theorems 15 and 25 imply the following.

▶ **Corollary 26.** *Let d be an r-graphic sequence, $d' = d \circ 1^{IG(d)}$, and $t = \max\{2r, t(d')\}$. Then, there is a t-bipartite $(IG(d), IG(d))$ super-realization for d.*

If d is quasi-well-behaved and graphic, we apply Corollary 18 yielding the following.

▶ **Corollary 27.** *Consider a quasi-well-behaved and graphic sequence d and let $d' = (1^{IG(d)})$. Either d is undecided and Algorithm $A(d, d')$ yields in poly-time a bipartite $(IG(d), IG(d))$ super-realization for d, or d is not bigraphic and has a 2-bipartite $(IG(d), IG(d))$ super-realization.*

References

1 P. Adams and Y. Nikolayevsky. Planar bipartite biregular degree sequences. *Discrete Mathematics*, 342:433–440, 2019.

2 G. E Andrews. *The Theory of Partitions*. Cambridge University Press, 1998.

3 C. Avin, M. Borokhovich, Z. Lotker, and D. Peleg. Distributed computing on core–periphery networks: Axiom-based design. *Journal of Parallel and Distributed Computing*, 99:51–67, 2017.

4 C. Avin, Z. Lotker, Y. Nahum, and D. Peleg. Core size and densification in preferential attachment networks. In *International Colloquium on Automata, Languages, and Programming*, pages 492–503. Springer, 2015.

5 C. Avin, Z. Lotker, D. Peleg, Y.-A. Pignolet, and I. Turkel. Elites in social networks: An axiomatic approach to power balance and price's square root law. *PloS one*, 13(10):e0205820, 2018.

6 A. Bar-Noy, T. Böhnlein, D. Peleg, M. Perry, and D. Rawitz. Relaxed and approximate graph realizations. In *International Workshop on Combinatorial Algorithms*, pages 3–19. Springer, 2021.

7 A. Bar-Noy, T. Böhnlein, D. Peleg, and D. Rawitz. On Realizing a Single Degree Sequence by a Bipartite Graph. In *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227, pages 1:1–1:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

8 A. Bar-Noy, T. Böhnlein, D. Peleg, and D. Rawitz. On realizing even degree sequences by bipartite graphs. Unpublished manuscript, 2022.

9 R. Bellman. Notes on the theory of dynamic programming iv-maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, 1956.

10 J. K. Blitzstein and P. Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, 6(4):489–522, 2011.

11 S. P Borgatti and M. G Everett. Models of core/periphery structures. *Social Networks*, 21(4):375–395, 2000.

12 A. A. Chernyak, Z. A. Chernyak, and R. I. Tyshkevich. On forcibly hereditary p-graphical sequences. *Discrete Mathematics*, 64:111–128, 1987.

13 S. A. Choudum. A simple proof of the Erdös-Gallai theorem on graph sequences. *Bull. Austral. Math. Soc.*, 33(1):67–70, 1991.

14 V Chungphaisan. Conditions for sequences to be r-graphic. *Discrete Mathematics*, 7(1-2):31–39, 1974.

15 B. Cloteaux. Fast sequential creation of random realizations of degree sequences. *Internet Mathematics*, 12(3):205–219, 2016.

16 T. H Cormen, C. E Leiserson, R. L Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

17 Dóra Erdös, R. Gemulla, and E. Terzi. Reconstructing graphs from neighborhood data. *ACM Trans. Knowledge Discovery from Data*, 8(4):23:1–23:22, 2014.

18 P. Erdös and T. Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.

19 D. Gale. A theorem on flows in networks. *Pacific J. Math.*, 7:1073–1082, 1957.

20 G. Gupta, P. Joshi, and A. Tripathi. Graphic sequences of trees and a problem of Frobenius. *Czechoslovak Math. J.*, 57:49–52, 2007.

21 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph –I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.

22 P. L. Hammer, T. Ibaraki, and B. Simeone. Threshold sequences. *SIAM J. Algebra. Discr.*, 2(1):39–49, 1981.

23 P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1:275–284, 1981.

24 V. Havel. A remark on the existence of finite graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.

25 H. Hulett, T. G Will, and G. J Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36(5):594–596, 2008.

26 P.J. Kelly. A congruence theorem for trees. *Pacific J. Math.*, 7:961–968, 1957.

27 D. J. Kleitman. Minimal number of multiple edges in realization of an incidence sequence without loops. *SIAM Journal on Applied Mathematics*, 18(1):25–28, 1970.

28 P. Marchioro, A. Morgana, R. Petreschi, and B. Simeone. Degree sequences of matrogenic graphs. *Discrete Mathematics*, 51(1):47–61, 1984. `doi:10.1016/0012-365X(84)90023-2`.

29 M. Mihail and N. Vishnoi. On generating graphs with prescribed degree sequences for complex network modeling applications. *3rd ARACNE*, 2002.

30 J. W Miller. Reduced criteria for degree sequences. *Discrete Mathematics*, 313(4):550–562, 2013.

31 A. B Owens. On determining the minimum number of multiple edges for an incidence sequence. *SIAM J. on Applied Math.*, 18(1):238–240, 1970.

32 AB Owens and HM Trent. On determining minimal singularities for the realizations of an incidence sequence. *SIAM J. on Applied Math.*, 15(2):406–418, 1967.

33 S. B. Rao. A survey of the theory of potentially p-graphic and forcibly p-graphic degree sequences. In *Combinatorics and graph theory*, volume 885 of *LNM*, pages 417–440, 1981.

34 M P. Rombach, M. A Porter, J. H Fowler, and P. J Mucha. Core-periphery structure in networks. *SIAM Journal on Applied mathematics*, 74(1):167–190, 2014.

35 H.J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canad. J. Math.*, 9:371–377, 1957.

36 E. F. Schmeichel and S. L. Hakimi. On planar graphical degree sequences. *SIAM J. Applied Math.*, 32:598–609, 1977.

37 G. Sierksma and H. Hoogeveen. Seven criteria for integer sequences being graphic. *J. Graph Theory*, 15(2):223–231, 1991.

38 A. Tatsuya and H. Nagamochi. Comparison and enumeration of chemical graphs. *Computational and structural biotechnology*, 5, 2013.

39 A. Tripathi and H. Tyagi. A simple criterion on degree sequences of graphs. *Discrete Applied Mathematics*, 156(18):3513–3517, 2008.

40 R. Tyshkevich. Decomposition of graphical sequences and unigraphs. *Discrete Mathematics*, 220:201–238, 2000.

41 R. I. Tyshkevich, A. A. Chernyak, and Z. A. Chernyak. Graphs and degree sequences: A survey, I. *Cybernetics*, 23:734–745, 1987.

42 R. I. Tyshkevich, A. A. Chernyak, and Z. A. Chernyak. Graphs and degree sequences: A survey, II. *Cybernetics*, 24:137–152, 1988.

43 R. I. Tyshkevich, A. A. Chernyak, and Z. A. Chernyak. Graphs and degree sequences: A survey, III. *Cybernetics*, 24:539–548, 1988.

44 S.M. Ulam. *A collection of mathematical problems*. Wiley, 1960.

45 N.C. Wormald. Models of random regular graphs. *Surveys in Combin.*, 267:239–298, 1999.

46 X. Zhang, T. Martin, and M. EJ Newman. Identification of core-periphery structure in networks. *Physical Review E*, 91(3):032803, 2015.

Towards a Model Theory of Ordered Logics: Expressivity and Interpolation

Bartosz Bednarczyk

Computational Logic Group, Technische Universität Dresden, Germany

Institute of Computer Science, University of Wrocław, Poland

Reijo Jaakkola

Tampere University, Finland

Abstract

We consider the family of guarded and unguarded ordered logics, that constitute a recently rediscovered family of decidable fragments of first-order logic (FO), in which the order of quantification of variables coincides with the order in which those variables appear as arguments of predicates. While the complexities of their satisfiability problems are now well-established, their model theory, however, is poorly understood. Our paper aims to provide some insight into it.

We start by providing suitable notions of bisimulation for ordered logics. We next employ bisimulations to compare the relative expressive power of ordered logics, and to characterise our logics as bisimulation-invariant fragments of FO à la van Benthem.

Afterwards, we study the Craig Interpolation Property (CIP). We refute yet another claim from the infamous work by Purdy, by showing that the fluted and forward fragments do not enjoy CIP. We complement this result by showing that the ordered fragment and the guarded ordered logics enjoy CIP. These positive results rely on novel and quite intricate model constructions, which take full advantage of the "forwardness" of our logics.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases ordered fragments, fluted fragment, guarded fragment, model theory, Craig Interpolation Property, expressive power, model checking

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.15

Related Version *Full Version*: https://arxiv.org/abs/2206.11751

Funding *Bartosz Bednarczyk*: supported by the ERC Consolidator Grant No. 771779 (DeciGUT).

Acknowledgements We would like to thank Antti Kuusisto and Jean Christoph Jung for fruitful discussions on related topics, as well as Tim Lyon and Emanuel Kieroński for language corrections. We would also like to thank the anonymous reviewers for their very helpful comments.

1 Introduction

An ongoing research in computational logic has lead to discovery of new decidable fragments of first-order logics (FO) that extend modal and description logics. The main ideas that were proposed in the past involve: restricting the number of variables [8], relativised quantification [1, 26], restricted use of negation [24], relativised negation [3], one-dimensionality and uniformity [11], separateness [25] and ordered quantification [12, 22]. To compare aforementioned logics, the authors of [1, Section 4.7] proposed a list of desirable meta-properties of logic, which can serve as a yardstick to measure how "nice" a given logic is. We expect a logic L to

(A) be decidable and have the *Finite Model Property* (FMP),

(B) satisfy the Craig Interpolation Property (CIP), *i.e.* for any L-formulae φ, ψ such that $\varphi \models \psi$ there should be an L-formulae χ, called an *interpolant*, that uses only symbols appearing in the common vocabulary of φ and ψ, so that $\varphi \models \chi \models \psi$ holds,

(C) and to satisfy the analog of Łoś-Tarski Preservation Theorem (ŁTPT), *i.e.* any L-formula φ preserved under substructures should be equivalent to some universal L-formula.

It turned out that FO^2 and GF, example logics based on restricted number of variables and relativised quantification, are not "nice" as they do not enjoy CIP [17, Examples 1–2]. In contrast, UNFO and GNFO, the logics based on relativised negation, fulfil the properties (A)–(C), consult: [24, 2, 6]. For one-dimensionality, separateness and ordered quantification we have partial results only.

In this paper we take a closer look at logics enjoying ordered quantification, which have been receiving increasing attention recently [20, 4, 15]. Their syntax can be informally explained as follows. We first require that all variables appearing in formulae are additionally indexed by the quantifier depth and then impose a certain restriction on such numbers in variable sequences in atoms. Assuming that $\alpha(\overline{x})$ is in the scope of the n-th quantifier (but not the $(n+1)$-th), in the fluted fragment $\mathsf{L}_{\mathsf{suf}}$ of Quine [22] (resp. in the ordered fragment $\mathsf{L}_{\mathsf{pre}}$ by Herzig [12][1]) the tuple $\overline{x}$ is required to be a suffix (resp. a prefix) of the sequence $x_1, x_2, \ldots, x_n$. The forward fragment $\mathsf{L}_{\mathsf{inf}}$ [4] is more liberal and allows infixes in place of suffixes or prefixes. An example formula $\varphi \in (\mathsf{L}_{\mathsf{suf}} \cap \mathsf{L}_{\mathsf{inf}}) \setminus \mathsf{L}_{\mathsf{pre}}$ is given below:

1. No student admires every professor.

$$\forall x_1\ (\mathrm{student}(x_1) \to \neg\forall x_2\ (\mathrm{professor}(x_2) \to \mathrm{admires}(x_1, x_2)))$$

2. No lecturer introduces any professor to every student.

$$\forall x_1\ \mathrm{lecturer}(x_1) \to \neg\exists x_2\ [\mathrm{professor}(x_2) \land \forall x_3\ (\mathrm{student}(x_3) \to \mathrm{introduce}(x_1, x_2, x_3))]$$

Next, we provide a few coexamples, *i.e.* formulae that, as stated, do not belong to any of $\mathsf{L}_{\mathsf{inf}}, \mathsf{L}_{\mathsf{pre}}, \mathsf{L}_{\mathsf{suf}}$. The blue colour indicates a mismatch in the variable ordering.

1. The relation isPartOf is transitive.

$$\forall x_1\ \forall x_2\ \forall x_3\ \mathrm{isPartOf}(x_1, x_2) \land \mathrm{isPartOf}(x_2, x_3) \to \mathrm{isPartOf}(\underline{x_1, x_3})$$

2. A narcissist is a person who loves himself.

$$\forall x_1\ \mathrm{narcissist}(x_1) \to \mathrm{person}(x_1) \land \mathrm{loves}(\underline{x_1, x_1})$$

3. The binary relation hasChild is the inverse of the hasParent relation.

$$\forall x_1\ \forall x_2\ \mathrm{hasParent}(x_1, x_2) \leftrightarrow \mathrm{hasParent}(\underline{x_2, x_1})$$

All of $\mathsf{L}_{\mathsf{inf}}, \mathsf{L}_{\mathsf{suf}}, \mathsf{L}_{\mathsf{pre}}$ are decidable and have the Finite Model Property. Their satisfiability problem is, respectively, Tower-complete for $\mathsf{L}_{\mathsf{inf}}$ and $\mathsf{L}_{\mathsf{suf}}$, and PSpace-complete for $\mathsf{L}_{\mathsf{pre}}$. Somehow unexpectedly, the Tower-completeness of $\mathsf{L}_{\mathsf{suf}}$ was established only recently by Pratt-Hartmann et al. [20], after pointing out a mistake in the proof of the exponential-size model of $\mathsf{L}_{\mathsf{suf}}$ by Purdy [21] and disproving Purdy's claim of NExpTime-completeness of $\mathsf{L}_{\mathsf{suf}}$. The model theory of $\mathsf{L}_{\mathsf{inf}}, \mathsf{L}_{\mathsf{suf}}$, and $\mathsf{L}_{\mathsf{pre}}$ is, however, poorly understood. The only results that we are aware of are Purdy's claims that $\mathsf{L}_{\mathsf{suf}}$ has CIP [21, Thm. 14] and ŁTPT [21, Corr. 17]. But in the light of previously discovered errors, one should treat Purdy's paper with caution.

[1] Strictly speaking, the syntax of $\mathsf{L}_{\mathsf{pre}}$ is slightly more liberal than the original syntax of the ordered fragment as defined by Herzig, since the syntax of $\mathsf{L}_{\mathsf{pre}}$ allows requantifying variables.

1.1 Our results

This paper kick-starts a project of understanding the model theory of *ordered logics*, by which we mean the logics $\mathsf{L}_{\mathsf{pre}}, \mathsf{L}_{\mathsf{suf}}$, and $\mathsf{L}_{\mathsf{inf}}$ as well as their intersections with the guarded fragment GF [1], focusing on the problems mentioned in the introduction.

In Section 3, we design a suitable notion of bisimulations and compare the relative expressive power of ordered logics. Our proofs employ standard model-theoretic constructions like the Compactness Theorem and ω-saturated structures. Next, we investigate CIP in Section 4, which is the main technical contribution of the paper. First, we focus on interpolation for the fluted and the forward fragments. We show that, surprisingly, $\mathsf{L}_{\mathsf{inf}}$ and $\mathsf{L}_{\mathsf{suf}}$ do not enjoy CIP, refuting yet another claim from the infamous work of Purdy [21, Thm. 14]. Fortunately, other members of the family of ordered logics enjoy CIP, as shown in Sections 4.2–4.3. We stress here that standard techniques for proving CIP, *e.g.* those based on zig-zag products [19, 14, 2, 16], do not seem to work in our case.[2] This forces us to take a different route: we construct models explicitly by specifying types of tuples.

We believe that our proof methods, which are based on novel and intricate model-theoretic constructions, are very general. In particular, we believe that our CIP proof for guarded ordered logics can serve as a useful meta-technique (or even a heuristic) for (dis)proving CIP for fragments of GF. For instance, the proof can be adopted to fragments with CIP, deriving existing results (*e.g.* for the 2-variable GF [14] or the uniform one-dimensional GF [16]) and its failure gives hints why a certain fragment may not have CIP (*e.g.* in the case of full GF).

2 Preliminaries

Henceforth, we employ standard terminology from (finite and classical) model theory [18, 13]. All the logics considered here will be fragments of the first-order logic (FO) over purely-relational equality-free vocabularies, under the usual syntax and semantics.

We fix a countably infinite set of variables $\{x_i \mid i \in \mathbb{N}\}$ and throughout this paper all the formulas use only variables from this set. With $\mathsf{sig}(\varphi)$ we denote the set of relational symbols appearing in φ. We use $\mathsf{ar}(\mathrm{R})$ to denote the arity of R. For a logic L and a signature σ we use $\mathsf{L}[\sigma]$ in place of $\{\varphi \in \mathsf{L} \mid \mathsf{sig}(\varphi) \subseteq \sigma\}$. The k-variable fragment of L (*i.e.* employing only the variables $x_1, x_2, \ldots, x_k$) is denoted L^k. We write $\varphi(\overline{x})$ to indicate that all free variables from φ are members of $\overline{x}$. If $\overline{x}$ contains precisely the free variables of φ, then we will emphasise this separately. Given a structure $\mathfrak{A}$ and $B \subseteq A$, we will use $\mathfrak{A} \restriction B$ to denote the *substructure* of $\mathfrak{A}$ that B induces.

Tuples and subsequences. An n-tuple is a tuple with n elements. The 0-tuple is denoted with ϵ. We use $\overline{x}_{i\ldots j}$ to denote the $(j{-}i{+}1)$-tuple $x_i, x_{i+1}, \ldots, x_j$. We say that $\overline{x}_{i\ldots j}$ is an infix of a tuple $\overline{x}_{k\ldots l}$ if $k \leq i \leq j \leq l$ holds. If, in addition, $k = i$ (resp. $j = l$) we say that $\overline{x}_{i\ldots j}$ is a prefix (resp. suffix) of $\overline{x}_{k\ldots l}$. We use the word *affix* as a place-holder for the words *pre*fix, *suf*fix or *inf*ix. For a set S, we write $\overline{x} \sqsubseteq S$ iff $x_i \in S$ for all indices $1 \leq i \leq |\overline{x}|$, where $|\overline{x}|$ denotes the length of $\overline{x}$. A tuple $\overline{\mathrm{a}} \sqsubseteq A$ is *σ-live* in $\mathfrak{A}$ if $|\overline{\mathrm{a}}| \leq 1$ or $\overline{\mathrm{a}} \in \mathrm{R}^{\mathfrak{A}}$ for some $\mathrm{R} \in \sigma$.

Logics. We next introduce the logics $\mathsf{L}_{\mathsf{affix}} \in \{\mathsf{L}_{\mathsf{pre}}, \mathsf{L}_{\mathsf{suf}}, \mathsf{L}_{\mathsf{inf}}\}$. We start from $\mathsf{L}_{\mathsf{suf}}$, which for technical reasons we need to define separately from $\mathsf{L}_{\mathsf{pre}}$ and $\mathsf{L}_{\mathsf{inf}}$. For every $n \in \mathbb{N}$, we define the set $\mathsf{L}_{\mathsf{suf}}(n)$ as follows:

[2] For logics that are closed under negation on the level of formulas, zig-zag constructions seem to work only if the logics are *one-dimensional* and *uniform*, see [16] for more details. None of our logics are one-dimensional nor uniform.

- an atom $\alpha(\overline{x})$ is in $\mathsf{L}_{\mathsf{suf}}(n)$ if $\overline{x}$ is a suffix of $\overline{x}_{1\ldots n}$,
- $\mathsf{L}_{\mathsf{suf}}(n)$ is closed under Boolean connectives $\wedge, \vee, \neg, \rightarrow$,
- if φ is in $\mathsf{L}_{\mathsf{suf}}(n{+}1)$ then $\exists x_{n+1}\ \varphi$ and $\forall x_{n+1}\ \varphi$ are in $\mathsf{L}_{\mathsf{suf}}(n)$.

We put $\mathsf{L}_{\mathsf{suf}} := \mathsf{L}_{\mathsf{suf}}(0)$, which is exclusively composed of sentences.

To define the fragments $\mathsf{L} \in \{\mathsf{L}_{\mathsf{pre}}, \mathsf{L}_{\mathsf{inf}}\}$, for every $n \in \mathbb{N}$ we define the set of $\mathsf{L}(n)$ as follows:

- an atom $\alpha(\overline{x})$ is in $\mathsf{L}_{\mathsf{pre}}(n)$ if $\overline{x}$ is a prefix of $\overline{x}_{1\ldots n}$ and in $\mathsf{L}_{\mathsf{inf}}(n)$ if $\overline{x}$ is an infix of $\overline{x}_{1\ldots n}$.
- if $\varphi \in \mathsf{L}(n_1)$ and $\psi \in \mathsf{L}(n_2)$, then for all $n \geq \max\{n_1, n_2\}$ we have that $\neg\varphi, (\varphi \vee \psi), (\varphi \wedge \psi), (\varphi \rightarrow \psi)$ are in $\mathsf{L}(n)$.
- if φ is in $\mathsf{L}(n{+}1)$ then $\exists x_{n+1}\ \varphi$ and $\forall x_{n+1}\ \varphi$ are in $\mathsf{L}(n)$.

We set $\mathsf{L} := \mathsf{L}(0)$, which is exclusively composed of sentences. We stress that in contrast to $\mathsf{L}_{\mathsf{suf}}$, the logics $\mathsf{L} \in \{\mathsf{L}_{\mathsf{pre}}, \mathsf{L}_{\mathsf{inf}}\}$ allow us to requantify variables. We recommend the reader to employ the above definition to show that $\forall x_1 \forall x_2 \forall x_3 (\mathrm{R}(x_1 x_2 x_3) \rightarrow (\mathrm{A}(x_1) \wedge \exists x_2 \exists x_3 \mathrm{S}(x_1 x_2 x_3))) \in \mathsf{L}_{\mathsf{inf}}$.

Notice that if $\varphi(\overline{x}) \in \mathsf{L}_{\mathsf{affix}}(n)$, where $\overline{x}$ lists all the free variables of φ in order (with respect to their indices), then $\overline{x}$ is an affix of the tuple $\overline{x}_{1\ldots n}$. The logics $\mathsf{L}_{\mathsf{pre}}, \mathsf{L}_{\mathsf{suf}}$, and $\mathsf{L}_{\mathsf{inf}}$ were studied under the names of ordered [12], fluted [22], and forward [4] fragments. The *guarded* counterparts $\mathsf{G}_{\mathsf{affix}}$ of $\mathsf{L}_{\mathsf{affix}}$, are defined as the intersection of $\mathsf{L}_{\mathsf{affix}}$ and the guarded fragment GF [1], *i.e.* by imposing that blocks of quantifiers are relativised by atoms (recalled below). Abusing notation, we speak about all these logics collectively as *ordered logics.*

For reader's convenience we recall that GF is the smallest fragment of FO such that:

- Every atomic formula is in GF;
- GF is closed under boolean connectives $\wedge, \vee, \neg, \rightarrow$;
- If $\varphi(\overline{x}, \overline{y})$ is in GF and $\alpha(\overline{x}, \overline{y})$ is an atom containing all free variables of φ then both $\forall \overline{y}\ (\alpha(\overline{x}, \overline{y}) \rightarrow \varphi(\overline{x}, \overline{y}))$ and $\exists \overline{y}\ (\alpha(\overline{x}, \overline{y}) \wedge \varphi(\overline{x}, \overline{y}))$ are in GF;
- If $\varphi(x)$ has only a single free-variable x, then $\forall x\ \varphi$ and $\exists x\ \varphi$ are in GF.

The atoms α, appearing in the 3rd item of the above definition is called a *guard.*

For a finite signature σ and $n \in \mathbb{N}$, a (σ, n)*-affix-type* is a conjunction of atoms with n free variables $\overline{x}_{1\ldots n}$, in which for every $\mathrm{R} \in \sigma$ and every affix $\overline{x}_{l\ldots k}$ of $\overline{x}_{1\ldots n}$, of length $\mathsf{ar}(\mathrm{R})$, exactly one of $\mathrm{R}(\overline{x}_{l\ldots k})$, $\neg\mathrm{R}(\overline{x}_{l\ldots k})$ appears as a conjunct. For a σ-structure $\mathfrak{A}$ and a tuple $\bar{\mathrm{a}} \mathrel{\overline{\in}} A$ with $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{affix}}[\sigma]}(\bar{\mathrm{a}})$ we denote the *unique* $(\sigma, |\bar{\mathrm{a}}|)$-affix-type realised by $\bar{\mathrm{a}}$ in $\mathfrak{A}$.

2.1 Model Checking

Before jumping into the main part of the paper, we would like to point out some results on the combined complexity of model checking problems of ordered logics, since these seem to be missing from the literature. In what follows we will employ the *matrix encoding of structure*, that is a standard encoding in finite model theory [18, p. 88]. Given a $\{\mathrm{R}_1, \ldots, \mathrm{R}_m\}$-structure $\mathfrak{A}$ with a linearly-ordered domain A, by its *matrix encoding* we mean a binary string $\mathrm{menc}(\mathfrak{A}) := 0^n 1 \mathrm{menc}(\mathrm{R}_1) \ldots \mathrm{menc}(\mathrm{R}_m)$, where $\mathrm{menc}(\mathrm{R}_i)$ is a binary sequence of length $|A|^{\mathsf{ar}(\mathrm{R}_i)}$, in which the j-th bit is 1 iff the j-th tuple in the lexicographic ordering of $|A|^{\mathsf{ar}(\mathrm{R}_i)}$ belongs to $\mathrm{R}_i^{\mathfrak{A}}$.

The following theorem collects our complexity results. We have not tried to optimise the upper bounds for $\mathsf{G}_{\mathsf{pre}}$ and $\mathsf{L}_{\mathsf{pre}}$: it is quite possible that they can be improved further.

▶ **Theorem 1.** *Under the matrix encoding of structures, the combined complexity of the model-checking problem for a logic* L *is*

1. *decidable in* PTime *for* $\mathsf{G}_{\mathsf{pre}}$ *and* $\mathsf{L}_{\mathsf{pre}}$*,*
2. PTime*-complete for* $\mathsf{L} \in \{\mathsf{G}_{\mathsf{suf}}, \mathsf{G}_{\mathsf{inf}}, \mathsf{L}_{\mathsf{suf}}\}$*, and*
3. PSpace*-complete for* $\mathsf{L} = \mathsf{L}_{\mathsf{inf}}$*.*

Proof. The upper bound for $\mathsf{G}_{\mathsf{pre}}$ follows from the second item while the upper bound for $\mathsf{L}_{\mathsf{pre}}$ is proved in [5, App. A.1]. For the second item, the lower bound follows for all of the logics from the fact that they embed standard modal logic, for which the combined complexity is PTime-complete [10, Cor. 3.1.7]. For $\mathsf{G}_{\mathsf{suf}}$ and $\mathsf{G}_{\mathsf{inf}}$ matching upper bounds follow from the fact that the combined complexity of the guarded fragment is PTime-complete, while for $\mathsf{L}_{\mathsf{suf}}$ the matching upper bound is proved in [5, App. A.1]. Finally, for the third item, the upper bound follows from the fact that the combined complexity of FO is PSpace-complete [7], while the matching lower bound follows from the fact that $\mathsf{L}_{\mathsf{inf}}$ contains monadic FO, for which the combined complexity of model-checking is PSpace-complete [18, p. 99]. ◀

The matrix encoding is not the only natural way of encoding models. Another option would be to use the *list/database encoding* of models, where one essentially encodes relations by listing the tuples that they contain, as opposed to describing their adjacency matrices. It is easy to see that, if there is no bound on the arities of the relation symbols, then the list encoding of a model can be exponentially more succinct than its matrix encoding. Our proofs for the upper bounds of $\mathsf{L}_{\mathsf{pre}}$ and $\mathsf{L}_{\mathsf{suf}}$ are heavily dependent on the fact that we are using the matrix encoding of models, and hence it is conceivable that the complexities are higher if we are using list encoding.[3] We leave the related investigations as a very interesting future research direction.

3 Expressive power

We study the relative expressive power of ordered logics with a suitable notion of bisimulations.

▶ **Definition 2.** *A non-empty set* $\mathcal{Z} \subseteq \bigcup_{n<\omega}(A^n \times B^n)$ *is a* $\mathsf{L}_{\mathsf{affix}}[\sigma]$-bisimulation *between pointed structures* $\mathfrak{A}, \bar{\mathrm{a}}$ *and* $\mathfrak{B}, \bar{\mathrm{b}}$*, where* $|\bar{\mathrm{a}}| = |\bar{\mathrm{b}}|$*, if and only if* $(\bar{\mathrm{a}}, \bar{\mathrm{b}}) \in \mathcal{Z}$ *and for all* $(\bar{\mathrm{c}}, \bar{\mathrm{d}}) \in \mathcal{Z}$ *the following conditions hold:*

(atomic harmony) $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{affix}}[\sigma]}(\bar{\mathrm{c}}) = \mathsf{tp}_{\mathfrak{B}}^{\mathsf{L}_{\mathsf{affix}}[\sigma]}(\bar{\mathrm{d}})$.

(forth) *For a (possibly empty) affix* $\bar{\mathrm{c}}_{i\ldots j}$ *of* $\bar{\mathrm{c}}$ *and* $\mathrm{e} \in A$ *there is* $\mathrm{f} \in B$ *s.t.* $(\bar{\mathrm{c}}_{i\ldots j}\mathrm{e}, \bar{\mathrm{d}}_{i\ldots j}\mathrm{f}) \in \mathcal{Z}$.

(back) *For a (possibly empty) affix* $\bar{\mathrm{d}}_{i\ldots j}$ *of* $\bar{\mathrm{d}}$ *and* $\mathrm{f} \in B$ *there is* $\mathrm{e} \in A$ *s.t.* $(\bar{\mathrm{c}}_{i\ldots j}\mathrm{d}, \bar{\mathrm{d}}_{i\ldots j}\mathrm{f}) \in \mathcal{Z}$.

For $\mathsf{G}_{\mathsf{affix}}$, we replace the conditions (forth), (back) by their guarded counterparts:

(gforth) For a (possibly empty) affix $\bar{\mathrm{c}}_{i\ldots j}$ of $\bar{\mathrm{c}}$ and a σ-live tuple $\bar{\mathrm{e}}$ in $\mathfrak{A}$ such that $\bar{\mathrm{c}}_{i\ldots j} = \bar{\mathrm{e}}_{1\ldots j-i+1}$ there is a σ-live tuple $\bar{\mathrm{f}}$ with $\bar{\mathrm{d}}_{i\ldots j} = \bar{\mathrm{f}}_{1\ldots j-i+1}$ and $(\bar{\mathrm{e}}, \bar{\mathrm{f}}) \in \mathcal{Z}$,

(gback) For a (possibly empty) affix $\bar{\mathrm{d}}_{i\ldots j}$ of $\bar{\mathrm{d}}$ and a σ-live tuple $\bar{\mathrm{f}}$ in $\mathfrak{B}$ such that $\bar{\mathrm{d}}_{i\ldots j} = \bar{\mathrm{f}}_{1\ldots j-i+1}$ there is a σ-live tuple $\bar{\mathrm{e}}$ with $\bar{\mathrm{c}}_{i\ldots j} = \bar{\mathrm{e}}_{1\ldots j-i+1}$ and $(\bar{\mathrm{e}}, \bar{\mathrm{f}}) \in \mathcal{Z}$,

For a logic L and a finite signature σ, we write $\mathfrak{A} \equiv_{\mathsf{L}[\sigma]} \mathfrak{B}$ if $\mathfrak{A}$ and $\mathfrak{B}$ satisfy the same $\mathsf{L}[\sigma]$-sentences, and we write $\mathfrak{A} \sim_{\mathsf{L}[\sigma]} \mathfrak{B}$ if there is an $\mathsf{L}[\sigma]$-bisimulation between $\mathfrak{A}$ and $\mathfrak{B}$. If $|\bar{\mathrm{a}}| = |\bar{\mathrm{b}}|$, we use $\mathfrak{A}, \bar{\mathrm{a}} \equiv_{\mathsf{L}[\sigma]} \mathfrak{B}, \bar{\mathrm{b}}$ to denote that for every (possibly empty) affix $\bar{\mathrm{a}}_{i\ldots j}$ of $\bar{\mathrm{a}}$ and $\varphi(\bar{x}_{i\ldots j}) \in \mathsf{L}[\sigma]$, where $\bar{x}_{i\ldots j}$ is an affix of $(x_1, \ldots, x_n)$, we have that $\mathfrak{A} \models \varphi(\bar{\mathrm{a}}_{i\ldots j})$ if and only if $\mathfrak{B} \models \varphi(\bar{\mathrm{b}}_{i\ldots j})$. For the next lemma consult [5, App. B.1]

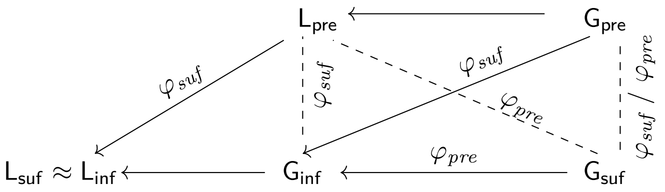
▶ **Lemma 3.** *Let* $\mathsf{L} \in \{\mathsf{L}_{\mathsf{affix}}, \mathsf{G}_{\mathsf{affix}}\}$*. Then* $\mathfrak{A}, \bar{\mathrm{a}} \sim_{\mathsf{L}[\sigma]} \mathfrak{B}, \bar{\mathrm{b}}$ *implies* $\mathfrak{A}, \bar{\mathrm{a}} \equiv_{\mathsf{L}[\sigma]} \mathfrak{B}, \bar{\mathrm{b}}$*. The converse holds over* ω*-saturated* $\mathfrak{A}$ *and* $\mathfrak{B}$*.*

[3] They can not decrease, because a list encoding of a model can always be constructed efficiently from its matrix encoding.

A logic L_2 is *at least as expressive as* a logic L_1 (written $\mathsf{L}_1 \preceq \mathsf{L}_2$) if for all $\varphi \in \mathsf{L}_1$ there is a $\psi \in \mathsf{L}_2$ such that $\varphi \equiv \psi$. We write $\mathsf{L}_1 \approx \mathsf{L}_2$ iff $\mathsf{L}_1 \preceq \mathsf{L}_2$ and $\mathsf{L}_2 \preceq \mathsf{L}_1$. In case $\mathsf{L}_1 \not\preceq \mathsf{L}_2$ and $\mathsf{L}_2 \not\preceq \mathsf{L}_1$ we call L_1 and L_2 incomparable. Lastly, $\mathsf{L}_1 \prec \mathsf{L}_2$ denotes that L_2 is *strictly more expressive* than L_1, *i.e.* $\mathsf{L}_1 \preceq \mathsf{L}_2$ and $\mathsf{L}_1 \not\approx \mathsf{L}_2$. Note that, by definition, all the considered fragments L satisfy $\mathsf{L} \preceq \mathsf{L}_{\mathsf{inf}}$ and $\mathsf{L} \prec \mathsf{FO}$ (every such L is decidable). Moreover, $\mathsf{G}_{\mathsf{affix}} \prec \mathsf{L}_{\mathsf{affix}}$ is a consequence of $\forall x_1 \forall x_2 \mathrm{R}(x_1, x_2)$ not being $\mathsf{GF}[\{\mathrm{R}\}]$-definable (which is well-known and follows from the fact that GF has the tree-model property). Our results are as follows:

▶ **Theorem 4.** *(a)* $\mathsf{L}_{\mathsf{pre}} \prec \mathsf{L}_{\mathsf{suf}} \approx \mathsf{L}_{\mathsf{inf}} \prec \mathsf{FO}$*, (b)* $\mathsf{G}_{\mathsf{affix}} \prec \mathsf{L}_{\mathsf{affix}}$ *for all affixes, (c)* $\mathsf{G}_{\mathsf{suf}} \prec \mathsf{G}_{\mathsf{inf}}$*, (d)* $\mathsf{G}_{\mathsf{pre}} \prec \mathsf{G}_{\mathsf{inf}}$*, and (e) otherwise the logics are incomparable.*

Proof. Full proofs are in [5, App. B.2]. The relationships between different logics with separating examples (omitting trivial examples due to guardedness) are depicted below. With φ_{pre} we denote the formula $\forall x_1 x_2 x_3\ \mathrm{R}(x_1 x_2 x_3) \rightarrow \mathrm{S}(x_1 x_2)$, while φ_{suf} denotes $\forall x_1 x_2 x_3\ \mathrm{R}(x_1 x_2 x_3) \rightarrow \mathrm{T}(x_2 x_3)$. Solid (resp. dashed) arrows from L_1 to L_2 denote that $\mathsf{L}_1 \prec \mathsf{L}_2$ holds (resp. that the logics are incomparable).


$\mathsf{L}_{\mathsf{pre}}$
$\mathsf{G}_{\mathsf{pre}}$
φ_{suf}
φ_{suf}
φ_{suf}
φ_{pre}
φ_{suf} / φ_{pre}
φ_{pre}
$\mathsf{L}_{\mathsf{suf}} \approx \mathsf{L}_{\mathsf{inf}}$
$\mathsf{G}_{\mathsf{inf}}$
$\mathsf{G}_{\mathsf{suf}}$

The equi-expressivity of $\mathsf{L}_{\mathsf{inf}}$ and $\mathsf{L}_{\mathsf{suf}}$ is an easy observation: we turn each maximally nested subformulae into DNF and push the atoms violating the definition of $\mathsf{L}_{\mathsf{affix}}$ outside. ◀

Knowing the relative expressive power of our logics, we would like to characterise them as bisimulation-invariant fragments of FO, as was done with other decidable logics, see *e.g.* [9]. Given a formula $\varphi(\overline{x}) \in \mathsf{L}$, we say that it is $\sim_{\mathsf{L}}$-invariant iff for all $\mathfrak{A}, \overline{\mathrm{a}} \sim_{\mathsf{L}}^{\mathsf{sig}(\varphi)} \mathfrak{B}, \overline{\mathrm{b}}$ we have $\mathfrak{A} \models \varphi(\overline{\mathrm{a}}) \Leftrightarrow \mathfrak{B} \models \varphi(\overline{\mathrm{b}})$. L is $\sim_{\mathsf{L}}$-invariant iff all its formulae are $\sim_{\mathsf{L}}$-invariant. We will next show that $\mathsf{L}_{\mathsf{affix}}$ (resp. $\mathsf{G}_{\mathsf{affix}}$) are exactly the $\sim_{\mathsf{L}_{\mathsf{affix}}}$- (resp. $\sim_{\mathsf{G}_{\mathsf{affix}}}$-) invariant fragments of FO. This confirms that our notion of bisimulation is the right one.

▶ **Theorem 5.** *Let* $\mathsf{L} \in \{\mathsf{L}_{\mathsf{affix}}, \mathsf{G}_{\mathsf{affix}}\}$ *and let* $\varphi(\overline{x})$ *be a* $\sim_{\mathsf{L}}$*-invariant* FO *formula. Then there exists a formula* $\psi(\overline{x})$ *in* L *which is equivalent with* $\varphi(\overline{x})$*.*

Proof. We follow standard proof methods, see *e.g.* [2, Thm. 3.2]. Suppose $\varphi(x_1, \ldots, x_n) \in \mathsf{FO}$ is $\sim_{\mathsf{L}}$-invariant, where $\overline{x} = (x_1, \ldots, x_n)$ enumerates precisely the set of free variables of φ. The case when φ is unsatisfiable φ is trivial, thus assume otherwise. Consider the set $\Gamma := \{\chi(\overline{x}_{i\ldots j}) \in \mathsf{L} \mid \varphi(\overline{x}) \models \chi(\overline{x}_{i\ldots j})\}$. Clearly $\varphi(\overline{x}) \models \Gamma$. Since FO is compact, it suffices to show that $\Gamma \models \varphi(\overline{x})$. Let $\mathfrak{A}$ be a structure and $\overline{\mathrm{a}} \in A^n$ so that $\mathfrak{A} \models \chi(\overline{\mathrm{a}}_{i\ldots j})$, for every $\chi(\overline{x}_{i\ldots j}) \in \Gamma$. Next, consider the set $\Sigma := \{\chi(\overline{x}_{i\ldots j}) \in \mathsf{L} \mid \mathfrak{A} \models \chi(\overline{\mathrm{a}}_{i\ldots j})\}$. Again, by compactness of FO we can show that $\Sigma \cup \{\varphi\}$ is consistent. Take a structure $\mathfrak{B}$ and $\overline{\mathrm{b}} \in B^n$ so that $\mathfrak{B} \models \varphi(\overline{\mathrm{b}})$ and $\mathfrak{B} \models \chi(\overline{\mathrm{b}}_{i\ldots j})$, for every $\chi(\overline{x}_{i\ldots j}) \in \Sigma$. Observe that by construction $\mathfrak{A}, \overline{\mathrm{a}} \equiv_{\mathsf{L}} \mathfrak{B}, \overline{\mathrm{b}}$. Replacing $\mathfrak{A}$ and $\mathfrak{B}$ with their ω-saturated elementary extensions $\hat{\mathfrak{A}}$ and $\hat{\mathfrak{B}}$, we know by Lemma 3 that $\hat{\mathfrak{A}}, \overline{\mathrm{a}} \sim_{\mathsf{L}} \hat{\mathfrak{B}}, \overline{\mathrm{b}}$. Chasing the resulting diagram we get $\mathfrak{A} \models \varphi(\overline{\mathrm{a}})$. ◀

4 Craig Interpolation

Recall that the Craig Interpolation Property (CIP) for a logic L states that if $\varphi(\overline{x}) \models \psi(\overline{x})$ holds (with φ and ψ having the same free variables), then there is a $\chi(\overline{x}) \in \mathsf{L}[\mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)]$ (an L-*interpolant*) such that $\varphi(\overline{x}) \models \chi(\overline{x})$ and $\chi(\overline{x}) \models \psi(\overline{x})$ hold. We always assume that both φ and ψ are satisfiable, otherwise we can take $\bot$ as a trivial interpolant.

To reason about interpolants we employ the notion of *joint consistency* [23]. We say that L-formulae $\varphi(\overline{x}_{1\ldots n})$ and $\psi(\overline{x}_{1\ldots n})$ (having exactly $\overline{x}_{1\ldots n}$ free) are *jointly-*$\mathsf{L}[\tau]$*-consistent* (or just *jointly consistent* in case $\tau := \mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)$ and L are known from the context), if there are structures $\mathfrak{A} \models \varphi(\overline{\mathrm{a}})$ and $\mathfrak{B} \models \psi(\overline{\mathrm{b}})$ such that $\mathfrak{A}, \overline{\mathrm{a}} \sim_{\mathsf{L}[\tau]} \mathfrak{B}, \overline{\mathrm{b}}$. The next lemma is classic and links joint consistency and interpolation: see [5, App. C.1].

▶ **Lemma 6.** *Let* $\mathsf{L} \subseteq \mathsf{FO}$*, and let* $\varphi(\overline{x}), \psi(\overline{x}) \in \mathsf{L}$ *with* $\tau := \mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)$*. Then* $\varphi(\overline{x})$ *and* $\neg\psi(\overline{x})$ *are jointly consistent iff there is no* $\mathsf{L}[\tau]$*-interpolant for* $\varphi(\overline{x}) \models \psi(\overline{x})$*.*

We simplify the reasoning about ordered logics by employing suitable normal forms. We say that a formula $\varphi(\overline{x})$ from[4] $\mathsf{L}_{\mathsf{pre}}$ (resp. from $\mathsf{G}_{\mathsf{affix}}$) is in *normal form* if it has the shape:
(NForm-$\mathsf{L}_{\mathsf{pre}}$) $\mathrm{H}(\overline{x}) \wedge \bigwedge_{i=1}^{s} \forall\overline{x}_{1\ldots\ell_i}(\alpha_i \rightarrow \exists x_{\ell_i+1}\beta_i) \wedge \bigwedge_{j=1}^{t} \forall\overline{x}_{1\ldots\ell_j}(\alpha_j \rightarrow \forall x_{\ell_j+1}\beta_j)$,
(NForm-$\mathsf{G}_{\mathsf{affix}}$) $\mathrm{H}(\overline{x}) \wedge \bigwedge_{i=1}^{s} \forall\overline{x}_{1\ldots\ell_i}(\mathrm{R}_i(\overline{x}_{1\ldots\ell_i}) \rightarrow \exists\overline{x}_{\ell_i+1\ldots\ell_i+k_i}(\mathrm{S}_i(\overline{x}_{1\ldots\ell_i+k_i}) \wedge \psi_i(\overline{x}_{1\ldots\ell_i+k_i}))) \wedge \bigwedge_{j=1}^{t} \forall\overline{x}_{1\ldots\ell_j}(\mathrm{R}_j(\overline{x}_{1\ldots\ell_j}) \rightarrow \psi_j(\overline{x}_{1\ldots\ell_j}) \rightarrow \forall\overline{x}_{\ell_j+1\ldots\ell'_j}(\mathrm{T}_j(\overline{x}_{1\ldots\ell'_j}) \rightarrow \psi'_j(\overline{x}_{1\ldots\ell'_j})))$,
where $\alpha_i, \alpha_j, \beta_i$ and β_j are quantifier-free $\mathsf{L}_{\mathsf{pre}}$-formulae, $\mathrm{R}_i, \mathrm{R}_j$, T_j and H are relational symbols, and ψ_i, ψ_j and ψ'_j are $\mathsf{G}_{\mathsf{affix}}$-formulae. The symbol H is called the *head* of $\varphi(\overline{x})$. We will often speak about *existential/universal requirements* of a formula, meaning the appropriate subformulae with the maximal quantifier prefix $\forall^*\exists^*$ and $\forall^*$. In aforementioned normal forms we implicitly allow various parameters to be zero, *e.g.* in subformulae of the form $\forall\overline{x}_{1\ldots\ell_i}(\alpha_i \rightarrow \exists x_{\ell_i+1}\beta_i)$ we allow $\ell_i = 0$, and we agree that the result is $\exists x_{\ell_i+1}\beta_i$.

The following lemma can be shown using standard renaming techniques, in complete analogy to [4, 15], with a minor (but technically tedious) modification in the case of $\mathsf{G}_{\mathsf{suf}}$, see [5, App. C.2].

▶ **Lemma 7.** *Let* $\mathsf{L} \in \{\mathsf{L}_{\mathsf{pre}}, \mathsf{G}_{\mathsf{affix}}\}$*, and take* $\varphi(\overline{x}), \psi(\overline{x}) \in \mathsf{L}$*. Suppose that there are models* $\mathfrak{A}$ *and* $\mathfrak{B}$ *such that* $\mathfrak{A} \models \varphi(\overline{\mathrm{a}})$*,* $\mathfrak{B} \models \psi(\overline{\mathrm{b}})$ *and* $\mathfrak{A}, \overline{\mathrm{a}} \sim_{\mathsf{L}[\tau]} \mathfrak{B}, \overline{\mathrm{b}}$*, where* $\tau = \mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)$*. Then there exist formulae* $\varphi'(\overline{x}), \psi'(\overline{x}) \in \mathsf{L}$ *in normal form and extensions* $\mathfrak{A}'$ *and* $\mathfrak{B}'$ *of* $\mathfrak{A}$ *and* $\mathfrak{B}$ *respectively, such that (i)* $\varphi'(\overline{x})$ *and* $\psi'(\overline{x})$ *have the same head* H*, (ii)* $\mathsf{sig}(\varphi') \cap \mathsf{sig}(\psi') = \tau \cup \{\mathrm{H}\}$*, (iii)* $\varphi'(\overline{x}) \models \varphi(\overline{x})$ *and* $\psi'(\overline{x}) \models \psi(\overline{x})$*, and (iii)* $(\mathfrak{A}', \overline{\mathrm{a}}) \sim_{\mathsf{L}[\tau\cup\{\mathrm{H}\}]} (\mathfrak{B}', \overline{\mathrm{b}})$ *holds.*

The following lemma is a useful tool when dealing with interpolation, allowing us to switch our attention to a certain satisfiability problem. Its proof is routine, consult [5, App. C.3].

▶ **Lemma 8.** *Let* $\mathsf{L} \in \{\mathsf{L}_{\mathsf{pre}}, \mathsf{G}_{\mathsf{affix}}\}$*. If for any jointly-consistent* L*-formulae* $\varphi(\overline{x}), \psi(\overline{x})$ *in normal forms from Lemma 7 with the same head, there is* $\mathfrak{U} \models \varphi(\overline{x}) \wedge \psi(\overline{x})$*, then* L *has CIP.*

4.1 Disproving CIP in $\mathsf{L}_{\mathsf{inf}}$ and $\mathsf{L}_{\mathsf{suf}}$

We start our investigation of CIP for $\mathsf{L}_{\mathsf{affix}}$ and $\mathsf{G}_{\mathsf{affix}}$ by further discrediting the infamous work of Purdy [21]. We prove, in stark contrast to [21, Thm. 14], that $\mathsf{L}_{\mathsf{suf}}$ does not have CIP.

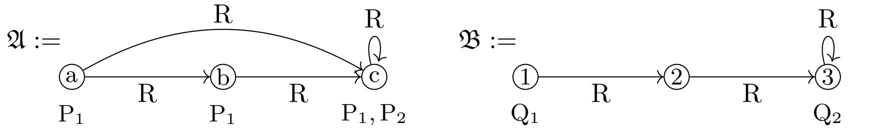
▶ **Theorem 9.** $\mathsf{L}_{\mathsf{inf}}$ *and* $\mathsf{L}_{\mathsf{suf}}$ *do not have CIP. More specifically, there are* $\mathsf{L}^2_{\mathsf{suf}}$*-sentences* φ, ψ *with* $\varphi \models \psi$ *but without any* $\mathsf{L}_{\mathsf{inf}}[\mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)]$*-interpolant.*

Proof. Consider the following $\mathsf{L}^3_{\mathsf{inf}}$-sentences φ and ψ, presented respectively below:

$$\forall\overline{x}_{1\ldots3}[(\mathrm{R}(x_1,x_2) \wedge \mathrm{R}(x_2,x_3)) \rightarrow (\mathrm{P}_1(x_1) \wedge \mathrm{P}_2(x_3))] \;\wedge\; \forall x_1\forall x_2[(\mathrm{P}_1(x_1) \wedge \mathrm{P}_2(x_2)) \rightarrow \mathrm{R}(x_1,x_2)]$$
$$\exists\overline{x}_{1\ldots3}[\mathrm{R}(x_1,x_2) \wedge \mathrm{R}(x_2,x_3) \wedge \mathrm{Q}_1(x_1) \wedge \mathrm{Q}_2(x_3)] \;\wedge\; \forall x_1\forall x_2[(\mathrm{Q}_1(x_1) \wedge \mathrm{Q}_2(x_2)) \rightarrow \neg\mathrm{R}(x_1,x_2)],$$

[4] To avoid notational glitter we will be a bit careless when dealing with formulae with free-variables.

with $\mathfrak{A} \models \varphi$ iff $(\mathrm{R}^{\mathfrak{A}} \circ \mathrm{R}^{\mathfrak{A}}) \subseteq \mathrm{P}_1^{\mathfrak{A}} \times \mathrm{P}_2^{\mathfrak{A}}$ and $\mathrm{P}_1^{\mathfrak{A}} \times \mathrm{P}_2^{\mathfrak{A}} \subseteq \mathrm{R}^{\mathfrak{A}}$, and $\mathfrak{B} \models \psi$ iff $\mathrm{Q}_1^{\mathfrak{B}} \times \mathrm{Q}_2^{\mathfrak{B}} \subseteq B^2 \setminus \mathrm{R}^{\mathfrak{B}}$ and there are $(\mathrm{a}, \mathrm{b}), (\mathrm{b}, \mathrm{c}) \in \mathrm{R}^{\mathfrak{B}}$ with $\mathrm{a} \in \mathrm{Q}_1^{\mathfrak{B}}$ and $\mathrm{c} \in \mathrm{Q}_2^{\mathfrak{B}}$. Observe that $\varphi \models \neg\psi$, since φ entails transitivity of R, while ψ entails that this is not the case. But φ and ψ are jointly-$\mathsf{L}_{\mathsf{inf}}[\{\mathrm{R}\}]$-consistent (it suffices to take $\mathfrak{A}$ and $\mathfrak{B}$ depicted below, *cf.* [5, App. C.5].



Hence, by Lemma 6 there is no $\mathsf{L}_{\mathsf{inf}}[\{\mathrm{R}\}]$-interpolant for $\varphi \models \neg\psi$. By slightly obfuscating φ and ψ (*i.e.* by shifting quantifiers and introducing a unary symbol to get rid of the third variable) we can make our counterexample formulae to be in $\mathsf{L}^2_{\mathsf{suf}}$; consult [5, App. C.6]. ◀

We left open the question whether $\mathsf{L}_{\mathsf{suf}}$ and $\mathsf{L}_{\mathsf{inf}}$ have the *Projective Beth Definability Property* (PBDP). We conjuncture that the answer is *no*, but we haven't found a suitable example yet.

4.2 Restoring CIP in $\mathsf{L}_{\mathsf{pre}}$

Even though $\mathsf{L}_{\mathsf{suf}}$ and $\mathsf{L}_{\mathsf{inf}}$ fail to have CIP, it turns out that $\mathsf{L}_{\mathsf{pre}}$ still has it. To prove interpolation for $\mathsf{L}_{\mathsf{pre}}$, we are going to construct a model for two jointly consistent $\mathsf{L}_{\mathsf{pre}}$ formulae $\varphi(\overline{x})$ and $\psi(\overline{x})$. However, rather than modifying existing amalgamation-based arguments used, for instance, in [19, 14, 2], we will construct our model explicitly by specifying prefix-types for tuples. We feel that our approach, which is more direct in nature than other arguments found in the literature, could potentially be useful also in other contexts.

Take $\varphi(\overline{x})$ and $\psi(\overline{x})$ in normal form (NForm-$\mathsf{L}_{\mathsf{pre}}$) satisfying the premise of Lemma 8. Hence, there are structures $\mathfrak{A}$ and $\mathfrak{B}$ and tuples $\overline{\mathrm{a}} \in A^k$ and $\overline{\mathrm{b}} \in B^k$ such that that $\mathfrak{A} \models \varphi(\overline{\mathrm{a}}), \mathfrak{B} \models \psi(\overline{\mathrm{b}})$ and $(\mathfrak{A}, \overline{\mathrm{a}}) \sim_{\mathsf{L}_{\mathsf{pre}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{b}})$, where $\sigma := \mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)$. Let $\tau := \mathsf{sig}(\varphi) \cup \mathsf{sig}(\psi)$.

We will define a sequence of τ-structures $\mathfrak{U}_1 \leq \ldots \leq \mathfrak{U}_M := \mathfrak{U}$, where $M = \max\{\mathsf{ar}(\mathrm{R}) \mid \mathrm{R} \in \tau\}$, satisfying the following inductive assumptions: (i) $U_i = \mathbb{N}$, (ii) the interpretation of symbols from τ of arity $> i$ is empty, and (iii) for any i-tuple $\overline{\mathrm{c}}$ in $\mathfrak{U}_i$ there are i-tuples $\overline{\mathrm{d}}$ in $\mathfrak{A}$ and $\overline{\mathrm{e}}$ in $\mathfrak{B}$ so that $(\mathfrak{A}, \overline{\mathrm{d}}) \sim_{\mathsf{L}_{\mathsf{pre}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{e}})$ and $\mathsf{tp}_{\mathfrak{U}_i}^{\mathsf{L}_{\mathsf{pre}}[\tau]}(\overline{\mathrm{c}}) = \mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{d}}) \cup \mathsf{tp}_{\mathfrak{B}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\psi)]}(\overline{\mathrm{e}})$ hold. The last condition guarantees that no tuple $\overline{\mathrm{c}}$ of $\mathfrak{U}_i$ violates the universal requirements of φ and ψ, since otherwise the corresponding tuple would violate them, contradicting modelhood of $\mathfrak{A}$ or $\mathfrak{B}$.[5]

For the inductive base, take $\mathfrak{U}_1$ with domain $\mathbb{N}$ and empty interpretation of symbols from τ. Our goal is to realise each $(\mathsf{sig}(\varphi), 1)$-prefix-type, which is realised in $\mathfrak{A}$ and $\mathfrak{B}$, in $\mathfrak{U}_1$ in a careful way, suggested by the inductive assumption. Let t be a $(\mathsf{sig}(\varphi), 1)$-prefix type realised in $\mathfrak{A}$ and let $\mathrm{c} \in A$ be some element witnessing it. Since $(\mathfrak{A}, \overline{\mathrm{a}}) \sim_{\mathsf{L}_{\mathsf{pre}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{b}})$ holds, there exists an element d of $\mathfrak{B}$ so that $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{pre}}[\sigma]}(\mathrm{c}) = \mathsf{tp}_{\mathfrak{B}}^{\mathsf{L}_{\mathsf{pre}}[\sigma]}(\mathrm{d})$. Now we will assign the $(\tau, 1)$-prefix-type $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\varphi)]}(\mathrm{c}) \cup \mathsf{tp}_{\mathfrak{B}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\psi)]}(\mathrm{d})$ to some element e of $\mathfrak{U}_1$, for which we have not yet assigned a $(\tau, 1)$-prefix-type. For the remaining elements of $\mathfrak{U}_1$, having no $(\tau, 1)$-prefix-type assigned, we assign any of the previously realised types.

[5] We note that this claim no longer holds if $\mathsf{L}_{\mathsf{pre}}$ is replaced by either $\mathsf{L}_{\mathsf{suf}}$ or $\mathsf{L}_{\mathsf{affix}}$, which is why the forthcoming construction does not work for these logics.

Suppose then that $\mathfrak{U}_k$ is defined. To define $\mathfrak{U}_{k+1}$, we will start by providing witnesses for the existential requirements of φ and ψ; since the two cases are rather analogous, we will restrict our attention to the former case. Consider an existential requirement $\varphi_i^{\exists}$ of $\varphi(\overline{x})$ and let $\overline{\mathrm{e}} \in U_k^k$ be a k-tuple so that $\mathfrak{U} \models \alpha_i(\overline{\mathrm{e}})$. By construction, there exists a tuple $\overline{\mathrm{a}} \in A^k$ witnessing $\mathsf{tp}_{\mathfrak{U}_k}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{e}}) = \mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{a}})$. Since $\mathfrak{A} \models \varphi_i^{\exists}$, there exists an element $\mathrm{c} \in A$ so that $\mathfrak{A} \models \beta_i(\overline{\mathrm{a}}, \mathrm{c})$. Due to $(\mathfrak{A}, \overline{\mathrm{a}}) \sim_{\mathsf{L}_{\mathsf{pre}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{b}})$, we know that there exists an element $\mathrm{d} \in B$ satisfying $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{pre}}[\sigma]}(\overline{\mathrm{a}}, \mathrm{c}) = \mathsf{tp}_{\mathfrak{B}}^{\mathsf{L}_{\mathsf{pre}}[\sigma]}(\overline{\mathrm{b}}, \mathrm{d})$. Now we pick an element $f \in U$ for which we have not yet assigned a $(\tau, k{+}1)$-prefix-type for the tuple $(\overline{\mathrm{e}}, \mathrm{f})$ (recall that the domain of our model is $\mathbb{N}$, so such an element always exists). We assign the following $(\tau, k{+}1)$-prefix-type to the tuple $(\overline{\mathrm{e}}, \mathrm{f})$: $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\varphi)]}((\overline{\mathrm{a}}, \mathrm{c})) \cup \mathsf{tp}_{\mathfrak{B}}^{\mathsf{L}_{\mathsf{pre}}[\mathsf{sig}(\psi)]}((\overline{\mathrm{b}}, \mathrm{d}))$. Note that the assigned $(\tau, k{+}1)$-prefix-type is consistent with the (τ, k)-prefix-type that we assigned to $\overline{\mathrm{e}}$. Having assigned witnesses to relevant existential requirements of $\varphi(\overline{x})$ and $\psi(\overline{x})$, there are still $(k{+}1)$-tuples of elements of $\mathfrak{U}$ for which we have not yet assigned a $(\tau, k{+}1)$-prefix-type. For those tuples we will assign any $(\tau, k{+}1)$-prefix-type that we have already assigned to some other $(k{+}1)$-tuple of elements of $\mathfrak{U}_{k+1}$. This completes the construction of $\mathfrak{U}_{k+1}$.

By construction, it is clear that there exists a tuple $\overline{\mathrm{e}}$ of elements of $\mathfrak{U}$ so that $\overline{\mathrm{e}} \in \mathrm{H}^{\mathfrak{U}}$; in particular, $\mathfrak{U} \models \varphi(\overline{\mathrm{e}}) \wedge \psi(\overline{\mathrm{e}})$ holds. Thus, by Lemma 8 we conclude:

▶ **Theorem 10.** $\mathsf{L}_{\mathsf{pre}}$ *enjoys the Craig Interpolation Property.*

4.3 Restoring CIP in guarded logics

Finally we turn our attention to the logics $\mathsf{G}_{\mathsf{affix}}$ and present the main contribution of the paper. It will be convenient to employ suitable tree-like models. Intuitively, HATs [4] are just trees in which relations connect elements but only in a level-by-level ascending order; see Figure 1. HAHs are collections of HATs.

▶ **Definition 11.** *A structure* $\mathfrak{T}$ *is a* higher-arity tree *(HAT) if its domain is a prefix-closed subset of sequences from* $\mathbb{N}^*$ *and for all relation symbols* R *we have that* $(d_1, \ldots, d_k) = \overline{\mathrm{d}} \in \mathrm{R}^{\mathfrak{T}}$ *implies that for each index* $i < k$ *there exists a number* n_i *such that* $\mathrm{d}_{i+1} = \mathrm{d}_i \cdot n_i$*, where* $\mathrm{d}_i \cdot n_i$ *means that the element* n_i *is appended to the sequence* d_i*. A structure* $\mathfrak{H}$ *is a* higher-arity hedge *(HAH) if* $\mathfrak{H}$ *becomes a HAT if extended by a single element* ε*.*

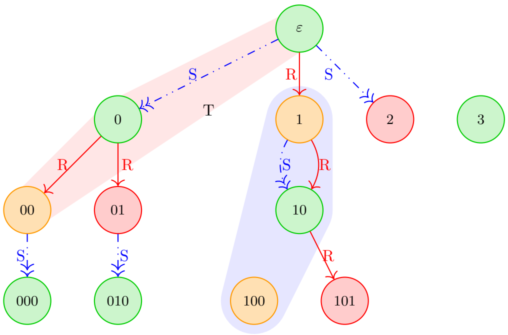


Figure 1 An example HAT $\mathfrak{T}$. All relations go down lvl-by-lvl. The red area means $(\varepsilon, 0, 00) \in \mathrm{T}^{\mathfrak{T}}$.

By a subtree of a HAT $\mathfrak{T}$ rooted at an element d we mean a substructure of $\mathfrak{T}$ with the domain composed of all elements of the form dw for a possibly empty word w. Note that such a subtree is also a HAT after an obvious renaming.

We are going to employ the following lemma, stating that for our purposes we can focus on tree-like models only. Its proof relies on the suitable notion of unravelling, see [5, App. C.4].

▶ **Lemma 12.** *Let a logic* L *be any of* $\mathsf{G}_{\mathsf{affix}}$*,* φ, ψ *be* L*-formulae and* $\sigma := \mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)$ *containing the predicate* H*. Assume that models* $\mathfrak{A} \models \varphi(\overline{\mathrm{a}}), \mathfrak{B} \models \psi(\overline{\mathrm{b}})$ *are given such that* $\overline{\mathrm{a}} \in \mathrm{H}^{\mathfrak{A}}, \overline{\mathrm{b}} \in \mathrm{H}^{\mathfrak{B}}$ *and* $(\mathfrak{A}, \overline{\mathrm{a}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{b}})$ *hold. Then there are HAH models* $\mathfrak{T}_{\mathfrak{A}} \models \varphi(\overline{\mathrm{c}}), \mathfrak{T}_{\mathfrak{B}} \models \psi(\overline{\mathrm{d}})$ *satisfying* $\overline{\mathrm{c}} \in \mathrm{H}^{\mathfrak{T}_{\mathfrak{A}}}, \overline{\mathrm{d}} \in \mathrm{H}^{\mathfrak{T}_{\mathfrak{B}}}$ *and* $(\mathfrak{T}_{\mathfrak{A}}, \overline{\mathrm{c}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{T}_{\mathfrak{B}}, \overline{\mathrm{d}})$*.*

Take $\varphi(\overline{x})$, $\psi(\overline{x})$ in form (NForm-$\mathsf{G}_{\mathsf{affix}}$) with the same head, satisfying the premise of Lemma 8. We have structures $\mathfrak{A}$ and $\mathfrak{B}$ and tuples $\overline{\mathrm{a}} \in A^k$ and $\overline{\mathrm{b}} \in B^k$ so that $\mathfrak{A} \models \varphi(\overline{\mathrm{a}}), \mathfrak{B} \models \psi(\overline{\mathrm{b}})$ and $(\mathfrak{A}, \overline{\mathrm{a}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{b}})$, where $\sigma := \mathsf{sig}(\varphi) \cap \mathsf{sig}(\psi)$ and $\tau := \mathsf{sig}(\varphi) \cup \mathsf{sig}(\psi)$. Using Lemma 12 we can assume that $\mathfrak{A}$ and $\mathfrak{B}$ are τ-HAHs. As done before, we aim at constructing a τ-structure being a model of both φ and ψ. To do this, we will construct a growing sequence of τ-HAHs $\mathfrak{U}_0 := \mathfrak{A} \leq \mathfrak{U}_1 \leq \ldots \leq \mathfrak{U}_n \leq \ldots$, whose limit $\mathfrak{U}$ will be a model of $\varphi \wedge \psi$. For simplicity, let us employ the following naming scheme. A tuple $\overline{\mathrm{d}}$ from $\mathfrak{U}_n$ ($\overline{\mathrm{d}} \sqsubseteq U_n$) is called (a) *n-fresh* if $\overline{\mathrm{d}} \sqsubseteq U_{n-1}$ and *n-aged* otherwise, (b) *maximal* if its not an affix of any different σ-live tuple.

A high-level idea of the construction of the sequence $\mathfrak{U}_i$, obfuscated by many challenging technical details, is as follows. Starting from $\mathfrak{A}$ we inductively "complete" types of all σ-live tuples to become proper τ-live tuples. This will help, if done carefully and in a bisimilarity-preserving way, the structure $\mathfrak{U}_i$ to fulfil the universal constraints of φ and ψ, but may introduce tuples without witnesses for the existential constraints. Hence, after each "completion" phase, we will "repair" the obtained structure by "copying" some substructures of $\mathfrak{A}$ and $\mathfrak{B}$ and "gluing" them on existing witness-lacking tuples (providing the required witnesses).

During the construction we will make sure that for every n-aged $\mathsf{sig}(\varphi)$-live (resp. $\mathsf{sig}(\psi)$-live) k-tuple in $\mathfrak{U}$, there exists a k-tuple in $\mathfrak{A}$ (resp. in $\mathfrak{B}$) having equal $(\mathsf{sig}(\varphi), k)$-affix-type (resp. $(\mathsf{sig}(\psi), k)$-affix-type). This will be controlled by means of partial *witness functions* $\mathsf{wit}_{\mathfrak{A}} : U_n \to A, \mathsf{wit}_{\mathfrak{B}} : U_n \to B$, intuitively pinpointing from where a tuple in $\mathfrak{U}$ originated from. To make the construction work, the witness function will fulfil several technical criteria, that are listed below. Conditions (a) and (b) speak about the compatibility of types between a tuple and its witness tuple; this guarantees that no tuple from $\mathfrak{U}_n$ violate the universal requirements of φ and ψ. Conditions (c)–(d) guarantees the satisfaction of the existential requirements of φ and ψ (condition (c) takes care of "local" requirements while (d) handles the "global" ones). Formally, for every n-aged $\overline{\mathrm{c}}$ from $\mathfrak{U}_n$ we have that:

(a) If $\overline{\mathrm{c}}$ is σ-live then both $\overline{\mathrm{d}} := \mathsf{wit}_{\mathfrak{A}}(\overline{\mathrm{c}})$ and $\overline{\mathrm{e}} := \mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{c}})$ are defined, $(\mathfrak{A}, \overline{\mathrm{d}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{e}})$ holds and $\mathsf{tp}_{\mathfrak{U}_n}^{\mathsf{G}_{\mathsf{affix}}[\tau]}(\overline{\mathrm{c}})$ is equal to $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{d}}) \cup \mathsf{tp}_{\mathfrak{B}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\psi)]}(\overline{\mathrm{e}})$.

(b) If $\overline{\mathrm{c}}$ is not σ-live but is $\mathsf{sig}(\varphi)$-live (resp. $\mathsf{sig}(\psi)$-live), then $\overline{\mathrm{d}} := \mathsf{wit}_{\mathfrak{A}}(\overline{\mathrm{c}})$ (resp. $\overline{\mathrm{d}} := \mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{c}})$) is defined, and $\mathsf{tp}_{\mathfrak{U}_n}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{c}})$ is equal to $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{d}})$ (resp. $\mathsf{tp}_{\mathfrak{B}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\psi)]}(\overline{\mathrm{d}})$).

(c) if $\overline{\mathrm{c}}$ is $\mathsf{sig}(\varphi)$-live (resp. $\mathsf{sig}(\psi)$-live), then for every existential requirement $\lambda := \mathrm{R}_i(\overline{x}_{1\ldots\ell_i}) \to \exists \overline{x}_{\ell_i\ldots\ell_i+k_i}(\mathrm{S}_i(\overline{x}_{1\ldots\ell_i+k_i}) \wedge \theta_i(\overline{x}_{1\ldots\ell_i+k_i}))$ from φ (resp. from ψ) with $\overline{\mathrm{c}}$ satisfying the premise of λ, there is a tuple $\overline{\mathrm{d}}$ in $\mathfrak{U}_n$ so that $\overline{\mathrm{c}}\overline{\mathrm{d}}$ satisfies the conclusion of λ.

(d) For every $\mathsf{sig}(\varphi)$-live (resp. $\mathsf{sig}(\psi)$-live) tuple $\overline{\mathrm{d}}$ from $\mathfrak{A}$ (resp. from $\mathfrak{B}$) there is a tuple $\overline{\mathrm{e}}$ in $\mathfrak{U}_1$ such that $\mathsf{tp}_{\mathfrak{U}_n}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{e}}) = \mathsf{tp}_{\mathfrak{A}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{d}})$ (resp. $\mathsf{tp}_{\mathfrak{U}_n}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\overline{\mathrm{e}}) = \mathsf{tp}_{\mathfrak{B}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\psi)]}(\overline{\mathrm{d}})$).

While the following property is not necessary to guarantee that the limit $\mathfrak{U}$ is a model of $\varphi \wedge \psi$, it plays an important technical role in the construction:

(e) If $\overline{\mathrm{d}}$ is an n-fresh σ-live tuple such that either $\mathsf{wit}_{\mathfrak{A}}(\overline{\mathrm{d}})$ or $\mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{d}})$ is undefined, then for every prefix $\overline{\mathrm{d}}_{1\ldots k}$ of $\overline{\mathrm{d}}$ that is contained in U_{n-1}, meaning that $\overline{\mathrm{d}}_{1\ldots k} \sqsubseteq U_{n-1}$, there exists an n-aged σ-live tuple $\overline{\mathrm{c}}$ which contains $\overline{\mathrm{d}}_{1\ldots k}$ as its affix.

Using conditions (a)–(d) it follows that $\mathfrak{U} \models \varphi \wedge \psi$, allowing us to conclude (by Lemma 8):

▶ **Theorem 13.** $\mathsf{G}_{\mathsf{inf}}, \mathsf{G}_{\mathsf{suf}}$ *and* $\mathsf{G}_{\mathsf{pre}}$ *enjoy the Craig Interpolation Property.*

We will now move on to the construction of $\mathfrak{U}$, described below. We start from the crucial, aforementioned notions of *completions* and *repairs*. Intuitively the *completion* just "completes a type of a tuple" in a bisimulation-preserving way, taking all symbols of τ into account. *Repair* simply "plugs in" certain subtrees from $\mathfrak{A}$ or $\mathfrak{B}$ into $\mathfrak{U}$, providing missing witnesses.

▶ **Definition 14** (completion). *Let $(\mathfrak{T}, \bar{\mathrm{d}})$ be a pointed τ-HAH, where $\bar{\mathrm{d}}$ is σ-live with $\mathsf{wit}_{\mathfrak{A}}, \mathsf{wit}_{\mathfrak{B}}$ defined. The* $\bar{\mathrm{d}}$-completion *of $\mathfrak{T}$ is obtained from $\mathfrak{T}$ by redefining interpretation of symbols from τ in a min. way so that $\mathsf{tp}_{\mathfrak{T}}^{\mathsf{G}_{\mathsf{affix}}[\tau]}(\bar{\mathrm{d}})$ equals $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\varphi)]}(\mathsf{wit}_{\mathfrak{A}}(\bar{\mathrm{d}})) \cup \mathsf{tp}_{\mathfrak{B}}^{\mathsf{G}_{\mathsf{affix}}[\mathsf{sig}(\psi)]}(\mathsf{wit}_{\mathfrak{B}}(\bar{\mathrm{d}}))$.*

▶ **Definition 15** (repair). *Let $(\mathfrak{T}, \bar{\mathrm{c}})$ be a pointed τ-HAH with only $\bar{\mathrm{d}} := \mathsf{wit}_{\mathfrak{A}}(\bar{\mathrm{c}})$ defined, where $\bar{\mathrm{c}}$ is σ-live in $\mathfrak{T}$. Suppose also that there is a tuple $\bar{\mathrm{e}}$ in $\mathfrak{B}$ such that $(\mathfrak{A}, \bar{\mathrm{d}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \bar{\mathrm{e}})$ holds. The* $(\mathfrak{B}, \bar{\mathrm{e}})$-repair *of $\bar{\mathrm{c}}$ is a τ-HAH $\mathfrak{T}'$ obtained from $\mathfrak{T}$ in the following five steps:*

1. *Let $\mathfrak{B}_0$ be the subtree of $\mathfrak{B}$ rooted at the first element of $\bar{\mathrm{e}}$.*
2. *Take $\mathfrak{T}'$ to be the union of $\mathfrak{T}$ and $\mathfrak{B}_0$ without $\bar{\mathrm{e}}$.*
3. *$\mathfrak{T}'$ will contain $\mathfrak{T}$ as a substructure.*
4. *By identifying $\bar{\mathrm{c}}$ with $\bar{\mathrm{e}}$, we interpret the relation symbols for tuples of elements of $\mathfrak{T}' \restriction B_0$ in such a way that the resulting substructure of $\mathfrak{T}'$ is isomorphic with $\mathfrak{B}_0$.*
5. *We set $\mathsf{wit}_{\mathfrak{B}}$ on freshly added elements to be the identity on $\mathfrak{B}_0$.*

The substructure $\mathfrak{T}' \restriction (B_0 \cup \bar{\mathrm{c}})$ is called a $\bar{\mathrm{c}}$-component *of $\mathfrak{T}'$. $\mathfrak{T}'$ becomes a HAH after a routine renaming. We define* $(\mathfrak{A}, \bar{\mathrm{d}})$-repair *of $\bar{\mathrm{c}}$ analogously.*

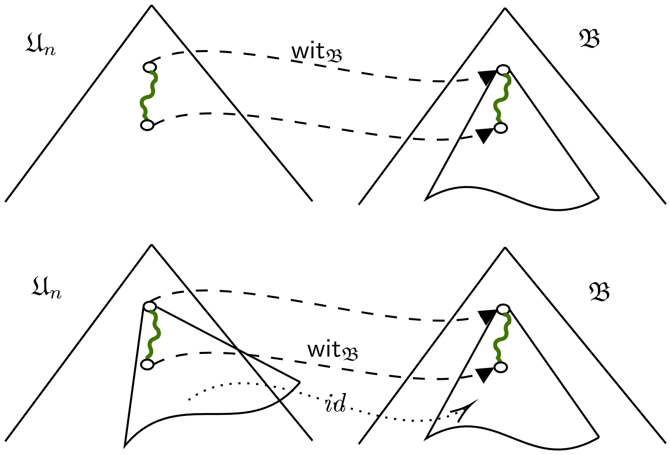


Figure 2 An example structure $\mathfrak{U}_n$ before and after we performed a "$\mathfrak{B}$"-repair.

We proceed with the base of induction, setting first $\mathfrak{U}_0$ to be $\mathfrak{A}$. It will be four-fold.

Base case: Step I. We set up $\mathsf{wit}_{\mathfrak{A}}$ and $\mathsf{wit}_{\mathfrak{B}}$ functions. For $\mathsf{wit}_{\mathfrak{A}}$ we will simply take the identity function. To define $\mathsf{wit}_{\mathfrak{B}}$, we intuitively proceed by traversing $\mathfrak{U}_0$ from top to bottom. More precisely, let $L_0^{\mathfrak{A}}$ denote the set of all maximal σ-live tuples in $\mathfrak{U}_0$. Letting $<_{lex}$ denote the lexicographic ordering of $\mathbb{N}^*$, we construct a well-founded linear ordering $\prec$ on $L_0^{\mathfrak{A}}$ as follows: $\bar{\mathrm{c}} \prec \bar{\mathrm{d}}$ iff there is an $i \leq \min\{|\bar{\mathrm{c}}|, |\bar{\mathrm{d}}|\}$ such that $\mathrm{c}_i <_{lex} \mathrm{d}_i$ and $\mathrm{c}_j = \mathrm{d}_j$ for every $j <_{lex} i$ (note that if there is no such i, then the tuples are equal due to maximality). One can show that $\bar{\mathrm{c}} \prec \bar{\mathrm{d}}$ implies that (♡): if $\bar{\mathrm{c}}$ and $\bar{\mathrm{d}}$ share some elements, then there exists i, j and k such that $\bar{\mathrm{c}}_{i \ldots j} = \bar{\mathrm{d}}_{1 \ldots k}$ and none of the elements d_ℓ, for $\ell > k$, occur in $\bar{\mathrm{c}}$. To prove this, one needs to simply show that if d_k occurs in $\bar{\mathrm{c}}$, then $(\mathrm{d}_1, \ldots, \mathrm{d}_k)$ is an affix of $\bar{\mathrm{c}}$ (the proof goes via careful inspection of the definition of HAHs, *cf.* [5, App. C.7]).

We define $\mathsf{wit}_{\mathfrak{B}}$ inductively w.r.t $\prec$. Consider a maximal σ-live tuple $\overline{\mathrm{d}}$ and suppose that we have defined $\mathsf{wit}_{\mathfrak{B}}$ for all the σ-live tuples $\overline{\mathrm{c}} \prec \overline{\mathrm{d}}$. There are two cases to consider.

- There exists a tuple $\overline{\mathrm{c}} \prec \overline{\mathrm{d}}$ sharing at least one element with $\overline{\mathrm{d}}$. By ($\heartsuit$), for every such tuple $\overline{\mathrm{c}}$ there are i, j and k so that $\overline{\mathrm{c}}_{i \ldots j} = \overline{\mathrm{d}}_{1 \ldots k}$ and none of the elements d_ℓ, for $\ell > k$, occur in $\overline{\mathrm{c}}$. Let $\overline{\mathrm{c}}$ be the tuple for which the corresponding value k is the largest. Since $\overline{\mathrm{c}}$ is σ-live, by induction hypothesis there exists some $\overline{\mathrm{e}} \sqsubseteq B$ such that $(\mathfrak{A}, \overline{\mathrm{c}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{e}})$ holds. Thus there exists some $\overline{\mathrm{f}} \sqsubseteq B$ such that $\overline{\mathrm{f}}_{i \ldots j} = \overline{\mathrm{e}}_{i \ldots j}$ and $(\mathfrak{A}, \overline{\mathrm{d}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{f}})$. We now extend $\mathsf{wit}_{\mathfrak{B}}$ in such a way that $\mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{d}}) = \overline{\mathrm{f}}$.
- Otherwise $\overline{\mathrm{c}}$ and $\overline{\mathrm{d}}$ do not share any elements. Since $\overline{\mathrm{d}}$ is σ-live and $(\mathfrak{A}, \overline{\mathrm{a}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{b}})$, there exists some $\overline{\mathrm{e}} \sqsubseteq B$ such that $(\mathfrak{A}, \overline{\mathrm{d}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \overline{\mathrm{e}})$. We then simply extend $\mathsf{wit}_{\mathfrak{B}}$ in such a way that $\mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{d}}) = \overline{\mathrm{e}}$.

The resulting mapping $\mathsf{wit}_{\mathfrak{B}}$. This finishes Step I.

Base case: Step II. We next complete types of all fresh (= all in this case) σ-live tuples of $\mathfrak{U}_0$. Take any maximal σ-live tuple $\overline{\mathrm{d}}$ from $\mathfrak{U}_0$ and perform the $\overline{\mathrm{d}}$-completion of $\mathfrak{U}_0$. It is easy to see that this process is conflict-free in the following sense: there is no tuple $\overline{\mathrm{c}}$ and $R \in \mathsf{sig}(\psi)$ so that we end up specifying both $\overline{\mathrm{c}} \in \mathrm{R}^{\mathfrak{U}_0}$ and $\overline{\mathrm{c}} \notin \mathrm{R}^{\mathfrak{U}_0}$. First, if two maximal σ-live tuples $\overline{\mathrm{d}}$ and $\overline{\mathrm{e}}$ have affixes $\overline{\mathrm{d}}_{i \ldots j}$ and $\overline{\mathrm{e}}_{k \ldots \ell}$ such that $\overline{\mathrm{d}}_{i \ldots j} = \overline{\mathrm{e}}_{k \ldots \ell}$, then we know that $\mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{d}}_{i \ldots j}) = \mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{e}}_{k \ldots \ell})$, and thus there are no conflicts in the "intersections" of σ-live tuples. Second, by construction $\mathsf{tp}_{\mathfrak{A}}^{\mathsf{G}_{\mathsf{affix}}[\sigma]}(\overline{\mathrm{d}}) = \mathsf{tp}_{\mathfrak{B}}^{\mathsf{G}_{\mathsf{affix}}[\sigma]}(\mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{d}}))$ holds for all maximal σ-live tuple $\overline{\mathrm{d}}$, and hence the σ-infix-types that we assigned to σ-live tuples are indeed types, *i.e.* they are consistent. Thus our process is conflict-free.

Note that our structure satisfies now conditions (a) and (b).

Base case: Step III. We finish the base case by providing witnesses for fresh $\mathsf{sig}(\psi)$-tuples via repairs. Recall that $L_0^{\mathfrak{A}}$ denotes the set of all maximal σ-live tuples in $\mathfrak{U}_0^{\mathfrak{A}}$. For each $\overline{\mathrm{d}} \in L_0^{\mathfrak{A}}$ we perform the $(\mathfrak{B}, \mathsf{wit}_{\mathfrak{B}}(\overline{\mathrm{d}}))$-repair of $\overline{\mathrm{d}}$; the resulting structure will be taken to be $\mathfrak{U}_1$. Note that now every $\mathsf{sig}(\psi)$-live tuple in $\mathfrak{U}_1$ has its witnesses for the existential requirements, but there may be new $\mathsf{sig}(\varphi)$-live tuples without them. Moreover, $\mathsf{wit}_{\mathfrak{B}}$ is defined for all freshly added elements, but $\mathsf{wit}_{\mathfrak{A}}$ is not. Furthermore, we note that $\mathfrak{U}_1$ now satisfies condition (e), since all the 1-fresh live tuples for which $\mathsf{wit}_{\mathfrak{A}}$ is not defined are present in the subtrees that we attached to $\mathfrak{U}_0$ during the repair, which is done only at (maximal) σ-live tuples.

Base case: Step IV. It could be the case that the structure $\mathfrak{U}_1$ produced in the previous step violates (d), due to the lack of realisation of a certain type from $\mathfrak{B}$. Thus, as an extra precaution, unique to the base case, we add a disjoint copy of $\mathfrak{B}$ to $\mathfrak{U}_1$ and define $\mathsf{wit}_{\mathfrak{B}}$ for it to be the identity. Note that now (d) will be satisfied in any extension of $\mathfrak{U}_1$.

Inductive step. The inductive step is analogous to Steps I-III from the base case, hence we keep its description short. Assume that $\mathfrak{U}_n$ is defined and that in the previous step of the construction we employed $\mathfrak{B}$-repairs (the case of $\mathfrak{A}$-repairs is symmetric). Given a component $\mathfrak{C}$ that was created during such a repair, we let $L_n^{\mathfrak{C}}$ denote the set of all maximal n-fresh σ-live tuples in $\mathfrak{C}$. Since $\mathfrak{C}$ is essentially a HAT (up to renaming), we can again define a well-founded linear order $\prec$ on $L_n^{\mathfrak{C}}$ in the same way as we did in the base case for $L_0^{\mathfrak{A}}$. As in the base case, we then define missing values of $\mathsf{wit}_{\mathfrak{A}}$ for elements of $\mathfrak{C}$ inductively w.r.t $\prec$.

Observe that some of the tuples in $L_n^{\mathfrak{C}}$ might contain a proper prefix of elements of U_{n-1}. In the case of $\mathsf{G}_{\mathsf{suf}}$ these tuples do not cause any problems to us, because suffix-types do not impose any constraints on proper prefixes. In the cases of $\mathsf{G}_{\mathsf{pre}}$ and $\mathsf{G}_{\mathsf{inf}}$ we handle these tuples by using the fact that $\mathfrak{U}_n$ satisfies condition (e) as follows. Let $\bar{\mathrm{d}} \in L_n^{\mathfrak{C}}$ be such a tuple and let k be the largest index so that $\bar{\mathrm{d}}_{1\ldots k} \sqsubseteq U_{n-1}$. Using condition (e), we know that there exists a σ-live n-aged tuple $\bar{\mathrm{c}}$ so that $\bar{\mathrm{c}}_{i\ldots j} = \bar{\mathrm{d}}_{1\ldots k}$, for some i and j. Employing condition (a), we know that $\bar{\mathrm{e}} := \mathsf{wit}_{\mathfrak{A}}(\bar{\mathrm{c}})$, $\bar{\mathrm{f}} := \mathsf{wit}_{\mathfrak{B}}(\bar{\mathrm{c}})$ and $\bar{\mathrm{h}} := \mathsf{wit}_{\mathfrak{B}}(\bar{\mathrm{d}})$ are defined and that $(\mathfrak{A}, \bar{\mathrm{e}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \bar{\mathrm{f}})$. Thus there exists a σ-live tuple $\bar{\mathrm{g}} \sqsubseteq A$ such that $\bar{\mathrm{f}}_{i\ldots j} = \bar{\mathrm{g}}_{i\ldots j}$ and $(\mathfrak{A}, \bar{\mathrm{g}}) \sim_{\mathsf{G}_{\mathsf{affix}}[\sigma]} (\mathfrak{B}, \bar{\mathrm{h}})$. We now extend $\mathsf{wit}_{\mathfrak{A}}$ in such a way that $\mathsf{wit}_{\mathfrak{A}}(\bar{\mathrm{d}}) = \bar{\mathrm{g}}$.

The above procedure is repeated for all components $\mathfrak{C}$ that were introduced during the previous repair. Having defined $\mathsf{wit}_{\mathfrak{A}}$ for all the elements, we perform a completion that works for exactly the same reasons as described before. Finally, letting L_n denote the set of all maximal n-fresh σ-live tuples, we perform repair of every tuple in L_n, which results in a model that we select as $\mathfrak{U}_{n+1}$. We stress that every $\mathsf{sig}(\varphi)$-live tuple in $\mathfrak{U}_{n+1}$ has its witnesses for the existential requirements, but there can now be new $\mathsf{sig}(\psi)$-live tuples without them. We also emphasise that $\mathsf{wit}_{\mathfrak{A}}$ is defined for all the new elements but $\mathsf{wit}_{\mathfrak{B}}$ might not be. This concludes the inductive step and hence, also the construction of $\mathfrak{U}$ and the proof of Theorem 13.

▶ Remark 16. The presented model construction is quite generic. Indeed, the only part of the construction which is really specific to $\mathsf{G}_{\mathsf{affix}}$ is the first step of the construction, namely the part where we define inductively the values of witness functions. We expect that the presented technique can be easily adapted to other logics, especially to other fragments of the guarded fragments. For instance, we believe that our technique can be adjusted, *e.g.* to the case of the two-variable GF from [14] as well as to the uniform one-dimensional GF from [16].

5 Conclusions

In this paper kick-started a project of understanding the model theory of the family of guarded and unguarded ordered logics. We first investigated the relative expressive power of ordered logics by means of suitable bisimulations. Afterwards, we proceed with the Craig Interpolation Property (CIP) showing that (i) the fluted and the forward fragments do not enjoy CIP, (ii) while the other logics that we consider enjoy it. The fact that the fluted fragment does not posses CIP was quite unexpected in the light of already existing claims for the contrary [21, Thm. 14]. For the other logics we proposed a novel model-theoretic "complete-and-repair" method of creating a model out of two bisimilar forest-like structures.

There are several interesting future work directions.

1. One example is to investigate the Łoś-Tarski Preservation Theorem as well as other preservation theorems. While we think that we already have a working construction for guarded ordered logics, the status of ŁTPT holding for $\mathsf{L}_{\mathsf{pre}}$, $\mathsf{L}_{\mathsf{suf}}$, and $\mathsf{L}_{\mathsf{inf}}$ is not clear.[6]
2. Another work direction is to take a look at on effective interpolation, similarly to what has been proposed in [6] as well as on the interpolant existence problem for $\mathsf{L}_{\mathsf{inf}}$ and $\mathsf{L}_{\mathsf{suf}}$, as done in [17]. Preliminary results were obtained. It is also interesting whether the guarded ordered logics enjoy stronger versions of interpolations, *e.g.* Lyndon's interpolation or Otto's interpolation. We are quite optimistic about it.
3. What is the complexity of the model checking problem for ordered logics, if we use list encoding to encode our structures?

[6] Purdy provides a "proof" in [21] that $\mathsf{L}_{\mathsf{suf}}$ has ŁTPT. However, his "proof" is sketchy and lacks sufficient mathematical arguments required to verify its correctness. In the light of our discovery of yet another false claim from [21], we believe that it is safe to assume that ŁTPT for $\mathsf{L}_{\mathsf{suf}}$ *is open.*

We are also actively working on the finitary versions of van Benthem theorem for the forward guarded fragment as well as the Lindström-style characterisation theorems. This is an ongoing work of Benno Fünfstück, a master student at TU Dresden, under the supervision of B. Bednarczyk.

References

1 Hajnal Andréka, István Németi, and Johan van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philos. Log.*, 1998.

2 Vince Bárány, Michael Benedikt, and Balder ten Cate. Some Model Theory of Guarded Negation. *J. Symb. Log.*, 2018.

3 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded Negation. *J. ACM*, 2015.

4 Bartosz Bednarczyk. Exploiting Forwardness: Satisfiability and Query-Entailment in Forward Guarded Fragment. In *JELIA*, 2021.

5 Bartosz Bednarczyk and Reijo Jaakkola. Towards a model theory of ordered logics: Expressivity and interpolation (extended version), arXiV 2022. doi:10.48550/ARXIV.2206.11751.

6 Michael Benedikt, Balder ten Cate, and Michael Vanden Boom. Effective Interpolation and Preservation in Guarded Logics. *ACM Trans. Comput. Log.*, 2016.

7 Dietmar Berwanger and Erich Graedel. Games and Model Checking for Guarded Logics. In *LPAR 2001*, 2001.

8 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bull. Symb. Log.*, 1997.

9 Erich Grädel and Martin Otto. The Freedoms of (Guarded) Bisimulation. In Alexandru Baltag and Sonja Smets, editors, *Johan van Benthem on Logic and Information Dynamics*. Springer, 2014.

10 Erich Graedel, Phokion Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007.

11 Lauri Hella and Antti Kuusisto. One-dimensional Fragment of First-order Logic. In *AIML 2014*, 2014.

12 Andreas Herzig. A New Decidable Fragment of First Order Logic. In *Third Logical Biennial, Summer School and Conference in Honour of S. C. Kleene*, 1990.

13 Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

14 Eva Hoogland, Maarten Marx, and Martin Otto. Beth Definability for the Guarded Fragment. In *LPAR*, 1999.

15 Reijo Jaakkola. Ordered Fragments of First-Order Logic. In *MFCS*, 2021.

16 Reijo Jaakkola. Uniform Guarded Fragments. In *FOSSACS*, 2022.

17 Jean Christoph Jung and Frank Wolter. Living without Beth and Craig: Definitions and Interpolants in the Guarded and Two-Variable Fragments. In *LICS*, 2021.

18 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

19 Maarten Marx. *Algebraic relativization and arrow logic*. University of Amsterdam, 1995.

20 Ian Pratt-Hartmann, Wieslaw Szwast, and Lidia Tendera. The Fluted Fragment Revisited. *J. Symb. Log.*, 2019.

21 William C. Purdy. Complexity and Nicety of Fluted Logic. *Stud Logica*, 2002.

22 Willard Quine. *The Ways of Paradox and Other Essays, Revised Edition*. Harvard University Press, 1976.

23 Abraham Robinson. A Result on Consistency and Its Application to the Theory of Definition. *Journal of Symbolic Logic*, 1960.

24 Luc Segoufin and Balder ten Cate. Unary negation. *Log. Methods Comput. Sci.*, 9(3), 2013.

25 Thomas Sturm, Marco Voigt, and Christoph Weidenbach. Deciding First-Order Satisfiability when Universal and Existential Variables are Separated. In *LICS 2016*, 2016.

26 Johan van Benthem. Modal Foundations for Predicate Logic. *Log. J. IGPL*, 1997.

Algebraic Representations of Unique Bipartite Perfect Matching

Gal Beniamini ✉
The Hebrew University of Jerusalem, Israel

Abstract

We obtain complete characterizations of the Unique Bipartite Perfect Matching function, and of its Boolean dual, using multilinear polynomials over the reals. Building on previous results [2, 3], we show that, surprisingly, the dual description is *sparse* and has *low* ℓ_1*-norm* – only exponential in $\Theta(n \log n)$, and this result extends even to other families of matching-related functions. Our approach relies on the Möbius numbers in the matching-covered lattice, and a key ingredient in our proof is Möbius' inversion formula.

These polynomial representations yield complexity-theoretic results. For instance, we show that unique bipartite matching is *evasive* for classical decision trees, and *nearly evasive* even for generalized query models. We also obtain a tight $\Theta(n \log n)$ bound on the log-rank of the associated two-party communication task.

2012 ACM Subject Classification Mathematics of computing → Matchings and factors; Theory of computation → Communication complexity; Theory of computation → Oracles and decision trees

Keywords and phrases Bipartite Perfect Matching, Boolean Functions, Partially Ordered Sets

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.16

1 Introduction

A perfect matching in a graph is a subset of edges spanning the graph, no two of which are incident to the same vertex. In this paper we consider the *decision problem* of unique bipartite matching: the input is a balanced bipartite graph over $2n$ vertices, and the goal is to determine whether the graph contains a *unique* perfect matching. This problem can be naturally cast as a Boolean function.

▶ **Definition.** *The unique bipartite perfect matching function* $\mathrm{UBPM_n} : \{0,1\}^{n^2} \to \{0,1\}$ *is*

$$\mathrm{UBPM_n}(x_{1,1}, \dots, x_{n,n}) = \begin{cases} 1 & \{(i,j) : x_{i,j} = 1\} \text{ has a unique perfect matching} \\ 0 & \text{otherwise.} \end{cases}$$

The complexity of $\mathrm{UBPM_n}$ is closely related to that of $\mathrm{BPM_n}$ – the problem in which we drop the uniqueness condition and simply ask whether a bipartite graph *contains* a perfect matching. Both $\mathrm{BPM_n}$ and $\mathrm{UBPM_n}$ are known to lie in **P**, due to a classical result by Edmonds [9]. However, despite their close connection, not all known algorithmic results extend from one problem to another. For instance, $\mathrm{UBPM_n}$ was shown by Kozen, Vazirani and Vazirani to be in **NC** [19] (see also [14]), and no such result is known for $\mathrm{BPM_n}$. Lovász showed that $\mathrm{BPM_n}$ is in **RNC** [21], and the current best-known *deterministic* parallel algorithm is due to Fenner, Gurjar and Thierauf [10], placing the problem in **Quasi-NC**. Determining the membership of bipartite perfect matching in **NC** remains one of the main open problems in parallelizability.

Our main results in this paper are the complete characterizations of both $\mathrm{UBPM_n}$ and its dual function, by means of polynomials. These characterizations leverage a deep connection to the polynomial representations of $\mathrm{BPM_n}$, obtained in [3, 2], and it is our hope that they can be used to further our understanding of the connection between the two. To present our results we require some notation. We say that a bipartite graph is *matching-covered* if every edge of the graph participates in some perfect matching. For a graph G we denote its *cyclomatic number*, a topological quantity, by $\chi(G) = e(G) - v(G) + c(G)$. The set of all perfect matchings of G is denoted $\mathrm{PM}(G)$, and the cardinality of this set is denoted $\mathrm{per}(G)$ (the permanent of G). Under these notations, our first theorem is the following closed-form description of the *unique* real multilinear polynomial representing $\mathrm{UBPM_n}$.

▶ **Theorem 1** (The Unique Bipartite Perfect Matching Polynomial).

$$\mathrm{UBPM_n}(x_{1,1}, \dots, x_{n,n}) = \sum_{G \subseteq K_{n,n}} c_G \prod_{(i,j) \in E(G)} x_{i,j}$$

where

$$c_G = \begin{cases} (-1)^{\chi(G)} \,\mathrm{per}(G) & G \textit{ is matching-covered} \\ 0 & \textit{otherwise.} \end{cases}$$

The polynomial appearing in Theorem 1 bears a striking resemblance to the representation of $\mathrm{BPM_n}$, the only difference being the multiplicative $\mathrm{per}(G)$ appearing in each term of $\mathrm{UBPM_n}$. This is a direct result of the connection between the two functions and the *matching-covered lattice*, hereafter $\mathcal{L}_n$, which is formed by all matching-covered graphs of order $2n$, ordered with respect to the subgraph relation. Billera and Sarangarajan [5] proved that $\mathcal{L}_n$ is isomorphic to the face lattice of the Birkhoff Polytope $\mathbf{B}_n$. Consequently, this combinatorial lattice is Eulerian, and its Möbius function is particularly well-behaved – a fact which we rely on indirectly throughout this paper. In [3], it was shown that $\mathrm{BPM_n}$ is intimately related to the matching-covered lattice: every such graph corresponds to a monomial, and their coefficients are given by Möbius numbers. Our proof of Theorem 1 extends this connection by leveraging Möbius Inversion Formula, and in fact allows us to derive the polynomial representation for *any* indicator function over $\mathcal{L}_n$ (including, for instance, $\mathrm{BPM_n}$), while also simplifying somewhat parts of the original proof.

Theorem 1 yields information-theoretic lower bounds. For example, $\mathrm{UBPM_n}$ has full total degree and is thus *evasive*, i.e., any decision tree computing it must have full depth, n^2. Unlike its analogue $\mathrm{BPM_n}$, which is a *monotone* bipartite graph property and thus known to be evasive [17], the *unique* perfect matching function is *not monotone*, and for such functions evasiveness is not guaranteed (see e.g. [23]). We also obtain lower bounds against generalized families of decision trees, whose internal nodes are labeled by arbitrary parity functions (XOR-DT), or conjunctions (AND-DT), over subsets of the inputs bits.

▶ **Corollary.** *For classical, parity, and conjunction trees, the following lower bounds hold:*

$$\mathrm{D}(\mathrm{UBPM_n}) = n^2, \quad \mathrm{D}^{\mathrm{XOR}}(\mathrm{UBPM_n}) \geq \left(\tfrac{1}{2} - o(1)\right) n^2 \quad \textit{and} \quad \mathrm{D}^{\mathrm{AND}}(\mathrm{UBPM_n}) \geq (\log_3 2) n^2 - o(1).$$

In the second part of this paper we consider the Boolean dual function $\mathrm{UBPM_n^\star}$, which is obtained by flipping all the input and output bits (or formally, $\mathrm{UBPM_n^\star}(x_{1,1}, \dots, x_{n,n}) = 1 - \mathrm{UBPM_n}(1 - x_{1,1}, \dots, 1 - x_{n,n})$). By construction, this is the indicator over all bipartite graphs whose *complement* does *not* contain a unique perfect matching. Our second result is a complete characterization of $\mathrm{UBPM_n^\star}$ as a real multilinear polynomial. This description relies *heavily* on the that of $\mathrm{BPM_n^\star}$ – which is the dual of the bipartite perfect matching function $\mathrm{BPM_n}$. The polynomial representation of the latter dual was obtained in a series of papers [3, 2], and is omitted here for brevity.

▶ **Theorem 2** (The Dual Polynomial of Unique Bipartite Perfect Matching)**.**

$$\mathrm{UBPM}_{\mathrm{n}}^{\star}(x_{1,1},\dots,x_{n,n}) = \sum_{G\subseteq K_{n,n}} c_G^{\star} \prod_{(i,j)\in E(G)} x_{i,j}$$

where

$$c_G^{\star} = \mathrm{per}(G)\cdot a_G^{\star} + \sum_{M\notin \mathrm{PM}(G)} (-1)^{|E(M)\setminus E(G)|}\cdot a_{G\cup M}^{\star}$$

and $a_G^{\star}$ *denotes the coefficient of* G *in* $\mathrm{BPM}_{\mathrm{n}}^{\star}$.

Theorem 2 expresses the coefficient of every graph G as an alternating sum over coefficients of $\mathrm{BPM}_{\mathrm{n}}^{\star}$, corresponding exactly to those graphs formed by adjoining a single perfect matching to G. This suffices in order to *inherit* the main structural result of [2] regarding $\mathrm{BPM}_{\mathrm{n}}^{\star}$: the ℓ_1-norm of $\mathrm{UBPM}_{\mathrm{n}}^{\star}$, i.e., the norm of the coefficient vector of the representing polynomial, is *very small* – only exponential in $\Theta(n\log n)$, and this is tight.

▶ **Corollary.** *The dual polynomial is sparse and its coefficients are small. Explicitly,*

$$\log\|\mathrm{UBPM}_{\mathrm{n}}^{\star}\|_1 = \Theta(n\log n).$$

The low norm of the dual yields algorithmic results for the unique-bipartite-matching problem, and for related matching problems. For instance, through the approximation scheme of [2, 29], it allows one to obtain a low-degree polynomial *approximation* of the unique bipartite matching function over the hypercube (i.e., "approximate degree"), which holds even for *exponentially small* error. The same ℓ_1-norm bound also directly extends to the spectral norm of $\mathrm{UBPM}_{\mathrm{n}}$,[1] which is a well-studied quantity in analysis of Boolean functions.

Finally, we consider the two-party deterministic communication complexity of unique bipartite matching. The input is a graph $G \subseteq K_{n,n}$, whose edges are distributed among two parties according to *any arbitrary* and *fixed* partition. The sparse polynomial representation of $\mathrm{UBPM}_{\mathrm{n}}^{\star}$ allows us to deduce that the log-rank of the communication matrix, for *any* of the above communication tasks, is bounded by only $\mathcal{O}(n\log n)$, and we prove that this is tight.[2] We remark that, while we show that unique matching has low log-rank, not much is known regarding its *deterministic communication complexity*. For the monotone variant $\mathrm{BPM}_{\mathrm{n}}$, known algorithms (e.g. [15]) can be translated into protocols using only $\widetilde{\mathcal{O}}(n^{3/2})$ bits [25]. However, it is currently not known how to convert algorithms for $\mathrm{UBPM}_{\mathrm{n}}$ (such as [11, 12]), into protocols using even $\mathcal{O}(n^{2-\varepsilon})$ bits, for any $\varepsilon > 0$. Determining the deterministic communication complexity of $\mathrm{UBPM}_{\mathrm{n}}$ is thus left as an open problem.

2 Preliminaries and Notation

2.1 Boolean Functions and Polynomials

Every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be *uniquely* represented by a multilinear polynomial $p \in \mathbb{R}[x_1,\dots,x_n]$ (see e.g. [27]), where f and p agree on all Boolean inputs $\{0,1\}^n$. The family of subsets corresponding to monomials in this polynomial representation

[1] The spectra of any function and its dual are identical up to sign, and the $\{0,1\}$-polynomial ℓ_1-norm is always trivially at least as large as the $\{\pm 1\}$-representation ("Fourier") ℓ_1-norm.

[2] In fact, our results hold even for a certain $\wedge$-*lifted* and dualised version of this problem.

(i.e., whose coefficient does not vanish) is denoted by $\mathrm{mon}(f)$. The cardinality of $\mathrm{mon}(f)$ is known as the *sparsity* of f, and the maximal cardinality of any $S \in \mathrm{mon}(f)$ is known as the *total degree* of f, hereafter $\deg(f)$. The ℓ_1-norm of f is the norm of its representing polynomial's coefficient vector, namely:

$$\|f\|_1 \overset{\mathrm{def}}{=} \left\|(a_S)_{S\subseteq[n]}\right\|_1, \text{ where } f \text{ is } \{0,1\}\text{-represented by } p(x_1,\ldots,x_n) = \sum_{S\subseteq[n]} a_S \prod_{i\in S} x_i.$$

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, it is often useful to consider the transformation in which we *invert* all the input and output bits. This process produces a new Boolean function $f^\star$, known as the Boolean dual.

▶ **Definition 3.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. The* ***Boolean dual*** *of f is the function $f^\star : \{0,1\}^n \to \{0,1\}$ where the symbols 0 are 1 are interchanged. Formally,*

$$\forall x \in \{0,1\}^n : \; f^\star(x_1,\ldots,x_n) = 1 - f(1-x_1,\ldots,1-x_n).$$

The polynomial representations of a Boolean function f and its dual $f^\star$ can differ substantially (for example $\mathrm{AND_n}^\star = \mathrm{OR_n}$, and while the former is represented by a single monomial, the latter consists of $2^n - 1$ monomials). However, since f and $f^\star$ are obtained by affine transformations of one another, they share many properties. For example, their Fourier spectra are identical [27], up to sign. Moreover, they have the same approximate degree [2] for any error ε, and the ranks of their associated communication matrices (see proceeding subsections) are identical up to an additive constant (of 1).

2.2 Graphs

We use the standard notation for quantities relating to graphs. In particular, the sets of vertices, edges and connected components of a graph are denoted by $V(G)$, $E(G)$ and $C(G)$, and their cardinalities are denoted $v(G)$, $e(G)$ and $c(G)$, respectively. A less common measure appearing in this paper is the cyclomatic number $\chi(G)$, a topological quantity.

▶ **Definition 4.** *Let G be a graph. The* ***cyclomatic number*** *of G is defined by:*

$$\chi(G) = e(G) - v(G) + c(G).$$

A *matching* in a graph $G \subseteq K_{n,n}$ is a collection of edges sharing no vertices, and said matchings are called *perfect* if they contain exactly n edges (i.e., every vertex in the graph is incident to precisely one edge in the matching). The set of *all* perfect matchings denoted by $\mathrm{PM}(G)$. For any graph $G \subseteq K_{n,n}$, we define the *permanent* $\mathrm{per}(G)$ and the *determinant* $\det(G)$ as the application of these two functions to the biadjacency matrix of G, noting that $\mathrm{per}(G)$ counts the number of perfect matchings in G.

Perfect matchings and the graphs formed by unions thereof play a central role in this paper. A graph $G \subseteq K_{n,n}$ is called **matching-covered** if and only if every edge of G participates in some perfect matching. Matching-covered graphs have interesting combinatorial properties. For example, this is precisely the family of all graphs admitting a bipartite ear decomposition (similar to the ear decomposition of 2-edge-connected graphs). This family had previously appeared extensively in the literature, and in particular had been studied at length by Lovász and Plummer [28], and by Hetyei [13]. Hereafter, we denote the set of all such graphs by

$$\mathrm{MC_n} = \Big\{G \subseteq K_{n,n} : G \text{ is matching-covered}\Big\}.$$

All graphs in this paper are *balanced bipartite graphs*, over the fixed vertex set of the complete bipartite graph $K_{n,n}$. Consequently, we use the notation $G \subseteq H$ to indicate inclusion over the edges, and similarly $G \cup H$ is the graph whose edges are $E(G) \cup E(H)$. Lastly, many of the Boolean functions appearing in this paper are defined over subgraphs of $K_{n,n}$, where every input bit is associated with a single edge. For such functions, the notation $f(G)$, where $G \subseteq K_{n,n}$, corresponds to this mapping.

2.3 Communication Complexity

In this paper we consider the *two-party deterministic communication model.* For a comprehensive textbook on the topic, we refer the reader to [20]. The **deterministic communication complexity** of f, hereafter $\mathrm{D}^{\mathrm{CC}}(f)$, is the *least number of bits communicated by a protocol computing f, on the worst-case input.* Any (*unpartitioned*) Boolean function $f : \{0,1\}^n \to \{0,1\}$ naturally gives rise to a *family* of associated two-party communication tasks: one corresponding to each possible partition of the input bits between the two parties. The **deterministic communication complexity of a Boolean function** is then defined as follows.

▶ **Definition 5.** *The deterministic communication complexity of $f : \{0,1\}^n \to \{0,1\}$ is defined*

$$\mathrm{D}^{\mathrm{CC}}(f) \overset{def}{=} \max_{S \sqcup \bar{S} = [n]} \mathrm{D}^{\mathrm{CC}}\left(f_S(x,y)\right)$$

where $\mathrm{D}^{\mathrm{CC}}\left(f_S(x,y)\right)$ is the deterministic communication complexity of the two-party Boolean function $f_S(x,y) : \{0,1\}^{|S|} \times \{0,1\}^{|\bar{S}|} \to \{0,1\}$, representing f under the partition $S \sqcup \bar{S}$.

For any two-party Boolean function, let us also define the following two useful objects.

▶ **Definition 6.** *Let $f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}$. The **communication matrix** of f is*

$$M_f \in \mathbb{R}^{\{0,1\}^m \times \{0,1\}^n}, \text{ where } \forall (x,y) \in \{0,1\}^m \times \{0,1\}^n : \ M_f(x,y) = f(x,y).$$

▶ **Definition 7.** *Let $f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}$ be a two-party function. We say that $S \subseteq \{0,1\}^m \times \{0,1\}^n$ is a **fooling set** for f if and only if:*

$$S \subseteq f^{-1}(1), \text{ and } \forall (x_1,y_1) \neq (x_2,y_2) \in S : \ \{(x_1,y_2),(x_2,y_1)\} \cap f^{-1}(0) \neq \emptyset.$$

The log of the rank of M_f *over the reals* (sometimes referred to as the "log-rank of f") is intimately related to the communication complexity of f. A classical theorem due to Mehlhorn and Schmidt [24] states that $\mathrm{D}^{\mathrm{CC}}(f) \geq \log_2 \operatorname{rank} M_f$, and these two quantities are famously conjectured to be polynomially related [22]. As for the fooling set, it is well known that $\mathrm{D}^{\mathrm{CC}}(f) \geq \log_2 \operatorname{fs}(f)$ for any two-party function f [20], where $\operatorname{fs}(f)$ is the maximum size of a fooling set. This bound was extended by Dietzfelbinger, Hromkovič and Schnitger [8], who showed that in fact $\log \operatorname{fs}(f) \leq 2 \log \operatorname{rank} f + 2$.

2.4 Posets, Lattices and Möbius Functions

Partially ordered sets (hereafter, **posets**) are defined by a tuple $\mathscr{P} = (P, \leq)$, where P is the element set, and $\leq$ is the order relation (which is reflexive, antisymmetric and transitive). For any two elements $x, y \in P$, the notation $[x,y] \overset{\text{def}}{=} \{z \in P : x \leq z \leq y\}$ denotes the *interval* from x to y. A *combinatorial lattice* is a poset satisfying two additional conditions: every two elements have a least upper bound (a “join”), and a greatest lower bound (a “meet”). The *face lattice of a polytope* is a combinatorial lattice whose elements correspond to the faces of a polytope, ordered by the subset relation. Such a lattice is *bounded* – it has a unique bottom element (the empty face $\hat{0}$), and a unique top element (the polytope itself), and it is also *graded*, meaning that the length of all maximal chains between any two elements x, y are identical (in other words, the elements can be *ranked*).

Partially ordered sets come equipped with an important function known as the **Möbius function**. The Möbius function of a poset is the inverse, with respect to convolution, of its zeta function $\zeta(x,y) = \mathbb{1}\{x < y\}$. For information on incidence algebra and the Möbius function, we refer the reader to [30].

▶ **Definition 8** (Möbius Function for Posets). *Let $\mathcal{P} = (P, \leq)$ be a finite poset. The Möbius function $\mu_{\mathcal{P}} : P \times P \to \mathbb{R}$ of $\mathcal{P}$ is defined*

$$\forall x \in P: \ \mu_{\mathcal{P}}(x,x) = 1, \quad \forall x, y \in P, \ y < x: \ \mu_{\mathcal{P}}(y,x) = -\sum_{y \leq z < x} \mu_{\mathcal{P}}(y,z).$$

The Möbius Inversion Formula allows one to relate two functions defined on a poset $\mathcal{P}$, where one function is a downwards closed sum of another, by means of the Möbius function. This can be seen as a generalization of its number-theoretic analogue (as indeed the Möbius function of number theory arises in this manner from the *divisibility poset*).

▶ **Theorem 9** (Möbius Inversion Formula, see [30]). *Let $\mathcal{P} = (P, \leq)$ be a finite poset and let $f, h : P \to \mathbb{F}$ be two functions, where $\mathbb{F}$ is a field. Then:*

$$\forall x \in P: \ h(x) = \sum_{y \leq x} f(y) \quad \iff \quad \forall x \in P: \ f(x) = \sum_{y \leq x} h(y)\mu_{\mathcal{P}}(y,x).$$

3 The Unique Perfect Matching Polynomial

Our main object of study is the unique bipartite matching function.

▶ **Definition 10.** *The Unique Bipartite Perfect Matching function is defined*

$$\mathrm{UBPM_n}(x_{1,1}, \ldots, x_{n,n}) = \begin{cases} 1 & \{(i,j) : x_{i,j} = 1\} \subseteq K_{n,n} \textit{ has a unique P.M.} \\ 0 & \textit{otherwise.} \end{cases}$$

The unique multilinear representation of $\mathrm{UBPM_n}$ is characterized in the following Theorem.

▶ **Theorem 1.** *The unique polynomial* $\mathrm{UBPM_n} : \{0,1\}^{n^2} \to \{0,1\}$ *is given by*

$$\mathrm{UBPM_n}\left(x_{1,1}, \ldots, x_{n,n}\right) = \sum_{G \in \mathrm{MC_n}} (-1)^{\chi(G)} \ \mathrm{per}(G) \prod_{(i,j) \in E(G)} x_{i,j}.$$

Proof. The proof is centered around the combinatorial *lattice of matching-covered graphs*,

$$\mathcal{L}_n = \left(\mathrm{MC_n} \cup \{\hat{0}\}, \subseteq\right), \text{ where } \hat{0} \text{ is the graph with } 2n \text{ isolated vertices}$$

where the order relation for this lattice is *containment over the edge set*, i.e., $G \supseteq H \iff E(G) \supseteq E(H)$. Let us consider the following two functions $f : \mathcal{L}_n \to \{0,1\}$ and $h : \mathcal{L}_n \to \mathbb{Z}$ on the lattice, which are the restrictions of $\mathrm{UBPM_n}$ and of the Permanent function, respectively:

$$\forall G \in \left(\mathrm{MC_n} \cup \{\hat{0}\}\right): \ f(G) = \mathrm{UBPM_n}(G) = \begin{cases} 1 & G \in PM(K_{n,n}) \\ 0 & \text{otherwise} \end{cases}$$

$$h(G) = \mathrm{per}(G) = \#\text{Perfect Matchings in } G.$$

These two functions are intimately related. Indeed, for any element G of the lattice, one can compute $h(G)$ by taking the sum $f(H)$ over all H in the downwards closed interval $[\hat{0}, G]$. Therefore, by an application of Möbius' Inversion Formula (Theorem 9) to the matching-covered lattice, we obtain:

$$\forall G \in \mathcal{L}_n: \; h(G) = \sum_{G \supseteq H \in \mathcal{L}_n} f(H) \iff \forall G \in \mathcal{L}_n: \; f(G) = \sum_{G \supseteq H \in \mathcal{L}_n} \mu(H, G) h(H)$$

where $\mu : \mathcal{L}_n \to \mathbb{Z}$ is the Möbius function of the lattice $\mathcal{L}_n$. A well known result due to Billera and Sarangarajan [5] states that $\mathcal{L}_n$ is isomorphic to the *face lattice* of the Birkhoff Polytope B_n, which is the convex hull of all $n \times n$ permutation matrices. Consequently, $\mathcal{L}_n$ is an Eulerian lattice – and its Möbius function μ is can be directly computed (see e.g. [30]), as follows:

$$\forall G, H \in \mathcal{L}_n, \; H \subseteq G: \quad \mu(H, G) = (-1)^{\operatorname{rank}(G) - \operatorname{rank}(H)}$$

where $\operatorname{rank}(x)$ denotes the maximal length of a chain from $\hat{0}$ to x (equivalently, $\operatorname{rank}(x) = dim(f_x) + 1$, where f_x is the face of B_n corresponding to the lattice element x). In [3] it was shown that the rank of every graph G in the matching-covered lattice is exactly $\chi(G) + 1$, where $\chi(G) = e(G) - v(G) + c(G)$ is the cyclomatic number, a topological quantity. Recalling our prior application of Möbius inversion, we obtain the following set of identities (note that the bottom element can be omitted, as $\operatorname{per}(\hat{0})$ is zero):

$$\forall G \in \mathcal{L}_n: \; (-1)^{\chi(G)} \sum_{G \supseteq H \in \mathrm{MC_n}} (-1)^{\chi(H)} \operatorname{per}(H) = \begin{cases} 1 & G \in \mathrm{PM}(K_{n,n}) \\ 0 & \text{otherwise.} \end{cases}$$

To conclude the proof, let us consider the following real multilinear polynomial, wherein we assign weight $(-1)^{\chi_G} \operatorname{per}(G)$ to every matching-covered graph:

$$p(x_{1,1}, \dots, x_{n,n}) = \sum_{G \in \mathrm{MC_n}} (-1)^{\chi(G)} \operatorname{per}(G) \prod_{(i,j) \in E(G)} x_{i,j}.$$

Let $G \subseteq K_{n,n}$ and observe that, by construction:

$$p(G) = \sum_{H \in \mathrm{MC_n}} (-1)^{\chi(H)} \operatorname{per}(H) \cdot \mathbb{1}\{E(H) \subseteq E(G)\} = \sum_{G \supseteq H \in \mathrm{MC_n}} (-1)^{\chi(H)} \operatorname{per}(H).$$

It remains to show that p "agrees" with $\mathrm{UBPM_n}$ on all inputs. It is not hard to see that it suffices to show this claim only for matching-covered graphs, since given any $G \subseteq K_{n,n}$ which is *not* matching-covered, one may consider the graph G' formed by the union of all perfect matching in G (in other words, the maximal matching-covered graph contained in G). By construction, we have $p(G') = p(G)$, and by definition, $\mathrm{UBPM_n}(G) = \mathrm{UBPM_n}(G')$ – thus, hereafter we consider only inputs $G \in \mathrm{MC_n}$. First, let us check the two trivial cases; the empty graph, and a single matching:

$$p(\hat{0}) = 0, \text{ and } p(M) = (-1)^{\chi(M)} = (-1)^{n-2n+n} = 1 \quad \forall M \in PM(K_{n,n}).$$

Finally, for any matching-covered graph G containing *more than a single matching*, i.e. $G \in \mathrm{MC_n}$ such that $G \notin PM(K_{n,n})$, it holds that:

$$p(G) = \sum_{G \supseteq H \in \mathrm{MC_n}} (-1)^{\chi(H)} \operatorname{per}(H) = 0$$

where the last equality follows from the identities obtained through Möbius' Inversion Formula. Thus, $p(x_{1,1}, \dots, x_{n,n})$ agrees with $\mathrm{UBPM_n}$ everywhere, and is its *unique* representation. ◀

3.1 Indicators on the Matching-Covered Lattice

We remark that the proof of Theorem 1 readily extends, through the same analysis using Möbius inversion, to any arbitrary *indicator function* over the matching-covered lattice. For any set $S \subseteq \mathrm{MC_n}$, let $I_S : \{0,1\}^{n^2} \to \{0,1\}$ be the Boolean function

$$\forall G \subseteq K_{n,n} : \ I_S(G) = \mathbb{1}\left\{ H \in S \text{ where } H = \bigcup_{M \in \mathrm{PM}(G)} M \right\}.$$

Then, the multilinear polynomial representing I_S is given by

$$I_S(x_{1,1}, \ldots, x_{n,n}) = \sum_{G \in \mathrm{MC_n}} \left((-1)^{\chi(G)} \sum_{H \in [\hat{0}, G] \cap S} (-1)^{\chi(H)+1} \right) \prod_{(i,j) \in E(G)} x_{i,j}.$$

3.2 Evasiveness and Generalized Decision Trees

The characterization of $\mathrm{UBPM_n}$ as a multilinear polynomial can be used to derive several complexity-theoretic corollaries. Firstly, this polynomial has *full total degree over* $\mathbb{R}$ and thus (see e.g. [6]):

▶ **Corollary 11.** $\mathrm{UBPM_n}$ *is evasive, i.e., any decision computing it has full depth,* n^2.

Let us remark that, contrary to its counterpart $\mathrm{BPM_n}$ which is a *monotone* bipartite graph property and thus known to be evasive [31], the *unique* matching function is *not monotone* and for such functions evasiveness is not guaranteed (see [23] for one such example). Theorem 1 can be also used to derive strong bounds (near evasiveness) versus larger classes of decision trees, for example trees whose internal nodes are labeled by arbitrary conjunctions of the input bits (hereafter AND-DT), and by arbitrary parity functions (XOR-DT). It is known [3] that the depth of any AND-DT computing a Boolean function f is at least $\log_3 |\mathrm{mon}(f)|$. Applying this to $\mathrm{UBPM_n}$ and recalling that asymptotically almost all balanced bipartite graphs are matching-covered ([3]), we have:

▶ **Corollary 12.** *Any* AND-DT *computing* $\mathrm{UBPM_n}$ *has depth at least* $(\log_3 2) \cdot n^2 - o_n(1)$.

As for parity decision trees, it is well known that the depth of any such tree is bounded by the total degree of its unique representing polynomial, over $\mathbb{F}_2$ (see [27, 3]). Noting that $\mathrm{per}(G) \equiv \det(G) \pmod 2$, we may write the $\mathbb{F}_2$-polynomial representation of $\mathrm{UBPM_n}$ as follows

$$\mathrm{UBPM_n}\,(x_{1,1}, \ldots, x_{n,n}) = \sum_{\substack{G \in \mathrm{MC_n} \\ \det(G) \equiv 1 \pmod 2}} \ \prod_{(i,j) \in E(G)} x_{i,j}.$$

Clearly this polynomial does not have full degree for any $n > 1$, as $\mathrm{per}(K_{n,n})$ is $n! \equiv 0 \pmod 2$[3]. Nevertheless, we claim that its $\mathbb{F}_2$-degree is at most a constant factor away from full. Observe that its monomials constitute precisely of all graphs that are both matching-covered, and whose biadjacency matrices are invertible over $\mathbb{F}_2$, i.e., are elements of the group $\mathrm{GL}_n(\mathbb{F}_2)$. However, asymptotically almost all graphs are matching-covered, and by a standard counting argument, the order of $\mathrm{GL}_n(\mathbb{F}_2)$ satisfies

[3] It is well known ([27]) that for any function $f : \{0,1\}^n \to \{0,1\}$, $\deg_2(f) = n \iff |f^{-1}(1)| \equiv 1 \pmod 2$. Therefore we obtain that the number of graphs $G \subseteq K_{n,n}$ containing a *unique perfect matching* is even, for any $n > 1$.

$$\Pr_{A\sim M_n(\mathbb{F}_2)}[A \in \mathrm{GL}_n(\mathbb{F}_2)] = \left(\tfrac{1}{2};\tfrac{1}{2}\right)_\infty \pm o_n(1)$$

where $\left(\frac{1}{2};\frac{1}{2}\right)_\infty \approx 0.28878$ is a Pochhammer symbol. Thus by a standard Chernoff argument, there exists a matching-covered graph with odd determinant and at least $\frac{1}{2}n^2 - o_n(1)$ edges.

▶ **Corollary 13.** $\mathrm{D}^{\mathrm{XOR}}(\mathrm{UBPM_n}) \geq \deg_2(\mathrm{UBPM_n}) \geq \left(\frac{1}{2} - o_n(1)\right) n^2$

4 The Dual Polynomial

In this section we consider the Boolean dual function (Definition 3) of $\mathrm{UBPM_n}$.

▶ **Definition 14.** *The function* $\mathrm{UBPM}^\star_\mathrm{n} : \{0,1\}^{n^2} \to \{0,1\}$ *is defined*

$$\mathrm{UBPM}^\star_\mathrm{n}(x_{1,1},\ldots,x_{n,n}) = \begin{cases} 1 & \left\{(i,j) : x_{i,j} = 0\right\} \subseteq K_{n,n} \textit{ does } \underline{\textit{not}} \textit{ have a unique P.M.} \\ 0 & \textit{otherwise.}\end{cases}$$

In what follows, we provide a full characterization of polynomial representing $\mathrm{UBPM}^\star_\mathrm{n}$. This description relies heavily on the that of another dual function – $\mathrm{BPM}^\star_\mathrm{n}$ – which is the dual of the bipartite perfect matching function $\mathrm{BPM_n}$ (which is defined identically to $\mathrm{UBPM_n}$, but without the *uniqueness* condition). The polynomial representation of $\mathrm{BPM}^\star_\mathrm{n}$ was obtained in a series of papers [3, 2]. Its monomials correspond to a family of graphs called "*totally ordered graphs*", and their coefficients are can be computed through a normal-form block decomposition of the aforementioned graphs. The full details are presented in [2], and are omitted here for brevity. In what follows, it suffices for us to denote

$$\mathrm{BPM}^\star_\mathrm{n}(x_{1,1},\ldots,x_{n,n}) = \sum_{G\subseteq K_{n,n}} a^\star_G \prod_{(i,j)\in E(G)} x_{i,j}.$$

Under this notation, our characterization of $\mathrm{UBPM}^\star_\mathrm{n}$ is the following.

▶ **Theorem 2.** *The unique polynomial representation of* $\mathrm{UBPM}^\star_\mathrm{n} : \{0,1\}^{n^2} \to \{0,1\}$ *is*

$$\mathrm{UBPM}^\star_\mathrm{n}(x_{1,1},\ldots,x_{n,n}) = \sum_{G\subseteq K_{n,n}} c^\star_G \prod_{(i,j)\in E(G)} x_{i,j}$$

where for every $G \subseteq K_{n,n}$ *we have:*

$$c^\star_G = \mathrm{per}(G)\cdot a^\star_G + \sum_{M\notin \mathrm{PM}(G)} (-1)^{|E(M)\setminus E(G)|}\cdot a^\star_{G\cup M}.$$

Proof. The polynomial representing $\mathrm{UBPM}^\star_\mathrm{n}$ can be expressed using $\mathrm{UBPM_n}$, via duality:

$$\mathrm{UBPM}^\star_\mathrm{n}(x_{1,1},\ldots,x_{n,n}) = 1 - \mathrm{UBPM_n}(1-x_{1,1},\ldots,1-x_{n,n}).$$

Substituting the characterization of Theorem 1 and expanding, we deduce that the coefficient of every graph $G \subseteq K_{n,n}$ in $\mathrm{UBPM_n}$ is:

$$c^\star_G = (-1)^{e(G)+1} \sum_{G\subseteq H\in \mathrm{MC_n}} (-1)^{\chi(H)}\,\mathrm{per}(H).$$

Writing $\mathrm{per}(H) = \sum_{M\in\mathrm{PM}(K_{n,n})} \mathbb{1}\{M \subseteq H\}$ and exchanging order of summation,

$$c^\star_G = (-1)^{e(G)+1} \sum_{M\in\mathrm{PM}(K_{n,n})} \sum_{G\subseteq H\in \mathrm{MC_n}} (-1)^{\chi(H)} \mathbb{1}\{M\subseteq H\}.$$

There are two possible cases in the above summation over all perfect matchings; either the matching is present in G, or it is not. Clearly every matching-covered graph containing G also contains any matching of G, so in the former case we get a contribution of $(-1)^{e(G)+1}\operatorname{per}(G)\cdot\sum_{G\subseteq H\in\mathrm{MC_n}}(-1)^{\chi(H)}$. As for the latter case, observe that for every $M\notin\mathrm{PM}(G)$, the set of matching-covered graphs containing G and M is exactly all matching-covered graphs containing $G\cup M$. Finally, we recall [3] that the coefficient of any graph $G\subseteq K_{n,n}$ in $\mathrm{BPM_n^\star}$ is given by:

$$a_G^\star=(-1)^{e(G)+1}\sum_{G\subseteq H\in\mathrm{MC_n}}(-1)^{\chi(H)}.$$

Putting the two together and simplifying, we obtain:

$$c_G^\star=\operatorname{per}(G)\cdot a_G^\star+\sum_{M\notin\mathrm{PM}(G)}(-1)^{|E(M)\setminus E(G)|}\cdot a_{G\cup M}^\star.$$ ◀

4.1 Corollary: The ℓ_1-norm of $\mathrm{UBPM_n^\star}$

One immediately corollary of Theorem 2 is the following fact: the multilinear polynomial representing $\mathrm{UBPM_n^\star}$ has *very low ℓ_1-norm* – i.e., it has few monomials, and the coefficient of every such monomial is not too large. A similar bound had previous been attained for $\mathrm{BPM_n^\star}$ in [2], which we heavily rely on for our proof.

▶ **Corollary 15.** *The ℓ_1-norm of $\mathrm{UBPM_n^\star}$ is bounded only by $\log_2\|\mathrm{UBPM_n^\star}\|_1=\Theta(n\log n)$.*

Proof. For the upper bound, we rely heavily on Theorem 2 and on the ℓ_1-norm of $\mathrm{BPM_n^\star}$ obtained in [2]. In the latter, it was shown that every coefficient in $\mathrm{BPM_n^\star}$ has magnitude at most 2^{2n}, and thus using the characterization of Theorem 2, the coefficient of any graph G satisfies

$$\log_2|c_G^\star|\le\log_2\left(\operatorname{per}(G)\cdot 2^{2n}+(n!-\operatorname{per}(G))\cdot 2^{2n}\right)\le n\log_2 n+n\log_2(4/e)+\Theta(\log n).$$

It remains to bound the *sparsity* of $\mathrm{UBPM_n^\star}$. To this end, consider the graphs whose coefficients do not vanish in $\mathrm{BPM_n^\star}$, and let us take a "ball" around every such graph $G\in\operatorname{mon}(\mathrm{BPM_n^\star})$, as follows:

$$B(G)=\Big\{H\subseteq K_{n,n}:\exists M\in\mathrm{PM}(K_{n,n})\text{ such that }E(H)\cup E(M)=E(G)\Big\}.$$

From Theorem 1 it follows that for every graph G, the coefficient $c_G^\star$ *does not vanish* only if either $G\in\operatorname{mon}(\mathrm{BPM_n^\star})$ or there exists some $H\in\operatorname{mon}(\mathrm{BPM_n^\star})$ such that $G\in B(H)$. However, each of the aforementioned balls is relatively small (in fact, can be bounded by $|B(G)|\le 2^n\cdot n!$), thus by the union bound:

$$|\operatorname{mon}(\mathrm{UBPM_n^\star})|\le|\operatorname{mon}(\mathrm{BPM_n^\star})|\,(1+2^n\cdot n!)=2^{\Theta(n\log n)}$$

where the last equality follows from the bound $\log_2|\operatorname{mon}(\mathrm{BPM_n^\star})|\le 2n\log_2 n+\mathcal{O}(n)$, obtained in [3]. This concludes the proof of the upper bound. The lower bound now follows directly from Theorem 1, as it suffices to observe that the coefficient of the complete bipartite graph is $\pm\operatorname{per}(K_{n,n})=\pm(n!)$. ◀

5 The Communication Rank of Unique Bipartite Matching

5.1 Rank and Polynomial Representation

The log-rank of a Boolean function is very closely related to its representation as a multilinear polynomial. This relationship is made very evident in the case of certain "lifted" functions: given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, one can define the following pair of functions $f_\wedge, f_\oplus : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, where

$$\forall x, y \in \{0,1\}^n : f_\wedge(x,y) = f(x \wedge y), \text{ and } f_\oplus(x,y) = f(x \oplus y).$$

It is well known [18, 4] that the rank of the communication matrices $M_{f_\wedge}$ and $M_{f_\oplus}$ is *exactly characterized* by the *sparsity* (i.e., number of monomials) of the polynomials representing f in the $\{0,1\}$-basis and the $\{\pm 1\}$-basis (the Fourier basis), respectively. In other words,

$$\begin{aligned}\operatorname{rank}(M_{f_\wedge}) &= \#\{\text{monomials in } \{0,1\}\text{-polynomial representing } f\}\\ \operatorname{rank}(M_{f_\oplus}) &= \#\{\text{monomials in } \{-1,1\}\text{-polynomial representing } f\}.\end{aligned}$$

The polynomial representation of a Boolean function f over the $\{0,1\}$-basis, or that of its dual $f^\star$, can also be used to derive communication rank upper bounds for non-lifted functions. The following lemma gives such a bound for the communication task of f, under *any* input partition.

▶ **Lemma 16.** *Let $f : \{0,1\}^n \to \{0,1\}$. Then, for every partition $S \sqcup \bar{S} = [n]$ we have:*

$$\operatorname{rank}(M_{f^{S\sqcup\bar{S}}}) \le \min\left\{ |\operatorname{mon}(f)|, |\operatorname{mon}(f^\star)| + 1 \right\}.$$

Proof. Let $S \sqcup \bar{S} = [n]$ be some input partition, and let M and M' be the communication matrices of f and $f^\star$ under this partition, respectively. By definition of Boolean duality, we have $M = J - M_\pi M' M_\sigma$ where $J = \mathbb{1} \otimes \mathbb{1}$ is the all-ones matrix, and M_π, M_σ are the permutation matrices for

$$\forall x \subseteq S : \ \pi(x) = S \setminus x, \quad \forall y \subseteq \bar{S} : \ \sigma(y) = \bar{S} \setminus y$$

therefore $|\operatorname{rank}(M) - \operatorname{rank}(M')| \le 1$, and it suffices to bound the rank of M'. However, we now observe that the polynomial representing f naturally induces a $|\operatorname{mon}(f)|$-rank decomposition of M (and likewise $f^\star$ for M'), as per [26], by considering the following sum of rank-1 matrices:

$$\forall T \in \operatorname{mon}(f), \text{ add the rank-1 matrix } a_T \cdot (\mathbb{1}_X \otimes \mathbb{1}_Y)$$

where a_T is the coefficient of T in f, and

$$X = \left\{ x : (T \cap S) \subseteq x \subseteq S \right\}, \quad Y = \left\{ y : \left(T \cap \bar{S}\right) \subseteq y \subseteq \bar{S} \right\}. \qquad ◀$$

5.2 The Rank of Unique Bipartite Matching

The log-rank of the unique bipartite matching function, ranging over all input partitions, is exactly characterized in the following Theorem.

▶ **Theorem 17.** *The log-rank of unique bipartite perfect matching is*

$$\max_{E \sqcup \bar{E} = E(K_{n,n})} \log \operatorname{rank}(M_{\mathrm{UBPM}_n^{E\sqcup\bar{E}}}) = \Theta(n \log n)$$

where $\mathrm{UBPM}_n^{E\sqcup\bar{E}}$ *is the two-party function whose input is partitioned according to* $E \sqcup \bar{E}$.

Proof. To obtain the lower bound, we must first fix a particular input partition. Assume without loss of generality that $n = 2m$ and let us partition the left and right vertices into two sets, $L = A \sqcup B$, $R = C \sqcup D$, where $A = \{a_1, \dots, a_m\}$, $B = \{b_1, \dots, b_m\}$, $C = \{c_1, \dots, c_m\}$ and $D = \{d_1, \dots, d_m\}$. Hereafter we consider the input partition wherein Alice receives all the edges incident to the left vertices A and Bob receives all the edges incident to the left vertices B. To prove our lower bound, we shall construct a *fooling set* (Definition 7). Let us introduce some notation: for every permutation $\pi \in S_m$ and two sets $X, Y \in \{A, B, C, D\}$, the notation $\pi(X, Y)$ refers to the matching from X to Y using the permutation π. Formally,

$$\forall X, Y \in \{A, B, C, D\} : \forall \pi \in S_m : \ \pi(X, Y) \overset{\text{def}}{=} \Big\{ \{x_i, y_{\pi(i)}\} : i \in [m] \} \Big\}.$$

Under this notation, we claim that

$$S = \Big\{ \big(\mathrm{id}(A, C) \ \sqcup \ \pi(A, D), \ \ \mathrm{id}(B, D) \ \sqcup \ \{\{b_{\pi(i)}, c_j\} : 1 \le i < j \le m\}\big) \ : \ \pi \in S_m \Big\}$$

is a fooling set for $\mathrm{UBPM}_n^{K_{A,R} \sqcup K_{B,R}}$, where $\mathrm{id} \in S_m$ is the identity element.

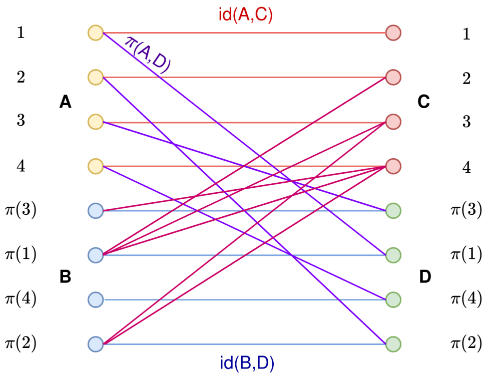


Figure 1 A graph G in the fooling set S, for $m = 4$ and $\pi = (2413)$.

$\underline{\{x \sqcup y : (x, y) \in S\} \subseteq \mathrm{UBPM}_n^{-1}(1)}$: Let $\pi \in S_m$ and consider $G \subseteq K_{n,n}$ where:

$$E(G) = \mathrm{id}(A, C) \ \sqcup \ \pi(A, D) \ \sqcup \ \mathrm{id}(B, D) \ \sqcup \ \big\{\{b_{\pi(i)}, c_j\}\big\}_{i<j}.$$

Clearly G has the identity perfect matching, whereby A is matched to C and B to D. Let us denote this matching by M. To show that M is *unique*, it suffices to show that there exists no M-alternating cycle in G. By construction, the vertices in any such cycle must alternate between $C - A - D - B$ (since the only edges joining $A \leftrightarrow C$ and $B \leftrightarrow D$ are those in the matching M). Thus, for any $i \in [m]$, an M-alternating path starting with c_i must be of the form:

$$c_i \sim a_i \sim d_{\pi(i)} \sim b_{\pi(i)} \sim c_j \sim \dots$$

where $j > i$. However, observe that $b_{\pi(m)}$ is not adjacent to any vertex in C, so any such path will eventually (after at most m passes through B) terminate at $b_{\pi(m)}$, without looping back to c_i. Therefore there exists no M-alternating cycle, and M is indeed unique.

$\underline{\forall (x_1, y_1), (x_2, y_2) \in S : \ (x_1 \sqcup y_2) \in \mathrm{UBPM_n}^{-1}(0)}$: Let $\pi, \sigma \in S_m$ where $\pi \ne \sigma$, and let G be the graph:

$$E(G) = \mathrm{id}(A, C) \ \sqcup \ \pi(A, D) \ \sqcup \ \mathrm{id}(B, D) \ \sqcup \ \big\{\{b_{\sigma(i)}, c_j\}\big\}_{i<j}.$$

Once again, clearly G has the identity matching M, whereby A is matched to C and B to D. To show that M is not unique, it suffices to exhibit an alternating cycle. Recall that $\sigma \neq \pi$ and therefore $\sigma^{-1} \circ \pi \neq \mathrm{id}$, and in particular, there exists some $i \in [m]$ such that $\sigma^{-1}(\pi(i)) < i$. By construction, the following M-alternating cycle is present in G:

$$c_i \sim a_i \sim d_{\pi(i)} \sim b_{\pi(i)} = b_{\sigma(\sigma^{-1}(\pi(i)))} \sim c_i.$$

Therefore, S is a fooling set for $\mathrm{UBPM_n}$ under the aforementioned input partition. To conclude the lower bound, we recall the following Theorem, due to Dietzfelbinger, Hromkovič and Schnitger [8]:

▶ **Theorem 18** ([8]). *$\forall f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}$ we have $\log_2 \mathrm{fs}(f) \leq 2\left(\log_2 \mathrm{rank}\, M_f + 1\right)$.*

Therefore, we have:

$$\log_2 \mathrm{rank}\left(M_{\mathrm{UBPM}_n^{K_{A,R} \sqcup K_{B,R}}}\right) \geq \tfrac{1}{2}\log_2 |S| - 1 = \tfrac{1}{4} n \log_2 n - \Theta(n)$$

concluding the lower bound. As for the upper bound, it follows directly from Lemma 16, and from the characterization of Theorem 2 (see Corollary 15). ◀

References

1 Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of Huang's sensitivity theorem. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

2 Gal Beniamini. The approximate degree of bipartite perfect matching. *arXiv preprint*, 2020. `arXiv:2004.14318`.

3 Gal Beniamini and Noam Nisan. Bipartite perfect matching as a real polynomial. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

4 Anna Bernasconi and Bruno Codenotti. Spectral analysis of boolean functions as a graph eigenvalue problem. *IEEE transactions on computers*, 48(3):345–351, 1999.

5 Louis J Billera and Aravamuthan Sarangarajan. The combinatorics of permutation polytopes. In *Formal power series and algebraic combinatorics*, volume 24, pages 1–23, 1994.

6 Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

7 Mark Bun and Justin Thaler. Guest column: Approximate degree in classical and quantum computing. *ACM SIGACT News*, 51(4):48–72, 2021.

8 Martin Dietzfelbinger, Juraj Hromkovič, and Georg Schnitger. A comparison of two lower-bound methods for communication complexity. *Theoretical Computer Science*, 168(1):39–51, 1996.

9 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

10 Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in Quasi-NC. *SIAM Journal on Computing*, 50(3):STOC16–218, 2019.

11 Harold N Gabow, Haim Kaplan, and Robert E Tarjan. Unique maximum matching algorithms. *Journal of Algorithms*, 40(2):159–183, 2001.

12 Martin Charles Golumbic, Tirza Hirst, and Moshe Lewenstein. Uniquely restricted matchings. *Algorithmica*, 31(2):139–154, 2001.

13 Gábor Hetyei. Rectangular configurations which can be covered by 2×1 rectangles. *Pécsi Tan. Foisk. Közl*, 8:351–367, 1964.

14 Thanh Minh Hoang, Meena Mahajan, and Thomas Thierauf. On the bipartite unique perfect matching problem. In *International Colloquium on Automata, Languages, and Programming*, pages 453–464. Springer, 2006.

15 John E Hopcroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

16 Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949–955, 2019.

17 Jeff Kahn, Michael Saks, and Dean Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984.

18 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.

19 Dexter Kozen, Umesh V Vazirani, and Vijay V Vazirani. NC algorithms for comparability graphs, interval graphs, and testing for unique perfect matching. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 496–503. Springer, 1985.

20 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, USA, 1996.

21 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, volume 79, pages 565–574, 1979.

22 László Lovász and Michael Saks. Lattices, Möbius functions and communications complexity. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 81–90. IEEE Computer Society, 1988.

23 Laszlo Lovasz and Neal E Young. Lecture notes on evasiveness of graph properties. *arXiv preprint cs/0205031*, 2002.

24 Kurt Mehlhorn and Erik M Schmidt. Las vegas is better than determinism in VLSI and distributed computing. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 330–337, 1982.

25 Noam Nisan. The demand query model for bipartite matching. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 592–599. SIAM, 2021.

26 Noam Nisan and Avi Wigderson. On rank vs. communication complexity. *Combinatorica*, 15(4):557–565, 1995.

27 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

28 M.D. Plummer and L. Lovász. *Matching Theory*. North-Holland Mathematics Studies. Elsevier Science, 1986.

29 Alexander A Sherstov. Algorithmic polynomials. *SIAM Journal on Computing*, 49(6):1173–1231, 2020.

30 Richard P. Stanley. *Enumerative Combinatorics: Volume 1*. Cambridge University Press, New York, NY, USA, 2nd edition, 2011.

31 Andrew Chi-Chih Yao. Monotone bipartite graph properties are evasive. *SIAM Journal on Computing*, 17(3):517–520, 1988.

A The Approximate Degree of $\mathrm{UBPM_n}$

The ε-approximate degree $\widetilde{\deg}_\epsilon(f)$, of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is the *least* degree of a real multilinear polynomial *approximating* f pointwise over $\{0,1\}^n$, with error at most ε. Formally,

▶ **Definition 19.** *Let $f : \{0,1\}^n \to \{0,1\}$ and let $0 < \varepsilon < \frac{1}{2}$. The ε-approximate degree of f, $\widetilde{\deg}_\epsilon(f)$, is the least degree of a real multilinear polynomial $p \in \mathbb{R}[x_1, \dots, x_n]$ such that:*

$$\forall x \in \{0,1\}^n : \ |f(x) - p(x)| \leq \varepsilon.$$

If $\varepsilon = {}^1\!/_3$, then we omit the subscript in the above notation, and instead write $\widetilde{\deg}(f)$.

Approximate degree is a well-studied complexity measure. For a comprehensive survey on the topic, we refer the reader to [7]. With regards to Theorem 2, we make the following observation: every Boolean function whose polynomial representation, or that of its dual, have low ℓ_1-norm – can be efficiently *approximated* in the ℓ_∞-norm by a low-degree polynomial. Firstly, it is not hard to see that for any Boolean function $f:\{0,1\}^n \to \{0,1\}$ and any $\varepsilon > 0$, the ε-approximate degree of f is identical to that of its dual $f^\star$. This follows since $f^\star$ can be obtained through an *affine transformation* of f, which cannot increase the degree, and the same transformation can similarly be applied to any approximating polynomial of f (and the converse follows since $(f^\star)^\star = f$). The second component of the approximation scheme is the following lemma.

▶ **Lemma 20** ([2], similar to [29]). *Let $f:\{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $p \in \mathbb{R}[x_1,\dots,x_n]$ be its representing polynomial, where $\|p\|_1 \in [3, 2^n]$. Then:*

$$\forall \, \|p\|_1^{-1} \le \varepsilon \le \frac{1}{3} : \quad \widetilde{\deg}_\epsilon(f) = \mathcal{O}\left(\sqrt{n \log \|p\|_1}\right).$$

The proof of Lemma 20 follows from the following simple approximation scheme: replace every *monomial* (of sufficiently large degree) with a polynomial that approximates it pointwise, to some sufficiently small error (depending only on the ℓ_1-norm of the representing polynomial). The full details of this scheme appeared previously in [2, 29]. Combining Lemma 20 with the ℓ_1-bound of Corollary 15, we obtain:

▶ **Corollary 21.** *For any $n > 1$, and $2^{-n\log n} \le \varepsilon \le \frac{1}{3}$, we have:*

$$\widetilde{\deg}_\epsilon(\mathrm{UBPM_n}) = \mathcal{O}(n^{3/2}\sqrt{\log n}).$$

B Families of Matching Functions having Low Dual ℓ_1-Norm

The main algorithmic result in this paper is the low ℓ_1-norm of the dual function of $\mathrm{UBPM_n}$, from which we deduce *upper bounds*, for instance on the communication rank and the approximate degree. In [2], a similar bound had been obtained for the dual of the perfect matching function, $\mathrm{BPM_n}$. These norm bounds and their corollaries extend to a wide range of *matching-related functions*, some of which are detailed below.

Functions Obtained by Restrictions

Consider any two Boolean functions f and g, such that g is obtained by a *restriction* of f (i.e., by fixing some of the inputs bits of f). As restrictions cannot increase the norm, it clearly holds that $\|g\|_1 \le \|f\|_1$ and $\|g^\star\|_1 \le \|f^\star\|_1$. Several intrinsically interesting matching-functions can be cast in this way. One notable example is the bipartite k-matching function, which is the indicator over all graphs $G \subseteq K_{n,n}$ containing a matching of size k.

$$\mathrm{BM_{n,k}}(x_{1,1},\dots,x_{n,n}) = \begin{cases} 1 & \{(i,j) : x_{i,j} = 1\} \subseteq K_{n,n} \text{ has a } k\text{-matching} \\ 0 & \text{otherwise.} \end{cases}$$

This function is obtained by a restriction of BPM_{2n-k}, as follows. Label the vertices of $K_{2n-k,2n-k}$ by

$$L = A \sqcup V, \text{ where } A = \{a_1,\dots,a_n\}, V = \{v_1,\dots,v_{n-k}\}$$
$$R = B \sqcup U, \text{ where } B = \{b_1,\dots,b_n\}, U = \{u_1,\dots,u_{n-k}\}.$$

Given any input $G \subseteq K_{n,n}$ to $\mathrm{BM}_{\mathrm{n,k}}$, the edges of G are encoded via the edges joining A and B, and moreover we fix two additional bicliques $K_{A,U}$, $K_{V,B}$. The resulting graph contains a bipartite perfect matching if and only if G contains a k-matching, and thus

▶ **Corollary.** *For every $0 < k \le n$, we have* $\log \left\|\mathrm{BM}^{\star}_{\mathrm{n,k}}\right\|_1 = \mathcal{O}(n \log n)$.

This norm bound is tight whenever $k = \alpha n$, for any *constant* $0 < \alpha < 1$, as are (up to log-factors) the bounds on the approximate degree and on the log-rank.

▶ **Corollary.** *Let $\alpha \in (0,1)$ be a constant. Then for every $n > 1$ and $2^{-n \log n} \le \varepsilon \le \frac{1}{3}$, we have:*

$$\log \left\|\mathrm{BM}^{\star}_{n,\alpha n}\right\|_1 = \Theta(n \log n), \quad \widetilde{\deg}_{\epsilon}\left(\mathrm{BM}_{n,\alpha n}\right) = \widetilde{\Theta}(n^{3/2}), \quad \textit{and} \ \log \operatorname{rank}\left(\mathrm{BM}_{n,\alpha n}\right) = \widetilde{\Theta}(n).$$

The aforementioned approximate degree lower bound follows using the method of *Spectral Sensitivity* – a complexity measure due to Aaronson, Ben-David, Kothari, Rao and Tal [1], based on Huang's proof of the sensitivity conjecture [16]. [1] proved that the approximate degree of any total function f is bounded below by the spectral radius of its *sensitivity graph* (i.e., the f-cut of the hypercube). As this graph is bipartite, its spectrum is symmetric, and it therefore suffices (by Cauchy interlacing) to obtain a lower bound on the spectral radius of any vertex induced subgraph of the sensitivity graph [2].

For $\mathrm{BM}_{\mathrm{n,k}}$ this construction is straightforward – consider the induced graph whose left vertices are all $(k-1)$-matchings, and right vertices are all k-matchings. This produces a biregular subgraph of the sensitivity graphs of $\mathrm{BM}_{\mathrm{n,k}}$, with left degrees $(n-k+1)^2$ and right degrees k. As it is well known that the spectral radius of a biregular graph is $\sqrt{d_L d_R}$ (where d_L and d_R are the left and right degrees, respectively), this concludes the bound on the spectral sensitivity of $\mathrm{BM}_{\mathrm{n,k}}$, and by extension, its approximate degree[4]. This lower bound on $\widetilde{\deg}\left(\mathrm{BM}_{\mathrm{n,k}}\right)$ now implies the ℓ_1-norm lower bound, through Lemma 20.

As for the log-rank lower bound, it follows by a simple fooling set argument, under the same input partition used in Theorem 17. Let $L = A \sqcup B$ be the left vertices corresponding to the input partition, where $|A| = |B| = n/2$, and let A' and B' be the first $k/2$ vertices of A and B, respectively. Let C be the first $k = \alpha n$ right vertices. Then, under the notation of Theorem 17,

$$S = \left\{ \left(\mathrm{id}(A', S), \mathrm{id}(B', \bar{S})\right) \ : \ S \subseteq C, \ \bar{S} = C \setminus S, \ |S| = |\bar{S}| = \tfrac{k}{2} \right\}$$

is a fooling set for $\mathrm{BM}_{n,\alpha n}$, where the indices of S and $\bar{S}$ correspond to a *fixed* ordering on C. Any pair (x,y) contains a k-matching, but for any mismatching pair belonging to sets $S_1 \neq S_2 \subseteq C$, we have that $S_1 \cap S_2 \neq \emptyset$ and thus the maximum matching is of size $|S_1 \cup S_2| < k$. By construction, this fooling set is of size

$$\log_2 |S| = \log_2 \binom{k}{k/2} = k - o(1)$$

and the log-rank bound now follows from Theorem 18.[5]

[4] We remark that the same construction also trivially extends to $\mathrm{UBM}_{\mathrm{n,k}}$; the *unique* k-matching function.

[5] For the *unique* bipartite k-matching function $\mathrm{UBM}_{\mathrm{n,k}}$ one can obtain a slightly stronger log-rank bound by repeating the construction of Theorem 17 with k-matchings rather than perfect matchings, and by adding $n-k$ isolated vertices. This yields a log-rank bound of $\log_2\left(k/2!\right) = \Theta(k \log k)$.

Formulas over Low-Norm Functions

Given any two nontrivial Boolean functions f and g, the norms of their conjunction, disjunction, and negation are at-most multiplicative in their respective norms, and the same holds for their duals. Therefore, the dual of any short De Morgan formula whose atoms are Boolean functions of low dual ℓ_1-norm, will similarly inherit the low-norm property. Several matching functions can be represented in this way, and thus have low dual norm. For example

$$\mathrm{MaxMatch}_{\mathrm{n,k}}(x_{1,1},\ldots,x_{n,n}) = \begin{cases} 1 & \text{The maximum matching of } \left\{(i,j) : x_{i,j}=1\right\} \text{ is of size } k \\ 0 & \text{otherwise} \end{cases}$$

can be constructed as $\mathrm{MaxMatch}_{\mathrm{n,k}} = \mathrm{BM}_{\mathrm{n,k}} \wedge\neg\, \mathrm{BM}_{n,k+1}$.

Fixed-Point Cycles and Approximate EFX Allocations

Benjamin Aram Berendsohn ✉

Institut für Informatik, Freie Universität Berlin, Germany

Simona Boyadzhiyska ✉

Institut für Mathematik, Freie Universität Berlin, Germany

László Kozma ✉

Institut für Informatik, Freie Universität Berlin, Germany

Abstract

We study edge-labelings of the complete bidirected graph $\overleftrightarrow{K}_n$ with *functions* that map the set $[d] = \{1, \ldots, d\}$ to itself. We call a directed cycle in $\overleftrightarrow{K}_n$ a *fixed-point cycle* if composing the labels of its edges in order results in a map that has a fixed point, and we say that a labeling is *fixed-point-free* if no fixed-point cycle exists. For a given d, we ask for the largest value of n, denoted $R_f(d)$, for which there exists a fixed-point-free labeling of $\overleftrightarrow{K}_n$. Determining $R_f(d)$ for all $d > 0$ is a natural Ramsey-type question, generalizing some well-studied zero-sum problems in extremal combinatorics. The problem was recently introduced by Chaudhury, Garg, Mehlhorn, Mehta, and Misra [EC 2021], who proved that $d \leq R_f(d) \leq d^4 + d$ and showed that the problem has close connections to *EFX allocations*, a central problem of fair allocation in social choice theory.

In this paper we show the improved bound $R_f(d) \leq d^{2+o(1)}$, yielding an efficient $(1 - \varepsilon)$-EFX allocation with n agents and $O((n/\varepsilon)^{0.67})$ unallocated goods; this improves the bound of $O((n/\varepsilon)^{0.8})$ of Chaudhury, Garg, Mehlhorn, Mehta, and Misra.

Additionally, we prove the stronger upper bound $2d - 2$, in the case where all edge-labels are *permutations*. A very special case of this problem, that of finding *zero-sum cycles* in digraphs whose edges are labeled with elements of $\mathbb{Z}_d$, was recently considered by Alon and Krivelevich [JGT 2021] and by Mészáros and Steiner [EJC 2021]. Our result improves the bounds obtained by these authors and extends them to labelings with elements of an arbitrary (not necessarily commutative) group, while also simplifying the proof.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Mathematics of computing → Extremal graph theory

Keywords and phrases fixed-point, zero-sum cycle, Ramsey theory, fair allocation, EFX

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.17

Related Version *Full Version*: https://arxiv.org/abs/2201.08753

Funding *Benjamin Aram Berendsohn*: Research supported by DFG grant KO 6140/1-1.
Simona Boyadzhiyska: Research supported by the Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg "Facets of Complexity" (GRK 2434).
László Kozma: Research supported by DFG grant KO 6140/1-1.

1 Introduction

Let $\overleftrightarrow{K}_n$ denote the complete bidirected graph on n vertices, that is, $\overleftrightarrow{K}_n$ has directed edges (u, v) and (v, u) for every pair u, v of distinct vertices. For an integer $d > 0$, a *d-labeling*, or simply *labeling*, ℓ of $\overleftrightarrow{K}_n$ is an assignment of a function $\ell_e \colon [d] \to [d]$ to each edge e of $\overleftrightarrow{K}_n$, where $[d] = \{1, \ldots, d\}$.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 17; pp. 17:1–17:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let C be a simple cycle of $\overset{\leftrightarrow}{K}_n$ with edges $e_1, \ldots, e_k$ (appearing in this order), where each edge e_i is labeled with a function f_i, for $i \in [k]$. We say that C is a *fixed-point cycle* if the function $f = f_1 \circ f_2 \circ \cdots \circ f_k = f_k(f_{k-1}(\cdots f_1(x) \cdots))$ has a fixed point, i.e., $f(x) = x$ for some $x \in [d]$. Observe that cyclically permuting the edges along C does not affect the existence of a fixed point.

Let $R_f(d)$ be the largest value n such that a d-labeling of $\overset{\leftrightarrow}{K}_n$ with no fixed-point cycle exists. Although not obvious a priori, the finiteness of $R_f(d)$ for all d can be seen through a simple reduction to Ramsey's theorem. In light of the known exponential bounds on multicolor Ramsey numbers [13, 20, 23], the bound obtained in this way is *doubly exponential* in d (see [12] for details).

A lower bound of $R_f(d) \geq d$ can be obtained through the following construction (found independently by various authors in different contexts). Say $V(\overset{\leftrightarrow}{K}_d) = [d]$, and label all edges (i, j) such that $i < j$ with the function $x \mapsto x$ and all other edges with the function $x \mapsto x + 1 \pmod{d}$. The avoidance of fixed-point cycles follows from the observation that every cycle contains at least one and at most $d - 1$ edges of the form (i, j) with $i > j$.

The question of determining $R_f(d)$ was recently raised by Chaudhury, Garg, Mehlhorn, Mehta, and Misra [12] in a slightly different, but equivalent form they call the *rainbow cycle problem.* They proved the upper bound $R_f(d) \leq d^4 + d$.

The motivation in [12] for introducing this problem comes from an application to *discrete fair division.* In particular, through an elegant and surprising connection, they showed that *polynomial* upper bounds on $R_f(d)$ yield nontrivial guarantees for the quality of allocations in a certain natural setting with a strong fairness condition.

EFX allocations

In economics and computational social choice theory, the *discrete fair division* problem asks to distribute a set of m indivisible goods among n agents in a *fair* way. The problem has a long history (see for example [22]) and different notions of fairness have been extensively studied, leading to a rich set of algorithmic and hardness results (e.g., see [8] or [9] and the references therein for an overview and precise definitions).

As simple examples show, complete *envy-freeness* cannot, in general, be achieved. One of the most compelling relaxations of this notion is *envy-freeness up to any good* (EFX). Informally, EFX means that no agent should prefer the goods received by any other agent to their own, if an arbitrary single good is removed from the other's set. The existence of EFX allocations is considered one of the central open questions of contemporary social choice theory (e.g., see [6, 9, 10, 12, 19]), and thus, various relaxations of it have been proposed. In particular, it is desirable to show that an EFX allocation exists (and can be efficiently computed) when (i) a certain global number t of goods are left unallocated, and (ii) envy-freeness is required to hold when every agent scales the values of others' goods by a factor of $1 - \varepsilon$ for some $0 \leq \varepsilon < 1$. Such an allocation is called $(1 - \varepsilon)$*-EFX with t unallocated goods.* It is desirable to minimize both t and ε, with the original EFX question requiring $t = \varepsilon = 0$.

Very recently, Chaudhury, Garg, Mehlhorn, Mehta, and Misra [12] introduced a correspondence between approximate EFX allocations and the fixed-point cycle problem described above (in their terminology, the *rainbow cycle problem*). Suppressing constant factors, the connection can be summarized as follows.

▶ **Theorem 1** (Theorem 4 in [12]). *For all $\varepsilon \in (0, 1/2]$, if $R_f(d) \in O(d^c)$ for some $c \geq 1$, then there exists a $(1 - \varepsilon)$-EFX allocation with $O((n/\varepsilon)^{\frac{c}{1+c}})$ unallocated goods, where n is the number of agents.*

Moreover, if the upper bound on $R_f(d)$ is *constructive* (i.e., a fixed-point cycle can be found in time polynomial in the number n of vertices whenever n exceeds the given bound on $R_f(d)$), then the claimed allocation can also be found in polynomial time. The result of $R_f(d) \leq d^4 + d$ from [12] is constructive; together with Theorem 1 it thus implies an efficient algorithm for computing an approximate EFX allocation with $O((n/\varepsilon)^{0.8})$ unallocated goods, the first guarantee of this kind with a *sublinear* (in n) number of unallocated goods.

Rainbow cycles

As remarked in [12], the question of determining $R_f(d)$ is in itself a natural question of extremal graph theory, independently of its application to EFX allocations.

Chaudhury, Garg, Mehlhorn, Mehta, and Misra formulated the problem in a slightly different, but essentially equivalent, way as follows. Given an n-partite digraph with each part having *at most* d vertices, a *rainbow cycle* is a cycle that visits each part *at most once*. The task is to determine the largest $n = n(d)$ for which such a digraph can avoid rainbow cycles, with the requirement that each vertex in part i has an incoming edge from at least one vertex in part j, for all distinct $i, j \in [n]$.

Note that we may assume that there is an extremal example containing *exactly* d vertices in each part. Indeed, if $\vec{G}$ is an extremal example and some part has fewer than d vertices, let v be any vertex from that part; then add the necessary number of new vertices, making them all "clones" of v, that is, every new vertex has the same in- and out-neighborhood as v. Then the newly obtained digraph has a rainbow cycle if and only if $\vec{G}$ does. Further, we may assume that each vertex v of $\vec{G}$ has *exactly* one incoming edge from every part other than its own. With these observations, the equivalence between the rainbow cycle problem and the fixed-point cycle problem is evident: we can view the bipartite digraph containing the edges from a part V_i to another part V_j as the mapping given by $y \mapsto x$, where $(x, y) \in E(V_i, V_j)$.

Our main result improves the upper bound of Chaudhury, Garg, Mehlhorn, Mehta, and Misra.

▶ **Theorem 2.** *For all $d > 0$ we have $R_f(d) \leq d^{2+o(1)}$.*

Similarly to the previous result, our bound is constructive. Combining Theorem 2 with Theorem 1 thus yields the following result about EFX allocations.

▶ **Corollary 3.** *For all $\varepsilon \in (0, 1/2]$, there exists an efficient $(1-\varepsilon)$-EFX allocation with $O((n/\varepsilon)^{0.67})$ unallocated goods.*

Zero-sum problems in extremal combinatorics

The problem of determining $R_f(d)$ can be cast as a generalization of some classical zero-sum problems in extremal combinatorics (this is somewhat less apparent in the original multipartite formulation of the problem). Zero-sum problems have received substantial attention and form a well-defined subfield of combinatorics, with an algebraic flavour. Perhaps the earliest result in this area is the Erdős-Ginzburg-Ziv theorem [14] which states that every collection of $2m - 1$ integers contains m integers whose sum modulo m is zero (see [3] for multiple proofs and extensions). Zero-sum problems *in graphs* typically ask, given an edge- or vertex-weighted graph, whether a certain substructure exists with zero total weight (modulo some fixed integer). Well-studied cases include complete graphs, cycles, stars, and trees (e.g., see [2, 5, 7, 11, 15, 21] for surveys and representative results).

More recently, for a given positive integer d, Alon and Krivelevich [4] asked for the maximum integer n such that the edges of the complete bidirected graph $\overleftrightarrow{K}_n$ can be labeled with integers so that there is no zero-sum cycle modulo d. In the following, we denote this quantity by $n = R_i(d)$. Alon and Krivelevich showed through an elegant probabilistic argument that $R_i(d) \in O(d \log d)$, with an improvement to $R_i(d) \in O(d)$ when d is prime. The application considered in [4] is finding cycles of length divisible by d in minors of complete graphs. It is easy to see this question as a special case of our fixed-point cycle problem: simply replace every edge-label k by the function $x \mapsto x + k \pmod{d}$. Zero-sum cycles in the original labeling are then in bijection with fixed-point cycles in our new labeling, and thus $R_i(d) \leq R_f(d)$.

Recently, Mészáros and Steiner [17] improved the result of Alon and Krivelevich, showing $R_i(d) \leq 8d - 1$, with further improvements for prime d. In fact, Mészáros and Steiner generalized the result, allowing the labels to come from an arbitrary commutative group of order d. The proof of the main result in [17] can be seen as an extension of an incremental construction of [4], combined with an intricate inductive argument that makes use of group-theoretic results.

We improve these results and show an upper bound of $R_i(d) \leq 2d - 2$ through somewhat similar, but arguably simpler arguments. Our result extends to arbitrary groups (not necessarily commutative). In fact, we prove the result in the more general setting of fixed-point cycles, when the edge-labels are restricted to *permutations* of $[d]$. The permutation case subsumes the integer case, since the functions $x \mapsto x + k \pmod{d}$ in the above reduction are permutations. Denoting the corresponding quantity by $R_p(d)$, we thus have $R_i(d) \leq R_p(d) \leq R_f(d)$, and (omitting the easy case $R_i(1) = R_p(1) = R_f(1) = 1$), prove the following.

▶ **Theorem 4.** *For all $d \geq 2$, we have $R_p(d) \leq 2d - 2$.*

By the above discussion, Theorem 4 implies that $R_i(d) \leq 2d - 2$. In fact, the following more general result holds.

▶ **Corollary 5.** *Let ℓ be a labeling of $\overleftrightarrow{K}_{2d-1}$ with elements of a (not necessarily abelian) group $(G, \cdot)$ of order d. Then there is a cycle whose labels multiply to the identity $1 \in G$.*

To our knowledge, the question of permutation labels has not been considered before. We see it as a natural problem of intermediate generality; it facilitates a simple proof for the integer case, allowing to sidestep the group-theoretic tools used in previous proofs.

In the case of permutation labels, it is natural to ask whether we can guarantee, instead of a fixed-point cycle, the existence of an *identity-cycle.* Indeed, Corollary 5 implies, as a very special case, that if $n \geq 2d! - 1$ and the edges of $\overleftrightarrow{K}_n$ are labeled with permutations of $[d]$, then there is a cycle whose labels compose to $\mathbf{1}_d$. The bound of $2d! - 1$ may appear rather loose, and one may wonder if a condition similar to that of Theorem 4, or at least one with polynomial dependence on d, would be sufficient. The following lower bound rules out this possibility.

▶ **Lemma 6.** *There exists a fixed $d_0 > 0$ such that for all $d \geq d_0$ and some $n \geq e^{\sqrt{d \ln d}}$, there is a labeling of $\overleftrightarrow{K}_n$ with permutations of $[d]$ such that there is no cycle whose labels compose to $\mathbf{1}_d$.*

Open questions

Closing the gap between the lower and upper bounds remains an interesting challenge for all three considered quantities ($R_f(d)$, $R_p(d)$, and $R_i(d)$). The lower bound construction with d vertices discussed earlier can be adapted to all three settings, and it remains a plausible conjecture that $R_i(d) = R_p(d) = R_f(d) = d$. This is easily verified [12] for $d \leq 3$ in the case of $R_f(d)$ (and hence for $R_p(d)$), and was verified via SAT-solvers [17] for $d \leq 6$ in the case of $R_i(d)$. Showing $R_f(d) \in O(d)$ would yield, via Theorem 1, an approximate EFX allocation with $O(\sqrt{n/\varepsilon})$ unallocated goods.

A further natural question is whether other classical zero-sum results from extremal combinatorics can be extended to the fixed-point setting.

Organization of the paper

In § 2 we introduce some useful terminology. In § 3 we prove Theorem 4, Corollary 5, and Lemma 6 and in § 4 we prove Theorem 2.

Independent work

Independently and concurrently to our work, Akrami, Chaudhury, Garg, Mehlhorn, and Mehta [1] also obtained the upper bound $R_p(d) \leq 2d - 2$, and an improved upper bound $R_f(d) \in O(d^2)$.

2 Preliminaries

Let $\overset{\leftrightarrow}{K}_n$ be given together with a labeling ℓ that assigns functions $[d] \to [d]$ to the edges. By *path* or *cycle* we always mean a *simple* path or cycle, and for a path P, we denote by $V(P)$ its set of vertices. All *edges* in this paper are directed, and an edge (u, v) is alternatively denoted by $u \to v$.

When we say that the edge $u \to v$ *maps* x to y, we mean that the function ℓ_{uv} assigned to $u \to v$ maps x to y. Similarly, a path (or cycle) $u_1 \to \cdots \to u_k$ maps x to y if, given the label f_i on $u_i \to u_{i+1}$ for each $i \in [k-1]$, we have $(f_1 \circ \cdots \circ f_{k-1})(x) = y$. For consistency, a path with only one vertex is said to map every value to itself.

We write $v{:}i$ for a pair of a vertex $v \in V(\overset{\leftrightarrow}{K}_n)$ and a value $i \in [d]$. By $u{:}x \xrightarrow{\ell} v{:}y$ (or simply $u{:}x \to v{:}y$ when the labeling ℓ is clear from the context) we denote the fact that the edge $u \to v$ maps x to y.

Let $W = (v_1{:}i_1, v_2{:}i_2, \ldots, v_k{:}i_k)$ be a sequence of vertex-value pairs with $v_j{:}i_j \to v_{j+1}{:}i_{j+1}$ for each $j \in [k-1]$. We call W a *valued walk* in $(\overset{\leftrightarrow}{K}_n, \ell)$. If $v_1 = v_k$, then W is a *valued circuit*. If all vertices in W are distinct, then W is a *valued path*. A valued circuit in which all vertices but the last are distinct is a *valued cycle*.

Finally, let $u, v, w \in V(\overset{\leftrightarrow}{K}_n)$ and $x, y \in [d]$. We say that w *routes* $u{:}x$ *to* $v{:}y$ if $u{:}x \to w{:}z \to v{:}y$ for some $z \in [d]$.

3 Permutation labels

In this section we prove our main result for permutation labels.

▶ **Theorem 4.** *For all $d \geq 2$, we have $R_p(d) \leq 2d - 2$.*

First, we introduce a useful tool, generalizing a technique used in [4, 17] to the fixed-point cycle setting.

Let ℓ be a labeling of $\overset{\leftrightarrow}{K}_n$ that assigns to every edge e a permutation $\ell_e \colon [d] \to [d]$. Consider an edge $u \to v$ of $\overset{\leftrightarrow}{K}_n$ with label ℓ_{uv}. Let $(g_i)_i$ and $(h_i)_i$ denote the labels of the incoming and outgoing edges at v, respectively. A *shifting at* $u \to v$ changes the labeling ℓ by replacing each g_i by $g_i \circ {\ell_{uv}}^{-1}$ and each h_i by $\ell_{uv} \circ h_i$. In particular, the label of $u \to v$ becomes the identity permutation $\mathbf{1}_d$. Let ℓ' be the resulting labeling. Observe that mappings along cycles remain unchanged, in particular $(\overset{\leftrightarrow}{K}_n, \ell')$ has a fixed-point cycle if and only if $(\overset{\leftrightarrow}{K}_n, \ell)$ does.

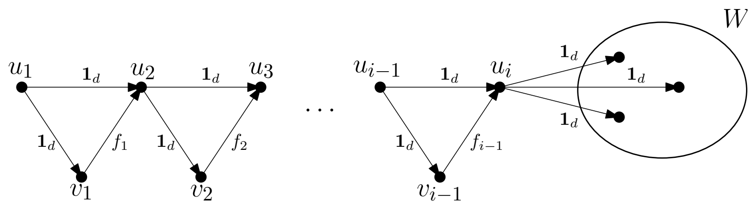


Figure 1 Illustration of the proof of Theorem 4.

Proof of Theorem 4. We start with the case $d = 2$. Suppose there is a permutation 2-labeling ℓ of $\overleftrightarrow{K}_3$ without a fixed-point cycle. The two possible labels are $\mathbf{1}_2$ and the function $\overline{\mathbf{1}}_2$ that maps $1 \mapsto 2$ and $2 \mapsto 1$. For each pair of vertices u, v the labels ℓ_{uv} and ℓ_{vu} must be distinct, otherwise we have a fixed-point cycle $u \to v \to u$. This means that the number of edges labeled $\overline{\mathbf{1}}_2$ in ℓ is precisely 3. Now consider two directed 3-cycles in $\overleftrightarrow{K}_3$ that form a partition of the edges. One of these cycles has to contain an even number of edges labeled $\overline{\mathbf{1}}_2$. Clearly, that cycle maps each value to itself, a contradiction. Thus, $R_p(2) \leq 2$.

Consider now the case $d \geq 3$; let $n \geq 2d - 1$, and let ℓ be an arbitrary permutation d-labeling of $\overleftrightarrow{K}_n$. We show that a fixed-point cycle exists.

We construct a "chain" consisting of vertices $u_1, \dots, u_j$ and $v_1, \dots, v_{j-1}$ (for some $j \leq d$) and transform the labeling so that each edge of the form $u_i \to u_{i+1}$ or $u_i \to v_i$ is labeled $\mathbf{1}_d$. By *step* i we mean either the edge $u_i \to u_{i+1}$ or the path $u_i \to v_i \to u_{i+1}$. Let $S_i \subseteq [d]$ denote the set of possible values to which 1 can be mapped along some path that concatenates steps $1, \dots, i-1$. Observe that if $S_i = [d]$ then we are done, since $u_i \to u_1$ maps some $x \in [d]$ to 1, and adding this edge to the path from u_1 to u_i that maps 1 to x yields a fixed-point cycle.

We construct the chain step-by-step (see Figure 1), ensuring that $|S_i| \geq i$ for all i. If we reach $|S_i| = d$, then we are done, having used at most $2d - 1$ vertices.

Pick vertex $u_1 \in V(\overleftrightarrow{K}_n)$ arbitrarily. The condition is then trivially satisfied.

Assume now that we have identified vertices $u_1, \dots, u_i$ and $v_1, \dots, v_{i-1}$ of the chain. Let $W \subseteq V(\overleftrightarrow{K}_n)$ denote the set of vertices *not used* yet, and shift at all edges $u_i \to u$ for $u \in W$. Observe that no label along the chain is affected. Now consider all edges *between* vertices in W. There are two possible cases.

Case 1: If some edge $v \to u$ (for $u, v \in W$) maps some element $x \in S_i$ to some element $y \notin S_i$, then extend the chain with $u_{i+1} = u$ and $v_i = v$. The set of reachable values becomes $S_{i+1} \supseteq S_i \cup \{y\}$, establishing the claim for the next step.

Case 2: All edges within W map S_i to S_i, and consequently $[d] \setminus S_i$ to $[d] \setminus S_i$. Since the chain has used up $2i - 1 \leq 2|S_i| - 1$ vertices, the digraph induced by W has at least $2d - 1 - (2|S_i| - 1) \geq 2\,|[d] \setminus S_i|$ vertices. If $i \leq d - 2$, then $|[d] \setminus S_i| \geq 2$, and we can argue inductively that the digraph induced by W has a fixed-point cycle. If $i = d - 1$, then $|[d] \setminus S_i| = 1$ and $|W| \geq 2$, so we trivially have a fixed-point cycle. ◀

We next show the extension of the result to the case of labels from an arbitrary (not necessarily abelian) group.

▶ **Corollary 5.** *Let ℓ be a labeling of $\overleftrightarrow{K}_{2d-1}$ with elements of a (not necessarily abelian) group $(G, \cdot)$ of order d. Then there is a cycle whose labels multiply to the identity $1 \in G$.*

Proof. Let ℓ be a labeling that assigns elements of the group $(G, \cdot)$ of order d to edges of $\overleftrightarrow{K}_{2d-1}$. Construct a labeling ℓ' of $\overleftrightarrow{K}_{2d-1}$, assigning the function $x \mapsto x \cdot k$ to every edge with label k in ℓ. By Theorem 4, ℓ' has a fixed-point cycle. Suppose its labels are $f_1, \dots, f_t$, where $f_i(x) = x \cdot k_i$ for all $i \in [t]$. Then $(f_1 \circ \cdots \circ f_t)(x) = x$ for some $x \in G$, which implies that $k_1 \cdot \cdots \cdot k_t = 1 \in G$. ◀

Finally, we prove that to guarantee a cycle whose labels compose to the identity permutation, we need a *super-polynomial* number of vertices.

▶ **Lemma 6.** *There exists a fixed $d_0 > 0$ such that for all $d \geq d_0$ and some $n \geq e^{\sqrt{d \ln d}}$, there is a labeling of $\overleftrightarrow{K}_n$ with permutations of $[d]$ such that there is no cycle whose labels compose to $\mathbf{1}_d$.*

Proof. For given $d > 0$, let $n = n(d)$ denote the *maximum order* of a permutation of $[d]$, that is, n is the largest integer for which there exists a permutation $\pi\colon [d] \to [d]$ such that $\pi^n = \mathbf{1}_d$ but $\pi^k \neq \mathbf{1}_d$ for all $1 \leq k < n$. The function $n(d)$ is *Landau's function*, and it is well known [16, 18] that $n(d) = e^{(1+o(1))\sqrt{d \ln d}}$. (One can obtain high-order permutations by combining cycles of different prime lengths.)

The construction is now similar to the one used earlier. Let $\pi\colon [d] \to [d]$ be a permutation of order n, let $V(\overleftrightarrow{K}_n) = [n]$, and label all edges (i, j) such that $i < j$ with $\mathbf{1}_d$ and all other edges with π. Since every cycle contains at least one and at most $n - 1$ edges of the form (i, j) with $i > j$, composing the labels along a cycle cannot yield $\mathbf{1}_d$. ◀

4 General function labels

Before proving our main theorem, we first present a weaker result, in order to introduce some techniques that will be used later. The result already improves the bound from [12], while using a similar argument.

▶ **Lemma 7.** *For all $d > 0$ we have $R_f(d) \leq d^3 - d^2 + d$.*

Proof. Suppose that $n \geq d^3 - d^2 + d + 1$, and let ℓ be an arbitrary d-labeling of $\overleftrightarrow{K}_n$. We show that a fixed-point cycle exists. For that, we proceed algorithmically.

Partition $V(\overleftrightarrow{K}_n)$ arbitrarily into two parts $V = \{v_1, \ldots, v_d\}$ and $U = \{u_1, \ldots, u_{d^3-d^2+1}\}$, and consider all vertices of U initially *unmarked*.

In the preprocessing phase, for each triplet $(i, x, y) \in [d-1] \times [d]^2$ in turn, check whether there exists an unmarked vertex $u \in U$ that routes $v_i{:}x$ to $v_{i+1}{:}y$. If yes, then *mark* u, and say that u is *responsible* for the triplet (i, x, y).

Observe that as we mark at most $(d-1)d^2$ vertices of U, we have at least one unmarked vertex remaining in U after the preprocessing phase. Let $c \in U$ be such a vertex. Consider now the walk that alternates between visiting c and visiting the vertices of V in order, giving rise to the valued walk $W = (c{:}c_0, v_1{:}x_1, c{:}c_1, v_2{:}x_2, \ldots, v_d{:}x_d, c{:}c_d)$, where $x_i, c_i \in [d]$ for all $i \in [d]$ and $c_0 = 1$. Since $|\{c_0, \ldots, c_d\}| \leq d$, there must be some i, j such that $0 \leq i < j \leq d$ and $c_i = c_j$, yielding the valued fixed-point *circuit* $C = (c{:}c_i, v_{i+1}{:}x_{i+1}, \ldots, v_j{:}x_j, c{:}c_j)$.

It remains to transform C into a cycle. For all k such that $i < k < j$ replace the subpath $v_k \to c \to v_{k+1}$ in C by $v_k \to c' \to v_{k+1}$, where $c' \in U$ is the *unique* marked vertex that is responsible for the triplet (k, x_k, x_{k+1}). Such a vertex must exist in U, for otherwise c itself would have been chosen as responsible for this triplet in the preprocessing phase. We have removed all occurrences of c in C but the first and last, and we have not changed the mappings of values; thus we obtain a fixed-point cycle. An efficient algorithm for finding this cycle is implicit in the proof. ◀

Next, we introduce a transformation that will be useful in the main proof.

Compression

Given a d-labeling ℓ of $\overleftrightarrow{K}_n$, define the *imageset* of a vertex v as $\mathrm{im}(v) = \{y \mid u{:}x \xrightarrow{\ell} v{:}y,$ for some $u, x\}$. In words, $\mathrm{im}(v)$ is the subset of values in $[d]$ that can be mapped to by edges to v.

We say that v is *k-compressed* if $|\mathrm{im}(v)| \leq k$. Observe that all vertices are trivially d-compressed, and the existence of a 1-compressed vertex would immediately yield a fixed-point cycle. Indeed, if v is 1-compressed, then the cycle $v \to u \to v$ maps the unique element in $\mathrm{im}(v)$ to itself for any $u \in V(\overleftrightarrow{K}_n)$.

We now describe the compression operation, illustrated in Figure 2. Let w be a k-compressed vertex for some $k \geq 2$ and $w' \in V(\overleftrightarrow{K}_n) \setminus \{w\}$. Suppose there exist two paths P_1 and P_2 from w to w', together with two distinct values $i_1, i_2 \in \mathrm{im}(w)$ and $j \in [d]$ such that P_1 maps i_1 to j and P_2 maps i_2 to j. Note that the sets of interior vertices of P_1 and P_2 do not need to be disjoint and that either of the paths may consist of a single edge.

Define the function $f\colon [d] \to [d]$ as follows: let $f(i_2) = j$; for all $x \in \mathrm{im}(w) \setminus \{i_2\}$, let $f(x) = y$, where P_1 maps x to y; for all $x \in [d] \setminus \mathrm{im}(w)$, choose $f(x)$ arbitrarily. Now, delete all vertices of P_1 and P_2, and add a new vertex $w^\star$ with edges to and from all remaining vertices. For all $v \in V(\overleftrightarrow{K}_n) \setminus (V(P_1) \cup V(P_2))$, if the edge $v \to w$ had the label g, then the edge $v \to w^\star$ gets the label $g \circ f$, and if an edge $w' \to u$ had the label h, then the edge $w^\star \to u$ gets the label h. The labels of edges not involving $w^\star$ remain unchanged.

We refer to this operation as *compressing P_1 and P_2 to $w^\star$*. Suppose we are left with n' vertices and observe that $n - n' \leq |V(P_1)| + |V(P_2)| - 3$, since P_1 and P_2 have common endpoints. Let ℓ' denote the resulting labeling of $\overleftrightarrow{K}_{n'}$. We prove two crucial properties.

▶ **Lemma 8.** *Suppose that P_1 and P_2 (with starting vertex w) are compressed to $w^\star$ in the way described above, resulting in a labeling ℓ' of $\overleftrightarrow{K}_{n'}$.*

(i) *If w is k-compressed in $(\overleftrightarrow{K}_n, \ell)$, then $w^\star$ is $(k-1)$-compressed in $(\overleftrightarrow{K}_{n'}, \ell')$.*

(ii) *If $(\overleftrightarrow{K}_{n'}, \ell')$ has a fixed-point cycle, then $(\overleftrightarrow{K}_n, \ell)$ has a fixed-point cycle.*

Proof.

(i) Let $S \subseteq [d]$ denote the set of values to which values in $\mathrm{im}(w) \setminus \{i_2\}$ are mapped by P_1. Clearly $|S| \leq |\mathrm{im}(w)| - 1 \leq k - 1$. Since P_2 maps i_2 to j and $j \in S$ by construction (as P_1 maps i_1 to j), every edge $v \to w^\star$ maps all values in $[d]$ to S and thus $|\mathrm{im}(w^\star)| \leq |\mathrm{im}(w)| - 1$.

(ii) If a fixed-point cycle in $(\overleftrightarrow{K}_{n'}, \ell')$ avoids $w^\star$, then it also exists in $(\overleftrightarrow{K}_n, \ell)$. Otherwise, suppose a cycle in $(\overleftrightarrow{K}_{n'}, \ell')$ contains the segment $v{:}x \to w^\star{:}y \to u{:}z$. Then, in $(\overleftrightarrow{K}_n, \ell)$, the edge $v \to w$ maps x to some value in $\mathrm{im}(w)$ that is mapped by P_1 or P_2 to $w'{:}y$, and $w' \to u$ maps y to z. Replacing $w^\star$ by P_1 or P_2 thus gives a fixed-point cycle in $(\overleftrightarrow{K}_n, \ell)$. Given a fixed-point cycle in $(\overleftrightarrow{K}_{n'}, \ell')$, a fixed-point cycle in $(\overleftrightarrow{K}_n, \ell)$ can be reconstructed (i.e., the compression can be undone) efficiently, with minor bookkeeping.

In the remainder of the section we prove our main theorem.

▶ **Theorem 2.** *For all $d > 0$ we have $R_f(d) \leq d^{2+o(1)}$.*

The high-level strategy is similar to the one used in the proof of Lemma 7. The main difference is that, instead of trying to build a cycle using a sequence of d designated vertices $(v_1, \ldots, v_d)$ and a well-chosen center-vertex (c), we pick a sequence of *far fewer* designated vertices, and use the structure imposed upon them by a special compressed vertex. We may fail to find a fixed-point cycle (or even a circuit) with any candidate center. In that case, however, we make progress by *compressing* the special vertex further. After repeating the

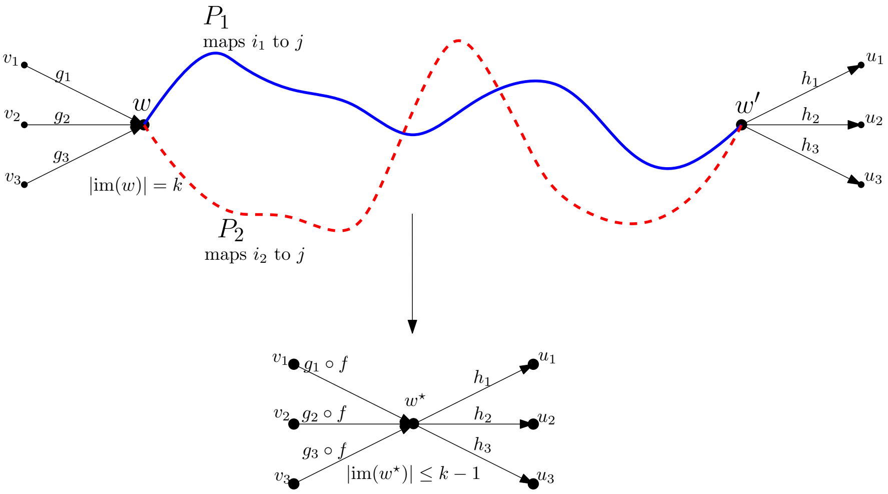


Figure 2 Compressing P_1 and P_2 to $w^\star$.

process sufficiently many times, we will have created a large number of highly compressed vertices. The mappings between these vertices are sufficiently restricted that we can find a fixed-point cycle in the digraph induced by them, through a *recursive application* of the same procedure.

The following lemma describes the key *win-win* step of the procedure: we either find a fixed-point cycle, or identify two paths that allow the compression of a vertex.

▶ **Lemma 9.** *Let $k \geq 2$, $n \geq 4d\lceil \frac{d}{k} \rceil^2 + 2\lceil \frac{d}{k} \rceil + 2$, and $w \in V(\overleftrightarrow{K}_n)$ be a k-compressed vertex. Then either $(\overleftrightarrow{K}_n, \ell)$ has a fixed-point cycle, or there exist a vertex $u \in V(\overleftrightarrow{K}_n) \setminus \{w\}$, values $i_1, i_2 \in \mathrm{im}(w)$ with $i_1 \neq i_2$ and $x \in [d]$, and two paths on at most $4\lceil \frac{d}{k} \rceil + 2$ vertices each from $w{:}i_1$ and $w{:}i_2$ to $u{:}x$.*

Proof. Let $q = 2\lceil \frac{d}{k} \rceil + 1$ and arbitrarily fix a q-subset $V \subseteq V(\overleftrightarrow{K}_n) \setminus \{w\}$. Write $V = \{v_1, v_2, \ldots, v_q\}$ and $U = V(\overleftrightarrow{K}_n) \setminus (V \cup \{w\})$. Notice that $|U| \geq 4d\lceil \frac{d}{k} \rceil^2$.

Call a vertex $c \in U$ *valid* for (i, x, y), where $i \in [q-1]$ and $x, y \in [d]$, if c routes $v_i{:}x$ to $v_{i+1}{:}y$ and there are at least $q-1$ vertices in U that route $v_i{:}x$ to $v_{i+1}{:}y$ (including c). Let k_c denote the number of triples (i, x, y) for which c is *not* valid. Double-counting yields

$$\sum_{c \in U} k_c \leq (q-1) \cdot d^2 \cdot (q-2) < 4d^2 \left\lceil \frac{d}{k} \right\rceil^2.$$

Thus, by the pigeonhole principle, there exists a $c \in U$ with $k_c < \frac{4d^2\lceil \frac{d}{k} \rceil^2}{|U|} \leq d$. Fix such a c.

Recall that $|\mathrm{im}(w)| = k$ and assume without loss of generality that $\mathrm{im}(w) = [k]$. For each $i \in [k]$, we construct a (valued) walk W_i that starts with $v_1{:}\ell_{wv_1}(i)$ and visits $v_2, \ldots, v_q$ in that order, possibly taking a detour through c at every step.

Fix $i \in [k]$. We describe the construction of W_i by iteratively constructing prefixes W_i^j that start with v_1 and end with v_j. First, let $W_i^1 = v_1{:}x_1$, where $x_1 = \ell_{wv_1}(i)$.

For $1 < j \leq q$, suppose that W_i^{j-1} ends with $v_{j-1}{:}x_{j-1}$, and consider the values y_j, x_j reached in the valued path $v_{j-1}{:}x_{j-1} \to c{:}y_j \to v_j{:}x_j$. If c is valid for $(j-1, x_{j-1}, x_j)$, then let $W_i^j = W_i^{j-1} \to c{:}y_j \to v_j{:}x_j$. Otherwise, let $W_i^j = W_i^{j-1} \to v_j{:}x'_j$ for the appropriate x'_j. Finally, let $W_i = W_i^q$. Note that W_i contains at most $2q-1$ vertices (including up to $q-1$ occurrences of c).

Observe that in the process of constructing $W_1, \dots, W_k$, we make $(q-1) \cdot k \geq 2d$ steps in total, where each step either adds c to the current walk, or finds that c is not valid for some triple $(j-1, x_{j-1}, x_j)$.

We claim that some vertex-value pair occurs at least twice in all the valued walks $W_1, W_2, \dots, W_k$. Suppose not. Then, in particular, c occurs at most d times. Moreover, the triples $(j-1, x_{j-1}, x_j)$ for which c is not valid that we encounter in the construction of the walks must be pairwise distinct (if $(j-1, x_{j-1}, x_j)$ occurs twice in this way, then two distinct paths must contain $v_{j-1}{:}x_{j-1}$). This means that we encounter such triples at most k_c times in total. The total number of steps is then at most $d + k_c < 2d$, contradicting our earlier observation. This proves the claim.

First, consider the case where some $v{:}x$ occurs in *two different walks* W_{i_1}, W_{i_2}. Define W'_{i_1} and W'_{i_2} by removing all vertices after v from W_{i_1} and W_{i_2}, respectively. We turn W'_{i_1} and W'_{i_2} into (simple) paths as follows. Suppose c occurs more than once in W'_{i_1}. Suppose W'_{i_1} contains the segment $v_i{:}x_i \to c{:}y_i \to v_{i+1}{:}x_{i+1}$. If this is not the first occurrence of c, then we simply replace c by some vertex $c' \in U \setminus \{c\}$ that routes $v_i{:}x_i$ to $v_{i+1}{:}x_{i+1}$ and has not been used in this way before. Note that we do this at most $q-2$ times (once for each occurrence of c in W'_{i_1} except for the first one), and there are at least $q-2$ such vertices in $U \setminus \{c\}$ by construction, so the operation is well-defined. We proceed the same way for W'_{i_2} (recall that the paths needed for the compression operation need not be internally disjoint). Call the resulting (simple) paths W''_{i_1} and W''_{i_2}, respectively. Then $w{:}i_1 \to W''_{i_1}$ and $w{:}i_2 \to W''_{i_2}$ are the desired paths of at most $2q \leq 4\lceil \frac{d}{k} \rceil + 2$ vertices, and we are done.

Second, suppose that some $v{:}x$ occurs twice *in a single walk* $W = W_i$. Then $v = c$, since each v_i occurs only once in W. We now find a sub-walk of W that has a fixed point. Remove all vertices before the first occurrence of $c{:}x$ and all vertices after the second occurrence of $c{:}x$, and call the resulting valued walk W'. Clearly, W' is a fixed-point walk and contains at most $q-1$ occurrences of c (counting both start and end individually). Then, similarly as in the first case, transform W' into a simple cycle by replacing all occurrences of c, except at the start/end, by suitable vertices (at most $q-3$ of them) in $U \setminus \{c\}$. The result is a fixed-point cycle. ◀

We prove Theorem 2 via the following recurrence.

▶ **Lemma 10.** *For all $d \geq 2^9$, we have $R_f(d) < 5d^2 + 9d \log_2 d \cdot (R_f(\lfloor \sqrt{d} \rfloor) + 1)$.*

Proof. Set $n = 5d^2 + 9d \log_2 d \cdot (R_f(\lfloor \sqrt{d} \rfloor) + 1)$ and consider an arbitrary labeling ℓ of $\overset{\leftrightarrow}{K}_n$. We show that $(\overset{\leftrightarrow}{K}_n, \ell)$ contains a fixed-point cycle. We again proceed algorithmically. Our strategy is to perform transformations on $\overset{\leftrightarrow}{K}_n$ and ℓ, such as to compress vertices using Lemma 9. We say that a $\lfloor \sqrt{d} \rfloor$-compressed vertex is *fully compressed*.

We first observe that if we have more than $R_f(\lfloor \sqrt{d} \rfloor)$ fully compressed vertices, then we are done. Indeed, consider the subdigraph induced by the set F of vertices that are fully compressed. For all $u, v \in F$, restrict each function ℓ_{uv} to an arbitrary subset of $[d]$ of size $\lfloor \sqrt{d} \rfloor$ containing $\mathrm{im}(u)$. Applying an arbitrary bijection from each such set to $[\lfloor \sqrt{d} \rfloor]$, transform the restricted labeling into a valid $\lfloor \sqrt{d} \rfloor$-labeling on the subdigraph induced by F without creating new fixed-point cycles. Then, if $|F| > R_f(\lfloor \sqrt{d} \rfloor)$, we can recursively find a fixed-point cycle in the induced subdigraph, from which we can recover a fixed-point cycle in $(\overset{\leftrightarrow}{K}_n, \ell)$.

Now we explain how we obtain the required number of fully compressed vertices. Let T be an arbitrary set of $R_f(\lfloor \sqrt{d} \rfloor) + 1$ vertices which are to be compressed and let S be the remaining set of vertices of $\overset{\leftrightarrow}{K}_n$. At each step, let $w \in T$ be any vertex that is *not* fully compressed; say w is k-compressed. Apply Lemma 9 to w and the subdigraph induced by $\{w\} \cup S$. If we find a fixed-point cycle, then we are immediately done by Lemma 8.

Otherwise, we find two paths P_1 and P_2 starting at w that we can compress into a single $(k-1)$-compressed vertex $w^\star$. Now set $T = T \setminus \{w\} \cup \{w^\star\}$ and $S = S \setminus (V(P_1) \cup V(P_2))$. Note that the size of S reduces by at most $8\lceil \frac{d}{k} \rceil + 1 \leq 8\frac{d}{k} + 9$.

We need to ensure that we always have enough vertices to apply Lemma 9. First we count the number of vertices that get removed from S throughout the process. Note that, for each k such that $\lfloor \sqrt{d} \rfloor + 1 \leq k \leq d$, we transform a k-compressed vertex of T into a $(k-1)$-compressed vertex of T at most $R_f(\lfloor \sqrt{d} \rfloor) + 1$ times, and every time we perform such an operation, we remove at most $8\frac{d}{k} + 9$ vertices from S. Thus, the number of vertices we remove throughout the process is at most

$$\begin{aligned}
(R_f(\lfloor \sqrt{d} \rfloor) + 1) \sum_{k=\lfloor \sqrt{d} \rfloor + 1}^{d} \left(8\frac{d}{k} + 9\right) &\leq (R_f(\lfloor \sqrt{d} \rfloor) + 1)\,(8d \log_2 d + 9d - 1) \\
&\leq (R_f(\lfloor \sqrt{d} \rfloor) + 1)(9d \log_2 d - 1),
\end{aligned}$$

where the first inequality uses the fact that $\sum_{k=1}^{d} 1/k \leq \log_2 d$, and the second inequality uses our assumption $\log_2 d \geq 9$.

Also accounting for the $R_f(\lfloor \sqrt{d} \rfloor) + 1$ vertices in T, it follows that the final set S has at least $5d^2$ remaining vertices, which is enough to ensure that we can apply Lemma 9. Indeed, the number of additional vertices needed to apply Lemma 9 is

$$\begin{aligned}
4d\left\lceil \frac{d}{k} \right\rceil^2 + 2\left\lceil \frac{d}{k} \right\rceil + 1 &\leq 4d(\sqrt{d}+1)^2 + 2(\sqrt{d}+1) + 1 \\
&\leq 5d^2,
\end{aligned}$$

since we always have $k \geq \lfloor \sqrt{d} \rfloor + 1 \geq \sqrt{d}$, and $d \geq 16$. Thus, the process terminates successfully, leaving us with the set T of $R_f(\lfloor \sqrt{d} \rfloor) + 1$ fully compressed vertices, using which we find a fixed-point cycle, as discussed above. ◀

Finally, we show that the above recurrence solves to $R_f(d) \in \mathcal{O}(d^2 \cdot 2^{(\log_2 \log_2 d)^2})$, thus implying Theorem 2.

▶ **Lemma 11.** *For all $d \geq 4$, we have $R_f(d) \leq 16 \cdot d^2 \cdot 2^{(\log_2 \log_2 d)^2}$.*

Proof. If $d < 2^{20}$, the claim follows by Lemma 7, since $16 \cdot d^2 \cdot 2^{(\log_2 \log_2 d)^2} \geq d^3 - d^2 + d$ for all $d < 2^{20}$. Assume therefore that $d \geq 2^{20}$. Denoting $C = 16$ we have:

$$\begin{aligned}
R_f(d) &\leq 5d^2 + 9d \log_2 d \cdot (R_f(\lfloor \sqrt{d} \rfloor) + 1) && \text{(from Lemma 10)} \\
&\leq 5d^2 + 9d \log_2 d \cdot \left(C \cdot (\lfloor \sqrt{d} \rfloor)^2 \cdot 2^{(\log_2 \log_2 \lfloor \sqrt{d} \rfloor)^2} + 1\right) && \text{(by induction)} \\
&\leq 5d^2 + 9d \log_2 d \cdot \left(C \cdot d \cdot 2^{(\log_2 \log_2 \sqrt{d})^2} + 1\right) && \text{(using } \lfloor \sqrt{d} \rfloor \leq \sqrt{d}\text{)} \\
&\leq 6d^2 + 9C \cdot d^2 \log_2 d \cdot 2^{(\log_2 \log_2 \sqrt{d})^2} && \text{(using } 9 \log_2 d \leq d, \text{ for } d \geq 52\text{)} \\
&\leq 10C \cdot d^2 \log_2 d \cdot 2^{(\log_2 \log_2 \sqrt{d})^2} && \text{(using } d \geq 2^6\text{)} \\
&= 10C \cdot d^2 \log_2 d \cdot 2^{((\log_2 \log_2 d)-1)^2} \\
&= 10C \cdot d^2 \cdot 2^{((\log_2 \log_2 d)-1)^2 + \log_2 \log_2 d} \\
&= C \cdot d^2 \cdot 2^{(\log_2 \log_2 d)^2} \cdot \left(10 \cdot 2^{1 - \log_2 \log_2 d}\right) \\
&= C \cdot d^2 \cdot 2^{(\log_2 \log_2 d)^2} \cdot (20/\log_2 d) \\
&\leq C \cdot d^2 \cdot 2^{(\log_2 \log_2 d)^2}. && \text{(using } d \geq 2^{20}\text{)}
\end{aligned}$$

This concludes the proof. ◀

Note that the constant factor 16 in Lemma 11 can be significantly reduced through a more careful optimization, but we have avoided this, preferring a simpler presentation.

References

1 Hannaneh Akrami, Bhaskar R. Chaudhury, Jugal Garg, Kurt Mehlhorn, and Ruta Mehta. EFX allocations: Simplifications and improvements. *CoRR*, abs/2205.07638, 2022. doi:10.48550/arXiv.2205.07638.

2 Noga Alon and Yair Caro. On three zero-sum Ramsey-type problems. *J. Graph Theory*, 17(2):177–192, 1993. doi:10.1002/jgt.3190170207.

3 Noga Alon and Moshe Dubiner. Zero-sum sets of prescribed size. *Combinatorics, Paul Erdős is eighty*, 1:33–50, 1993.

4 Noga Alon and Michael Krivelevich. Divisible subdivisions. *J. Graph Theory*, 98(4):623–629, 2021. doi:10.1002/jgt.22716.

5 Noga Alon and Nathan Linial. Cycles of length 0 modulo k in directed graphs. *J. Comb. Theory, Ser. B*, 47(1):114–119, 1989. doi:10.1016/0095-8956(89)90071-3.

6 Ben Berger, Avi Cohen, Michal Feldman, and Amos Fiat. (Almost Full) EFX exists for four agents (and Beyond). *CoRR*, abs/2102.10654, 2021. doi:10.48550/arXiv.2102.10654.

7 Arie Bialostocki. *Zero Sum Trees: A Survey of Results and Open Problems*, pages 19–29. Springer Netherlands, Dordrecht, 1993. doi:10.1007/978-94-011-2080-7_2.

8 Steven J. Brams and Alan D. Taylor. *Fair division – from cake-cutting to dispute resolution.* Cambridge University Press, 1996.

9 Ioannis Caragiannis, Nick Gravin, and Xin Huang. Envy-freeness up to any item with high Nash welfare: The virtue of donating items. In *EC'19: The 19th ACM Conf. on Economics and Computation*, pages 527–545. ACM, 2019. doi:10.1145/3328526.3329574.

10 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Trans. Econ. Comput.*, 7(3), September 2019. doi:10.1145/3355902.

11 Yair Caro. Zero-sum problems – A survey. *Discret. Math.*, 152(1-3):93–113, 1996. doi:10.1016/0012-365X(94)00308-6.

12 Bhaskar R. Chaudhury, Jugal Garg, Kurt Mehlhorn, Ruta Mehta, and Pranabendu Misra. Improving EFX guarantees through rainbow cycle number. In *EC '21: The 22nd ACM Conf. on Economics and Computation*, pages 310–311. ACM, 2021. doi:10.1145/3465456.3467605.

13 David Conlon and Asaf Ferber. Lower bounds for multicolor Ramsey numbers. *Advances in Mathematics*, 378:107528, 2021. doi:10.1016/j.aim.2020.107528.

14 Paul Erdős, Abraham Ginzburg, and Abraham Ziv. A theorem in additive number theory. Bull. Res. Council Israel 10F, 41-43, 1961.

15 Zoltán Füredi and Daniel J. Kleitman. On zero-trees. *J. Graph Theory*, 16(2):107–120, 1992. doi:10.1002/jgt.3190160202.

16 Edmund Landau. Über die Maximalordnung der Permutationen gegebenen Grades. *Archiv der Math. und Phys*, 3:92–103, 1903.

17 Tamás Mészáros and Raphael Steiner. Zero sum cycles in complete digraphs. *Eur. J. Comb.*, 98:103399, 2021. doi:10.1016/j.ejc.2021.103399.

18 Jean-Louis Nicolas. On Landau's function $g(n)$. In *The Mathematics of Paul Erdős I*, pages 228–240. Springer, 1997.

19 Ariel D. Procaccia. Technical perspective: An answer to fair division's most enigmatic question. *Commun. ACM*, 63(4):118, 2020. doi:10.1145/3382131.

20 Will Sawin. An improved lower bound for multicolor Ramsey numbers and a problem of Erdős. *J. Comb. Theory, Ser. A*, 188:105579, 2022. doi:10.1016/j.jcta.2021.105579.

21 Alexander Schrijver and Paul D. Seymour. A simpler proof and a generalization of the zero-trees theorem. *J. Comb. Theory, Ser. A*, 58(2):301–305, 1991. doi:10.1016/0097-3165(91)90063-M.

22 Hugo Steinhaus. The problem of fair division. *Econometrica: Journal of the Econometric Society*, 16:101–104, 1948.

23 Yuval Wigderson. An improved lower bound on multicolor Ramsey numbers. *Proc. Amer. Math. Soc.*, 149(6):2371–2374, 2021. doi:10.1090/proc/15447.

Improved Lower Bound, and Proof Barrier, for Constant Depth Algebraic Circuits

C. S. Bhargav
Indian Institute of Technology, Kanpur, India

Sagnik Dutta
Chennai Mathematical Institute, Chennai, India

Nitin Saxena
Indian Institute of Technology, Kanpur, India

Abstract

We show that any product-depth Δ algebraic circuit for the Iterated Matrix Multiplication Polynomial $\mathrm{IMM}_{n,d}$ (when $d = O(\log n/\log\log n)$) must be of size at least $n^{\Omega\left(d^{1/(\varphi^2)^\Delta}\right)}$ where $\varphi = 1.618\ldots$ is the golden ratio. This improves the recent breakthrough result of Limaye, Srinivasan and Tavenas (FOCS'21) who showed a super polynomial lower bound of the form $n^{\Omega\left(d^{1/4^\Delta}\right)}$ for constant-depth circuits.

One crucial idea of the (LST21) result was to use set-multilinear polynomials where each of the sets in the underlying partition of the variables could be of different sizes. By picking the set sizes more carefully (depending on the depth we are working with), we first show that any product-depth Δ *set-multilinear* circuit for $\mathrm{IMM}_{n,d}$ (when $d = O(\log n)$) needs size at least $n^{\Omega\left(d^{1/\varphi^\Delta}\right)}$. This improves the $n^{\Omega\left(d^{1/2^\Delta}\right)}$ lower bound of (LST21). We then use their Hardness Escalation technique to lift this to general circuits.

We also show that our lower bound cannot be improved significantly using these same techniques. For the *specific* two set sizes used in (LST21), they showed that their lower bound cannot be improved. We show that for any $d^{o(1)}$ set sizes (out of maximum possible d), the scope for improving our lower bound is minuscule: there exists a set-multilinear circuit that has product-depth Δ and size almost matching our lower bound such that the value of the measure used to prove the lower bound is maximum for this circuit. This results in a barrier to further improvement using the same measure.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases polynomials, lower bounds, algebraic circuits, proof barrier, fibonacci numbers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.18

Related Version Detailed proofs can be found in the full version of the paper.
Full Version: `https://www.cse.iitk.ac.in/users/nitin/papers/lst-limit.pdf`

Funding *Nitin Saxena*: N.S. thanks the funding support from DST-SERB (CRG/2020/000045) and N. Rama Rao Chair.

Acknowledgements We would like to thank the anonymous reviewers of MFCS 2022 for spotting errors and providing helpful suggestions that improved the presentation.

1 Introduction

An *Arithmetic Circuit* is a natural model to compute multivariate polynomials over a field $\mathbb{F}$. It is a layered *directed acyclic graph* with leaves labelled by variables $x_1, \ldots, x_n$ or elements from $\mathbb{F}$. The internal nodes are alternating layers of either addition $(+)$ or multiplication $(\times)$ gates. The circuit computes a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ in the natural way: the $+$ gates compute arbitrary $\mathbb{F}$-linear combination of their inputs and the $\times$ gates compute the product.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 18; pp. 18:1–18:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Some associated *complexity measures* are of particular importance. The *size* of the circuit is the total number of nodes (edges) in the graph. The *depth* of the circuit is the number of layers in the circuit. By *product-depth*, we mean the number of layers of multiplication gates (depth is roughly twice the product-depth). *Arithmetic Formulas* are a subclass of circuits whose underlying graph is a *tree*. For general survey of the field of Algebraic Complexity Theory, see [3, 30, 21].

Valiant [34], in a very influential work defined the classes VP and VNP which can be considered the arithmetic analogues of P and NP. Much like in the Boolean world, the question of whether VP and VNP are the same is one of the central open problems of algebraic complexity theory. Though the best known lower bounds for general arithmetic circuits [2] ($\Omega(n \log n)$) and formulas [10] ($\Omega(n^2)$) fall far short of the super polynomial lower bounds that we hope to prove, there have been many super polynomial lower bounds known for various restricted classes [22, 23, 24]. See [4, 26] for excellent surveys of lower bounds.

One of the most interesting restrictions is that of bounding the *depth* of circuits and formulas. When the depth is a constant, circuits and formulas are equivalent up to polynomial blow up in their size and hence we use them interchangeably in this paper. Unlike the Boolean world though, a very curious phenomenon of *depth reduction* occurs in arithmetic circuits [35, 1, 16, 31, 8] which essentially says that depth 3 and depth 4 circuits are almost as powerful as general ones. Formally, any degree d polynomial f that has a size s circuit can also be computed by a depth 4 *homogeneous* circuit or a depth 3 (possibly non homogeneous) circuit of size $s^{O(\sqrt{d})}$. Hence proving an $n^{\omega(\sqrt{d})}$ lower bound on these special circuits is enough to separate VP from VNP. The extreme importance of bounded depth circuits has led to a large body of work proving lower bounds for these models and their variants [28, 29, 25, 11, 7, 13, 17, 6, 18, 14, 12, 15, 9].

The LST breakthrough. Recently in a remarkable work, Limaye, Srinivasan and Tavenas [20] proved the first superpolynomial lower bound for general constant depth circuits. More precisely, they showed that the Iterated Matrix Multiplication polynomial $\mathrm{IMM}_{n,d}$ (where $d = o(\log n)$) has no product-depth Δ circuits of size $n^{d^{\exp(-O(\Delta))}}$. The polynomial $\mathrm{IMM}_{n,d}$ is defined on $N = dn^2$ variables. The variables are partitioned into d sets $X_1, \ldots, X_d$ of n^2 variables each (viewed as $n \times n$ matrices). The polynomial is defined as the $(1,1)$-th entry of the matrix product $X_1 X_2 \cdots X_d$. All monomials of the polynomial are of the same degree and so $\mathrm{IMM}_{n,d}$ is *homogeneous.* As the the individual degree of any variable is at most 1, it is also *multilinear.* Moreover, every monomial has exactly one variable from each of the sets $X_1, \ldots, X_d$. Hence the polynomial is also *set-multilinear.* For any $\Delta \leq \log d$, $\mathrm{IMM}_{n,d}$ has a set-multilinear circuit of product-depth Δ and size $n^{O(d^{1/\Delta})}$. There are no significantly better upper bounds known even if we allow general circuits. It makes sense to conjecture that this upper bound is tight (see [5] for limitations to improvement in special cases).

The lower bound of [20] proceeds by first transforming size s, product-depth Δ, general algebraic circuits computing a set-multilinear polynomial of degree d to *set-multilinear* algebraic circuits of product-depth 2Δ and size $\mathrm{poly}(s)d^{O(d)}$ (which is not huge if d is small). Hence lower bounds on bounded depth set-multilinear circuits translate to bounded depth general circuit lower bounds albeit with some loss. Finally, considering set-multilinear circuits with variables partitioned into sets of *different sizes* and crucially using this discrepancy of set sizes helps in obtaining strong set-multilinear lower bounds.

Our Results. In this work, we improve the lower bound for IMM against constant depth circuits. We also exhibit barriers to improving the bound further using these techniques, which is of importance as this is the only known approach to achieve super polynomial lower bounds for general low depth circuits.

For the rest of this paper, let $\mu(\Delta) = 1/(F(\Delta) - 1)$ where $F(n) = \Theta(\varphi^n)$ is the n-th Fibonacci number (starting with $F(0) = 1$, $F(1) = 2$) and $\varphi = (1 + \sqrt{5})/2 = 1.618\ldots$ is the golden ratio.

▶ **Theorem 1** (General circuit lower bound)**.** *Fix a field* $\mathbb{F}$ *of characteristic* 0 *or characteristic* $> d$*. Let* N, d, Δ *be such that* $d = o(\log N/\log\log N)$*. Then, any product-depth* Δ *circuit computing* $\mathrm{IMM}_{n,d}$ *on* $N = dn^2$ *variables must have size at least* $N^{\Omega\left(d^{\mu(2\Delta)}/\Delta\right)}$*.*

▶ Remark. Theorem 1 improves on the lower bound of $N^{\Omega\left(d^{1/(2^{2\Delta}-1)}/\Delta\right)}$ of [20] since $F(2\Delta) = \Theta(\varphi^{2\Delta}) \ll 2^{2\Delta}$.

To prove Theorem 1, we use the hardness escalation given by (Lemma 6) which allows for a way to convert general circuits to set-multilinear ones without too much size blow up, provided the degree is small. The actual lower bound is proved on set-multilinear circuits.

▶ **Theorem 2** (Set-multilinear circuit lower bound)**.** *Let* $d \leq (\log n)/4$*. Any product-depth* Δ *set-multilinear circuit computing* $\mathrm{IMM}_{n,d}$ *must have size at least* $n^{\Omega\left(d^{\mu(\Delta)}/\Delta\right)}$*.*

▶ Remark. This is an improvement over the $n^{\Omega\left(d^{1/(2^{\Delta}-1)}/\Delta\right)}$ bound of [20, Lemma 15]. Also, the result holds over any field $\mathbb{F}$. The restriction on the characteristic in Theorem 1 comes from the conversion to set-multilinear circuits. The difference between $\mu(2\Delta)$ in Theorem 1 and $\mu(\Delta)$ in Theorem 2 is also due to the doubling of product-depth during this conversion.

In a recent work [32], Limaye, Srinivasan and Tavenas proved a product-depth Δ *set-multilinear* formula lower bound of $(\log n)^{\Omega(\Delta d^{1/\Delta})}$ for $\mathrm{IMM}_{n,d}$. There is no restriction of degree, but in the small degree regime, the bound is much weaker than [20] and cannot be used for escalation. Improving on it, Kush and Saraf [19] showed a lower bound of $n^{\Omega(n^{1/\Delta}/\Delta)}$ for the size of product-depth Δ set-multilinear formulas computing an n^2-variate, degree n polynomial in VNP from the family of Nisan-Wigderson design-based polynomials. Unlike both [32] and [19], we are interested in the low degree regime where set-multilinear lower bounds can be lifted, and our bounds will be for IMM (a polynomial in VP), making these works incomparable to ours. We now prove Theorem 1 à la [20, Corollary 4]:

Proof of Theorem 1. From Lemma 6 and Theorem 2, for a circuit of product-depth Δ and size s computing $\mathrm{IMM}_{n,d}$ we get that $d^{O(d)}\mathrm{poly}(s) \geq N^{\Omega\left(d^{\mu(2\Delta)}/2\Delta\right)}$. Since $d = O(\log N/\log\log N)$, it follows that $d^{O(d)} = N^{O(1)}$. Therefore, $\mathrm{poly}(s) \geq N^{\Omega\left(d^{\mu(2\Delta)}/2\Delta\right)}/d^{O(d)}$ $\geq N^{\Omega\left(d^{\mu(2\Delta)}/4\Delta\right)}$ implying the required lower bound on s and thus, also Theorem 1. ◀

▶ Remark. Theorem 1 also holds when $d = o(\log N)$ and $\Delta \leq 1/4 \log_\phi \log d$. This is because the above bound on Δ implies that $d^{\mu(2\Delta)}/\Delta \geq d^{\Omega(1/\varphi^{2\Delta})}/\Delta \geq d^{\Omega(1/\sqrt{\log d})}/\log\log d \geq \log d$. Using this inequality together with the assumption $d = o(\log N)$, we get $d^{O(d)} = 2^{O(d\log d)} \leq 2^{o(\log N \cdot d^{\mu(2\Delta)}/\Delta)} = N^{o(d^{\mu(2\Delta)}/\Delta)}$ whence we can proceed similarly to the proof of Theorem 1.

The hard polynomial for which we prove set-multilinear lower bound is actually a word polynomial (Definition 4) which is a set-multilinear restriction of IMM (Lemma 5). Hence the lower bound gets translated to $\mathrm{IMM}_{n,d}$. These word polynomials are set-multilinear with respect to $(X_1, \ldots, X_d)$ where each of the X_is could potentially be of different sizes.

For the two specific set sizes considered in [20], they also exhibit polynomials that match their lower bound. It still leaves open the question whether we can improve the lower bound if we choose some other set sizes. In Theorem 2, by choosing two set sizes that are distinctly different from the ones in [20], we show that it is indeed possible to improve their lower bound. It might then seem plausible, that using many more set sizes could improve the lower bound further. We show that this is false for most cases. Suppose there are γ different set sizes among the X_is. We show that there are set-multilinear polynomials which can be computed by product-depth Δ circuits having size roughly comparable to the size lower bound of Theorem 2, provided γ is not too large. Formally,

▶ **Theorem 3** (Barrier). *Let $s_1, \dots, s_\gamma$ be positive integers. Fix sets $X_1, \dots, X_d$ where for all i, $|X_i| \in \{s_1, \dots, s_\gamma\}$. For any fixed positive integer Δ, there exist polynomials P_Δ and Q_Δ that are set-multilinear with respect to $X_1, \dots, X_d$ such that P_Δ can be computed by product-depth Δ circuits of size $n^{O\left(\Delta\gamma d^{\mu(\Delta)}\right)}$ and Q_Δ can be computed by product-depth Δ circuits of size $n^{O\left(\Delta d^{\mu(\Delta-1)}+\gamma\right)}$. Moreover, both P_Δ and Q_Δ maximise the measure used to prove lower bounds.*

▶ Remark. The two different polynomials with slightly different sizes will imply barriers to improving the lower bound in different regimes of γ. Suppose Δ is small (say $\Delta = O(1)$). When $\gamma = O(1)$, the size of P_Δ matches our lower bound, essentially implying its tightness. When γ is $d^{o(1)}$, the size of Q_Δ is only slightly larger than the lower bound (note $\mu(\Delta-1)$ vs $\mu(\Delta)$). Hence even using multiple set sizes, the scope for improvement is tiny.

In an almost parallel work [33], Limaye, Srinivasan and Tavenas show similar barrier results. They simplify the proof framework of [20] and give a characterization of the lower bounds that can be proved via this technique using a combinatorial property which they term *Tree Bias*. Their result works for any d set sizes but the upper bound they obtain is weaker. More precisely, for any partition $(X_1, \dots, X_d)$ of the input variables they exhibit a set-multilinear polynomial that can be computed by product-depth Δ set-multilinear circuits of size $n^{d^{1/\Delta^{\Omega(\log \Delta)}}}$ while simultaneously maximising the measure. These barrier results (Theorem 3 and results of [33]) suggest that new measures might be necessary to improve the lower bounds.

2 Preliminaries

For any positive integer n, we denote by $F(n)$ the n-th Fibonacci number with $F(0) = 1$, $F(1) = 2$ and $F(n) = F(n-1) + F(n-2)$. The nearest integer to any real number r is denoted by $\lfloor r \rceil$. We follow the notation of [20] as much as possible for better readability.

We consider words that are tuples $(w_1, \dots, w_d)$ of length d where $2^{|w_i|}$ are integers. These words define the actual set sizes of the set-multilinear polynomials we will be working with. Given a word w, let $\overline{X}(w)$ denote the tuple of sets of variables $(X_1(w), \dots, X_d(w))$ where the size of each $X_i(w)$ is $2^{|w_i|}$. We denote the space of set-multilinear polynomials over $\overline{X}(w)$ by $\mathbb{F}_{sm}[\overline{X}(w)]$.

For a word w and any subset $S \subseteq [d]$, the sum of elements of w indexed by S is denoted by $w_S = \sum_{i \in S} w_i$. If for all $t \le d$, $|w_{[t]}| \le b$, then we call w b-unbiased. Denote by $w_{|S}$ the sub-word indexed by S. The positive and negative indices of w are denoted $\mathcal{P}_w = \{i \mid w_i \ge 0\}$ and $\mathcal{N}_w = \{i \mid w_i < 0\}$ respectively with the corresponding collections $\{X_i(w)\}_{i \in \mathcal{P}_w}$ and $\{X_i(w)\}_{i \in \mathcal{N}_w}$ being the positive and negative variable sets. We denote by $\mathcal{M}_w^{\mathcal{P}}$ (resp. $\mathcal{M}_w^{\mathcal{N}}$) the set of all set-multilinear monomials over the positive (resp. negative) variable sets.

The *partial derivative matrix* $\mathcal{M}_w(f)$ has rows indexed by $\mathcal{M}_w^{\mathcal{P}}$ and columns by $\mathcal{M}_w^{\mathcal{N}}$. The entry corresponding to row $m_+ \in \mathcal{M}_w^{\mathcal{P}}$ and $m_- \in \mathcal{M}_w^{\mathcal{N}}$ is the coefficient of the monomial $m_+ m_-$ in f. The complexity measure we use is the *relative rank*, same as [20]:

$$\mathrm{relrk}_w(f) := \frac{\mathrm{rank}(\mathcal{M}_w(f))}{\sqrt{|\mathcal{M}_w^{\mathcal{P}}| \cdot |\mathcal{M}_w^{\mathcal{N}}|}} = \frac{\mathrm{rank}(\mathcal{M}_w(f))}{2^{\frac{1}{2}\sum_{i \in [d]} |w_i|}} \le 1 \; .$$

The following properties of relrk_w will be useful (see [20] for the proofs).

1. (Imbalance) For any $f \in \mathbb{F}_{sm}[\overline{X}(w)]$, $\mathrm{relrk}_w(f) \leq 2^{-|w_{[d]}|/2}$.
2. (Sub-additivity) For any $f, g \in \mathbb{F}_{sm}[\overline{X}(w)]$, $\mathrm{relrk}_w(f+g) \leq \mathrm{relrk}_w(f) + \mathrm{relrk}_w(g)$.
3. (Multiplicativity) Suppose $f = f_1 f_2 \cdots f_t$ where $f_i \in \mathbb{F}_{sm}[\overline{X}(w_{|S_i})]$ and $(S_1, \ldots, S_t)$ is a partition of $[d]$. Then, $\mathrm{relrk}_w(f) = \mathrm{relrk}_w(f_1 f_2 \cdots f_t) = \prod_{i\in[t]} \mathrm{relrk}_{w_{|S_i}}(f_i)$.

We now define the hard polynomials we prove lower bounds for. For any monomial $m \in \mathbb{F}_{sm}[\overline{X}(w)]$, let $m_+ \in \mathcal{M}_w^{\mathcal{P}}$ and $m_- \in \mathcal{M}_w^{\mathcal{N}}$ be its "positive" and "negative" parts. As $|X_i| = 2^{|w_i|}$, the variables of X_i can be indexed using boolean strings of length $|w_i|$. This gives a way to associate a boolean string with any monomial. Let $\sigma(m_+)$ and $\sigma(m_-)$ be the strings associated with m_+ and m_- respectively. We write $\sigma(m_+) \sim \sigma(m_-)$ if one is a prefix of the other.

▶ **Definition 4** ([20, Word polynomials]). *Let w be any word. The polynomial P_w is defined as the sum of all monomials $m \in \mathbb{F}_{sm}[\overline{X}(w)]$ such that $\sigma(m_+) \sim \sigma(m_-)$.*

The matrices $M_w(P_w)$ have full rank (equal to either the number of rows or columns, whichever is smaller) and hence $\mathrm{relrk}_w(P_w) = 2^{-|w_{[d]}|/2}$. We also note (without proof) that these polynomials can be obtained as *set-multilinear* restrictions of $\mathrm{IMM}_{n,d}$.

▶ **Lemma 5** ([20, Lemma 8]). *Let w be any b-unbiased word. If there is a set-multilinear circuit computing $\mathrm{IMM}_{2^b,d}$ of size s and product-depth Δ, then there is also a set-multilinear circuit of size s and product-depth Δ computing the polynomial $P_w \in \mathbb{F}_{sm}[\overline{X}(w)]$. Moreover, $\mathrm{relrk}_w(P_w) \geq 2^{-b/2}$.*

We also state the set-multilinearization lemma alluded to before.

▶ **Lemma 6** ([20, Proposition 9]). *Let s, N, d, Δ be growing parameters with $s \geq Nd$. If C is a circuit of size at most s and product-depth at most Δ computing a set-multilinear polynomial P over the sets of variables $(X_1, \ldots, X_d)$ (with $|X_i| \leq N$), then there is a* set-multilinear circuit *$\tilde{C}$ of size $d^{O(d)}\mathrm{poly}(s)$ and product-depth at most 2Δ computing P.*

3 Proof outline

From the discussion in Section 1 and Lemmas 5 and 6, in order to prove general circuit lower bounds, it suffices to prove that there is a high rank word polynomial that needs large set-multilinear formulas. For a word (and hence set sizes) of our choice, we show that relrk_w is small for set-multilinear formulas of a certain size.

Let k be an integer close to $\log_2 n$. In [20], the authors choose the positive entries of the word w to be an integer close to $k/\sqrt{2}$ and the negative entries to be $-k$. Evidently, these entries are independent of the product-depth Δ. In this paper, we take the positive entries to be $(1 - p/q)k$ and the negative entries to be $-k$ where p and q are suitable integers dependent on Δ. This *depth-dependent* construction of the word enables us to improve the lower bound. We demonstrate the high level proof strategy of the lower bound for the case of product-depth 3.

Proof overview of Theorem 2 for $\Delta = 3$. Define $G(i) = 1/\mu(i) = F(i) - 1$ for all i and let $\lambda = \lfloor d^{1/G(3)} \rfloor$. Consider a set-multilinear forumula C of product-depth 3 and let v be a gate in it. Suppose that the subformula $C^{(v)}$ rooted at v has product-depth $\delta \leq 3$, size s and degree $\geq \lambda^{G(\delta)}/2$. We will prove that $\mathrm{relrk}_w(C^{(v)}) \leq s2^{-k\lambda/48}$ by induction on δ. This will give us the desired upper bound of the form $s2^{-k\lambda/48} = sn^{-\Omega(d^{\mu(3)})}$ on the relative rank of the whole formula when v is taken to be the output gate. Write $C^{(v)} = C_1 + \cdots + C_t$ where each C_i is a subformula of size s_i rooted at a product gate. Because of the subadditivity of relrk_w, it suffices to show that $\mathrm{relrk}_w(C_i) \leq s_i 2^{-k\lambda/48}$ for all i.

Base case. If $\delta = 1$, then C_i is a product of linear forms. Thus, it has rank 1 and hence low relative rank.

Induction step. $\delta \in \{2, 3\}$. Write $C_i = C_{i,1} \dots C_{i,t_i}$ where each $C_{i,j}$ is a subformula of product-depth $\delta - 1$. If any $C_{i,j}$ has degree $\geq \lambda^{G(\delta-1)}/2$, then by induction hypothesis, the relative rank of $C_{i,j}$ and hence C_i will have the desired upper bound and we are done.

Otherwise each $C_{i,j}$ has degree $D_{ij} < \lambda^{G(\delta-1)}/2$. As the formula is set-multilinear, there is a collection of variable-sets $(X_l)_{l\in S_j}$ with respect to which $C_{i,j}$ is set-multilinear. For $j \in [t_i]$, let a_{ij} be the number of positive indices in S_j i.e. the number of positive sets in the collection $(X_l)_{l\in S_j}$. Then the number of negative indices is $(D_{ij} - a_{ij})$.

We consider two cases: if $a_{ij} \leq D_{ij}/3$, then $w_{S_j} \leq (D_{ij}/3) \cdot (1 - p/q)k + (2D_{ij}/3) \cdot (-k) \leq -D_{ij}k/3$. Otherwise $a_{ij} > D_{ij}/3$ and if we can prove that $|w_{S_j}| \geq a_{ij}k/(4\lambda^{G(\delta)-1})$, then in both of the above cases, we would have $|w_{S_j}| \geq D_{ij}k/(12\lambda^{G(\delta)-1})$. By the multiplicativity and imbalance property of relrk_w, it would follow that $\mathrm{relrk}_w(C_i) \leq 2^{\sum_{j=1}^{t_i} -\frac{1}{2}|w_{S_j}|} \leq 2^{-k\lambda/48}$ and we would be done. Thus, we now only have to show that $|w_{S_j}| \geq a_{ij}k/(4\lambda^{G(\delta)-1})$. We have

$$|w_{S_j}| = |a_{ij}(1 - p/q) - (D_{ij} - a_{ij})|\, k \ .$$

Notice that $|w_{S_j}|/k$ is the distance of $a_{ij}p/q$ from some integer, so it must be at least the minimum of $\{a_{ij}p/q\}$ and $1 - \{a_{ij}p/q\}$ where $\{.\}$ denotes the fractional part. The number $a_{ij}p/q$ being rational, has a fractional part $\zeta = (a_{ij}p \bmod q)/q$ and hence it comes down to solving the following system of inequalities:

$$\min(\zeta,\ 1 - \zeta) \geq a_{ij}/(4\lambda^{G(\delta)-1}) \text{ for } \delta \in \{2,3\} \text{ when } a_{ij} \leq D_{ij} < \lambda^{G(\delta-1)}/2 \ .$$

Assign $p = \lambda$, $q = \lambda^2 + 1$. The $\delta = 2$ case is clearly satisfied as $(a_{ij}\lambda \mod (\lambda^2 + 1)) = a_{ij}\lambda$ when $0 \leq a_{ij} \leq \lambda/2$.

Consider the case of $\delta = 3$ and $a_{ij} < \lambda^2/2$. Write $a_{ij} = y_1\lambda + y_0$ for integers $y_1 = \lfloor a_{ij}/\lambda \rfloor < \lambda/2$ and $y_0 \leq \lambda - 1$. Thus, $a_{ij}\lambda \equiv -y_1 + y_0\lambda \mod (\lambda^2 + 1)$. Through some case analysis, one can show that $\min\Big(|y_0\lambda - y_1|,\ \lambda^2 + 1 - |y_0\lambda - y_1|\Big) \geq y_1$ which immediately implies the inequality for the $\delta = 3$ case as $y_1 = \lfloor a_{ij}/\lambda \rfloor \geq a_{ij}/(2\lambda)$.

We can attempt to extend this proof technique to product-depth 4 as follows:
We would similarly want to express a_{ij} as $a_{ij} = y_2\lambda^2 + y_1\lambda + y_0$ for integers $y_2 = \lfloor a_{ij}/\lambda^2 \rfloor, y_0 \leq \lambda - 1$ and $y_1 \leq \lambda - 1$. Ideally, we would want that for some $q \approx \lambda^4$,

$$p\lambda^2 \equiv 1 \bmod q,\ p\lambda \equiv -\lambda^2 \bmod q \text{ and } p \equiv \lambda^3 \bmod q$$

so that $a_{ij}p \equiv y_2 - y_1\lambda^2 + y_0\lambda^3 \bmod q$ and then we can carry out a similar analysis as in the $\Delta = 3$ case. But this is not possible since multiplying the second congruence equation by λ gives $p\lambda^2 \equiv -\lambda^3 \bmod q$, which contradicts the first congruence equation. So we decide to express a_{ij} as $a_{ij} = y_2b_2 + y_1b_1 + y_0b_0$ where b_2, b_1, b_0 are close to $\lambda^2, \lambda, 1$ respectively, instead of being precisely equal to these powers of λ. Then we choose $c_2 \approx 1, c_1 \approx -\lambda^2, c_0 \approx \lambda^3$ and we assign values to p and q such that

$$pb_2 \equiv c_2 \bmod q,\ pb_1 \equiv c_1 \bmod q \text{ and } pb_0 \equiv c_0 \bmod q.$$

It is easy to verify that all these conditions are satisfied if we define

$$b_0 = 1, b_1 = \lambda, b_2 = b_1(\lambda - 1) + b_0; \qquad c_2 = 1, c_1 = -\lambda^2, c_0 = c_2 - c_1(\lambda - 1);$$
$$p = c_0 \text{ and } q = pb_1 - c_1.$$

This inspired our construction of the sequences $\{b_m\}$ and $\{c_m\}$ for general product-depth Δ.

Proof overview of Theorem 3. As mentioned before, we would like to find a family of polynomials for which our lower bound is tight. All the same, we want to maintain high relative rank of these polynomials. If we are able to achieve this and find the appropriate small sized formulas for the said polynomials, we will have that the lower bound cannot be improved using the relative rank measure.

The polynomial P we define will be a close variant of the word polynomials from before. This will ensure that the partial derivative matrix has the maximum possible rank for a matrix of its dimension. From the Imbalance property, the relative rank we obtain is $2^{-|w_{[d]}|/2}$ where we have ensured that $w_{[d]}$ is small. We want to construct the formula F for P such that it has a nice inductive structure. That is, we want the polynomials computed by the subformulas of F to also have high relative rank. This will help us construct a formula from its sub formulas while maintaining high relative rank.

Suppose a subformula F' of F is set multilinear with respect to a subtuple $\mathcal{T}$ of the sets of variables $\overline{X}(w)$. Let these sets in $\mathcal{T}$ be indexed by a set $S_{\mathcal{T}} \subseteq [d]$. As we would like high relative rank of F', the Imbalance property again suggests that $|w_{S_{\mathcal{T}}}|$ be small. And we desire this of every subformula, their subformulas, and so on. So roughly, we want a way to partition our intial index set $[d]$ into some number of index sets $S_1, \dots, S_r$ such that each $|w_{S_i}|$ is small. Suppose we are then able to create subformulas of rank $2^{-|w_{S_i}|/2}$. It turns out that we will have to add roughly $2^{\sum_i |w_{S_i}|}$ many of them to get a polynomial of high relative rank. So to control the size of the formula, we would like $\sum_i |w_{S_i}|$ to be small as well.

In their Depth Hierarchy section, [20] use Dirichlet's approximation principle [27] to pick these nice index sets $\{S_i\}$. Their procedure only works for the particular two variable-set sizes they choose. We extend this to any two set sizes in Claim 13. Interestingly, we do not use Dirichlet to pick the index sets but rather to obtain a lower bound on the size of the sets that we do eventually pick. We think of picking sets as an investment process: when we pick a set S, we buy the $|S|$ elements in it for a cost of $|w_S|$. Hence the cost per element is $|w_S|/|S|$. At each product-depth, we are only allowed to pick sets of size under a certain threshold and we pick the ones with the lowest cost per element. It turns out that this lowest cost decreases exponentially as the depth increases and helps us build a small formula. The decrease is captured by the Fibonacci numbers and is the reason why they emerge in our lower bound and upper bound.

Making these ideas precise requires extensive notation and we postpone further discussion to Section 5.

4 The lower bound: Proof of Theorem 2

In this section we prove the set-multilinear lower bound of Theorem 2.

Fix the product-depth Δ for which we want to prove the lower bound. Define $G(i) := F(i) - 1$ for all i and $\lambda = \lfloor d^{1/G(\Delta)} \rfloor$. We can assume that $\lambda \geq 3$ because otherwise $d^{\mu(\Delta)} < 3$ and in that case, the lower bound is trivial. The lower bound we aim to prove is $n^{\Omega(d^{1/G(\Delta)})}$. We first define the sequences $\{b_m\}$ and $\{c_m\}$ mentioned in the proof overview:

Let $r_m := \lambda^{G(m+1)-G(m)} - 1$ for $0 \leq m \leq \Delta - 2$.

Define

$$b_0 := 1, \quad b_1 := \lambda \text{ and } b_m := b_{m-2} + r_{m-1} b_{m-1} \text{ for } 2 \leq m \leq \Delta - 2\ .$$

Define

$$c_{\Delta-2} := (-1)^{\Delta-2}, \qquad c_{\Delta-3} := (-1)^{\Delta-3} \lambda^{G(\Delta-1)-G(\Delta-2)} \text{ and}$$
$$c_m := (-1)^m (|c_{m+2}| + r_{m+1}|c_{m+1}|) \text{ for } \Delta - 4 \geq m \geq 0\ .$$

Note that the sign parity of c_m is $(-1)^m$ for all m.
Thus, $c_{m-2} = (-1)^{m-2}(|c_m| + r_{m-1}|c_{m-1}|) = c_m - r_{m-1}c_{m-1}$ which implies

$$c_m = c_{m-2} + r_{m-1}c_{m-1} \text{ for } 2 \leq m \leq \Delta - 2\ .$$

It can be shown (see the full version for the proof) that each b_m is close to $\lambda^{G(m)}$ and each $|c_m|$ is close to $\lambda^{G(\Delta-1)-G(m+1)}$:

$$\frac{\lambda^{G(m)}}{2} \leq b_m \leq \lambda^{G(m)} \quad \text{and} \quad \frac{\lambda^{G(\Delta-1)-G(m+1)}}{2} \leq |c_m| \leq \lambda^{G(\Delta-1)-G(m+1)} \quad \text{for all } m. \tag{1}$$

Define

$$p := c_0 \text{ and } q := pb_1 - c_1 = c_0(r_0+1) - c_1\ .$$

By defining the integers p and q this way, we have ensured that $pb_0 \equiv c_0 \bmod q$ and $pb_1 \equiv c_1 \bmod q$. Hence from the relations $b_m = b_{m-2}+r_{m-1}b_{m-1}$ and $c_m = c_{m-2}+r_{m-1}c_{m-1}$, it inductively follows that

$$pb_m \equiv c_m \bmod q \qquad \text{for } 0 \leq m \leq \Delta - 2\ . \tag{2}$$

Constructing the word. Define $\alpha = 1 - p/q$. As $\dfrac{p}{q} \leq \dfrac{c_0}{c_0(r_0+1)} = 1/\lambda$, we have $\alpha \geq 1/2$. Since $q = c_0\lambda - c_1$, it implies that

$$q \leq |c_0|\lambda + |c_1| \leq 2\lambda^{G(\Delta-1)} \leq d < \lfloor \log_2 n \rfloor / 2$$

where the second inequality follows from the upper bound on each $|c_m|$ in (1). Therefore, there exists a multiple of q in the interval $\left[\frac{\lfloor \log_2 n \rfloor}{2}, \lfloor \log_2 n \rfloor\right]$. Let k be this multiple of q. Then αk is an integer. We can construct a word w over the alphabet $\{\alpha k, -k\}$ such that w is k-unbiased. This can be done using induction: if $|w_{[i]}| \leq 0$, set $w_{i+1} = \alpha k$, otherwise set $w_{i+1} = -k$.

With these definitions in place, we are ready to prove Theorem 2. Assume the following lemma:

▶ **Lemma 7.** *Let $\delta \leq \Delta$ be an integer and α, k be as defined above. Let w be any word of length d over the alphabet $\{\alpha k, -k\}$. Then any set-multilinear formula C of product-depth δ, degree $D \geq \lambda^{G(\delta)}/8$ and size at most s satisfies*

$$\mathrm{relrk}_w(C) \leq s2^{-k\lambda/256}.$$

Proof of Theorem 2. By lemma 5, there exists a set-multilinear projection P_w of $\mathrm{IMM}_{2^k,d}$ such that $\mathrm{relrk}_w(P_w) \geq 2^{-k}$. If there is a set-multilinear circuit of size s and product-depth Δ computing $\mathrm{IMM}_{n,d}$, then we can expand it to a set-multilinear formula of size at most $s^{2\Delta}$ which computes the same polynomial. Hence we will also have a set-multilinear formula of size at most $s^{2\Delta}$ computing P_w. As $d \geq \lambda^{G(\Delta)}/8$, taking the particular case of $\delta = \Delta$ in Lemma 7, we obtain $\mathrm{relrk}_w(P_w) \leq s^{2\Delta}2^{-k\lambda/256}$. This gives the desired lower bound

$$s^{2\Delta} \geq 2^{-k}2^{k\lambda/256} \geq \left(\frac{n}{4}\right)^{\frac{d^{1/G(\Delta)}}{512}}/n = n^{\Omega(d^{\mu(\Delta)})}. \qquad \blacktriangleleft$$

Proof of Lemma 7. We proceed by induction on δ. We can write $C = C_1 + \cdots + C_t$ where each C_i is a subformula of size s_i rooted at a product gate. Because of the subadditivity of relrk_w, it suffices to show that

$$\mathrm{relrk}_w(C_i) \leq s_i 2^{-k\lambda/256} \qquad \text{for all } i.$$

Base case. C has product-depth $\delta = 1$ and degree $D \geq \lambda/8$.
Then C_i is a product of linear forms. If L is linear form on some variable set $X(w_j)$, then $\mathrm{relrk}_w(L) \leq 2^{-|w_j|/2} \leq 2^{-k/4}$. Therefore by the multiplicativity of relrk_w,

$$\mathrm{relrk}_w(C_i) \leq 2^{-kD/4} \leq 2^{-k\lambda/32}\ .$$

Induction hypothesis. Assume that the lemma is true for all product-depths $\leq \delta - 1$.

Induction step. Let C be a formula of product-depth δ and degree $D \geq \lambda^{G(\delta)}/8$.
We can write $C_i = C_{i,1} \ldots C_{i,t_i}$ where each $C_{i,j}$ is a subformula of product-depth $\delta - 1$.
If C_i has a factor, say $C_{i,1}$, of degree $\geq \lambda^{G(\delta-1)}/8$, then by induction hypothesis,

$$\mathrm{relrk}_w(C_i) \leq \mathrm{relrk}_w(C_{i,1}) \leq s_i 2^{-k\lambda/256}\ .$$

Otherwise every factor of C_i has degree $< \lambda^{G(\delta-1)}/8$. Let $C_i = C_{i,1} \ldots C_{i,t_i}$ where each $C_{i,j}$ has degree $D_{ij} < \lambda^{G(\delta-1)}/8$. If C_i is set-multilinear with respect to $(X_l)_{l\in S}$, then let $(S_1, \ldots, S_{t_i})$ be the partition of S such that each $C_{i,j}$ is set-multilinear with respect to $(X_l)_{l\in S_j}$.
For $j \in [t_i]$, let a_{ij} be the number of positive indices in S_j. We have two cases: If $a_{ij} \leq D_{ij}/2$, then

$$w_{S_j} \leq \frac{D_{ij}}{2} \cdot \alpha k + \frac{D_{ij}}{2} \cdot (-k) = -\frac{D_{ij}p}{2q}k \leq -\frac{D_{ij}k}{4\lambda}$$

where the last inequality follows from $\frac{p}{q} \geq \frac{c_0}{2c_0(r_0+1)} = \frac{1}{2\lambda}$. The other case is $a_{ij} > D_{ij}/2$. If we can prove that $|w_{S_j}| \geq a_{ij}k/(8\lambda^{G(\delta)-1})$, then in both of the above cases, we would have $|w_{S_j}| \geq D_{ij}k/(16\lambda^{G(\delta)-1})$. By the multiplicativity and imbalance property of relrk_w and the assumption $D \geq \lambda^{G(\delta)}/8$, it would follow that

$$\mathrm{relrk}_w(C_i) \leq \prod_{j=1}^{t_i} 2^{-\frac{1}{2}|w_{S_j}|} \leq 2^{-\sum_{j=1}^{t_i} D_{ij}k/(32\lambda^{G(\delta)-1})} = 2^{-Dk/(32\lambda^{G(\delta)-1})} \leq 2^{-k\lambda/256}$$

and we would be done. Thus, we now only have to show that $|w_{S_j}| \geq a_{ij}k/(8\lambda^{G(\delta)-1})$.

$$\begin{aligned} |w_{S_j}| &= |a_{ij} \cdot \alpha k + (D_{ij} - a_{ij}) \cdot (-k)| = \left|a_{ij}\frac{p}{q} - (2a_{ij} - D_{ij})\right| k && \text{as } \alpha = 1 - p/q \\ &\geq \left|\frac{a_{ij}p}{q} - \left\lfloor \frac{a_{ij}p}{q} \right\rceil\right| k && \text{where } \lfloor . \rceil \text{ denotes the nearest integer.} \end{aligned}$$

The fractional part of $\dfrac{a_{ij}p}{q}$ is $\dfrac{a_{ij}p \bmod q}{q}$. Hence in order to prove that $|w_{S_j}| \geq a_{ij}k/(8\lambda^{G(\delta)-1})$, it is enough to verify that the following inequality is satisfied:

$$\min\left(\frac{a_{ij}p \bmod q}{q}, 1-\frac{a_{ij}p \bmod q}{q}\right) \geq \frac{a_{ij}}{8\lambda^{G(\delta)-1}} \tag{3}$$

Showing that the p, q we defined satisfy the inequality (3). We will first find what we call the base $(b_0, \dots, b_{\Delta-2})$ representation of the number a_{ij}. For $0 \leq m \leq \Delta-2$, inductively define y_m to be the integer quotient when $\left(a_{ij} - \sum\limits_{m'=m+1}^{\Delta-2} b_{m'}y_{m'}\right)$ is divided by b_m. Then we can express a_{ij} as $a_{ij} = \sum\limits_{m=0}^{\Delta-2} b_m y_m$. Since $b_m \geq \lambda^{G(m)}/2$ for all m and $a_{ij} \leq D_{ij} < \lambda^{G(\delta-1)}/8$, we have the following bounds on the values of y_m:

$$y_m = 0 \text{ for } m \geq \delta-1, \tag{4}$$

$$y_{\delta-2} = \left\lfloor \frac{a_{ij}}{b_{\delta-2}} \right\rfloor < \frac{\frac{\lambda^{G(\delta-1)}}{8}}{\frac{\lambda^{G(\delta-2)}}{2}} \leq \frac{\lambda^{G(\delta-1)-G(\delta-2)}-1}{2} = \frac{r_{\delta-2}}{2}, \tag{5}$$

$$y_m \leq \left\lfloor \frac{b_{m+1}-1}{b_m} \right\rfloor = r_m \text{ for } m < \delta-2\ . \tag{6}$$

By (2), $a_{ij}p \equiv \sum\limits_{m=0}^{\Delta-2} c_m y_m \bmod q$. Therefore,

$$\min\left(\frac{a_{ij}p \bmod q}{q}, 1-\frac{a_{ij}p \bmod q}{q}\right) = \min\left(\left|\sum_{m=0}^{\Delta-2} c_m y_m\right|/q,\ 1-\left|\sum_{m=0}^{\Delta-2} c_m y_m\right|/q\right) \tag{7}$$

if $\left|\sum_{m=0}^{\Delta-2} c_m y_m\right|/q \leq 1$, which is true by the following claim (see the full version for the proof):

▷ Claim 8. If $0 \leq y_m \leq r_m$ for all m, then $\left|\sum\limits_{m=0}^{\Delta-2} c_m y_m\right| < q - c_0$.

Now let f be the highest index such that $y_f \geq 1$ (by (4), $f \leq \delta-2$) and e be the smallest index such that $y_e \geq 1$. Then $\left|\sum_{m=0}^{\Delta-2} c_m y_m\right| = \left|\sum_{m=e}^{f} c_m y_m\right|$. We need two more claims whose proofs can be found in the full version.

▷ Claim 9. Let y_m be non-negative integers such that $y_e \geq 1$. Then $\left|\sum_{m=e}^{f} c_m y_m\right| \geq \min\Big(|c_f y_f|, |c_{f-1}| - |c_f y_f|\Big)$.

▷ Claim 10. Let $\{y_m\}_{m=0}^{\delta-2}$ be a sequence of non-negative integers. Let $f \leq \delta-2$ be the highest index such that $y_f \geq 1$. If $y_{\delta-2} = \lfloor \frac{a_{ij}}{b_{\delta-2}} \rfloor \leq r_{\delta-2}/2$ and $0 \leq y_m \leq r_m$ for all $m \leq \delta-2$, then $\min\Big(|c_f y_f|, |c_{f-1}| - |c_f y_f|\Big) \geq |c_{\delta-2}a_{ij}/(2b_{\delta-2})|$.

If $\delta = 2$, then $f = 0$ by (4). Thus $q - \left|\sum_{m=e}^{f} c_m y_m\right| > c_0 r_0 - |c_0 y_0| > c_0 r_0/2 > |c_f y_f|$ where the last two inequalities follow from (5).

Otherwise $\delta > 2$. By Claim 8, $q - \left|\sum_{m=e}^{f} c_m y_m\right| > c_0$. From the definition of the sequence $\{c_m\}$, we have $c_0 \geq |c_f r_f| \geq |c_f y_f|$ when $f > 0$. But when $f = 0$, it follows that $y_{\delta-2} = 0$ implying $a_{ij} < b_{\delta-2}$. This further implies $c_0 \geq |c_{\delta-2}| \geq |c_{\delta-2} a_{ij}/b_{\delta-2}|$.

From the analysis of the two cases above and by Claims 9 and 10, we get that

$$\min\left(\left|\sum_{m=e}^{f} c_m y_m\right|,\ q - \left|\sum_{m=e}^{f} c_m y_m\right|\right)/q \geq \left|\frac{c_{\delta-2} a_{ij}}{2 b_{\delta-2} q}\right|.$$

The bounds on each b_m and $|c_m|$ given in (1) imply the following:

$$|c_{\delta-2}| \geq \lambda^{G(\Delta-1)-G(\delta-1)}/2, \quad b_{\delta-2} \leq \lambda^{G(\delta-2)}, \quad q \leq |c_0|\lambda + |c_1| \leq 2\lambda^{G(\Delta-1)}\ .$$

Hence $\min\left(\left|\sum_{m=e}^{f} c_m y_m\right|/q,\ 1 - \left|\sum_{m=e}^{f} c_m y_m\right|/q\right) \geq \frac{a_{ij}}{8\lambda^{G(\delta-1)+G(\delta-2)}} = \frac{a_{ij}}{8\lambda^{G(\delta)-1}}$ which together with (7) implies (3). ◀

5 Limitations on improving the bounds: Proof of Theorem 3

We will show here that the techniques of [20] cannot hope to prove much stronger lower bounds. We do this by constructing polynomials for which the lower bound we proved earlier is tight. We begin by showing this in the case of two different set sizes. We can normalize with respect to the bigger set size to assume that the weights are $-k$ and αk ($\alpha \in [0,1]$) without loss of generality. Clearly, $k \leq \log n$.

▶ **Lemma 11.** *Let n, d, Δ be such that $d \leq n$. For any $\alpha \in [0,1]$ let $w \in \{-k, \alpha k\}^d$ be a word. There is a polynomial $P_\Delta \in \mathbb{F}_{sm}[\overline{X}(w)]$ which is computable by a set-multilinear formula of product-depth at most Δ, size at most $n^{O(\Delta d^{\mu(\Delta)})}$ and has the maximum possible relative rank.*

▶ Remark. We can replace αk with $\lfloor \alpha k \rfloor$ and assume that the weights in w are integers. It can be shown that this will not change the arguments in any significant way (see Claim 21 in the full version).

We will need the extensive notation from [20]. We restate it here.

Notation.

- As in Section 2 and from the remark above, we assume $|X(w_i)| = 2^{|w_i|}$ and that the variables are indexed by binary strings $\{0,1\}^{|w_i|}$.
- Given any subset $S \subseteq [d]$, we denote by $S_+ = \{i \in S \mid w_i > 0\}$ the positive indices of S and similarly by S_-, the negative indices.
- We let $K = \sum_{i\in[d]} |w_i|$, $k_+ = \sum_{i\in S_+} |w_i|$ and $k_- = \sum_{i\in S_-} |w_i|$. We say S is $\mathcal{P}$-heavy if $k_+ \geq k_-$ and $\mathcal{N}$-heavy otherwise.
- Setting $I = [K]$, we partition the set $I = I_1 \cup \cdots \cup I_d$ where I_j is an interval of length $|w_j|$ that starts at $\sum_{i<j} |w_j| + 1$. Given a $T \subseteq [d]$, we let $I(T) = \bigcup_{j\in T} I_j$.
- Let $m = m_+ m_- \in \mathcal{M}_w^S$ be any monomial. The boolean string $\sigma(m_+)$ associated with the positive monomial (as defined in Section 2) can be thought of as a labelling of the elements of $I(S_+)$ in the natural way - $\sigma(m_+) : I(S_+) \to \{0,1\}$. Similarly for $\sigma(m_-)$.

Given a set S, we define a sequence of polynomials that we will later show to have small size set multilinear formulas but large rank.

Fix $J_+ \subseteq I(S_+)$ and $J_- \subseteq I(S_-)$ such that $|J_+| = |J_-| = \min\{k_+, k_-\}$. Let π be a bijection from J_+ to J_-. Such a tuple (S, J_+, J_-, π) is called valid. Fix a valid (S, J_+, J_-, π).

A string $\tau \in \{0,1\}^{|k_+ - k_-|}$ defines a map $I(S_+) \setminus J_+ \to \{0,1\}$ if S is $\mathcal{P}$-heavy and a map $I(S_-) \setminus J_- \to \{0,1\}$ if S is $\mathcal{N}$-heavy.

The polynomial $P_{(S,J_+,J_-,\pi,\tau)}$ is the sum of all monomials m such that

1. $\sigma(m_+)(j) = \sigma(m_-)(\pi(j))$ for all $j \in J_+$, and
2. $\sigma(m_+)(j) = \tau(j)$ for all $j \in I(S_+) \setminus J_+$ if S is $\mathcal{P}$-heavy or $\sigma(m_-)(j) = \tau(j)$ for all $j \in I(S_-) \setminus J_-$ if S is $\mathcal{N}$-heavy.

As observed in [20], these polynomials have maximum possible relative rank and other properties that help in building formulas for them inductively (precise statements in the full version).

To proceed, we introduce a few notions that help make the ideas in the proof overview above precise. Fix Δ as in Lemma 11. We define the *fractional cost* fc. Set $\mathrm{fc}(0) = 1$ and

$$\mathrm{fc}(\delta) \coloneqq \min_{q < d^{\mu(\Delta)}/\mathrm{fc}(\delta-1)} |q\alpha - \lfloor q\alpha \rceil|/q \qquad \text{for } 1 \leq \delta \leq \Delta - 1.$$

The quantity $|q\alpha - \lfloor q\alpha \rceil|$ is the distance to the nearest integer from $q\alpha$. For $1 \leq \delta \leq \Delta - 1$, we denote by p_δ the (least) value of q for which the above expression attains the minimum. We also denote by $n_\delta \coloneqq \lfloor p_\delta \alpha \rceil$ the nearest integer to $p_\delta \alpha$. Finally, we set $p_\Delta \coloneqq |\mathcal{P}_w|$ (total number of positive sets) and $n_\Delta \coloneqq |\mathcal{N}_w|$ (total number of negative sets).

We state (without proof) a few properties of the terms defined above and point the reader to the full version for details.

(C1) (Exponential decline) The fractional cost falls exponentially with depth i.e., $\mathrm{fc}(\delta) \leq 1/(d^{\mu(\Delta)})^{F(\delta+1)-2}$ for $1 \leq \delta \leq \Delta - 1$. This exponential decline causes $\mathrm{fc}(\Delta - 1)$ to be very small: $\mathrm{fc}(\Delta - 1) \leq 2d^{\mu(\Delta)}/p_\Delta$.

(C2) (Monotonicity) Let $\Delta' \leq \Delta - 1$ be the smallest integer for which $\mathrm{fc}(\Delta') \leq 2d^{\mu(\Delta)}/p_\Delta$ holds (such a Δ' exists from the second part of (C1)). Redefine $p_{\Delta'+1} \coloneqq p_\Delta$ and $n_{\Delta'+1} \coloneqq n_\Delta$. We have that $p_{\delta-1} \leq p_\delta$ and $n_{\delta-1} \leq n_\delta$ for all $\delta \leq \Delta' + 1$.

With the notation in place, we can now state the following central claim that constructs the polynomial needed for Lemma 11:

▷ Claim 12. Let Δ, Δ' be as fixed above and $S \subseteq [d]$ be such that $|w_S| \leq k$. Then, there exist J_+, J_-, π such that (S, J_+, J_-, π) is valid and for any integer $\delta \leq \Delta' + 1$ and for all $\tau \in \{0,1\}^{|k_+ - k_-|}$, the polynomial $P_{(S,J_+,J_-,\pi,\tau)}$ can be computed by a set-multilinear formula of product-depth δ and size at most $|S|^\delta 2^{5k\delta d^{\mu(\Delta)}}$.

We finish the proof of Lemma 11 assuming the above claim:

Proof of Lemma 11. As $w_{[d]} \leq k$, applying Claim 12 to $S = [d]$ and $\delta = \Delta' + 1$, gives a polynomial $P_{\Delta'+1} \in \mathbb{F}_{sm}[\overline{X}(w)]$ with $\mathrm{relrk}_w(P_{\Delta'+1}) = 2^{-|w_{[d]}|/2}$. The polynomial $P_{\Delta'+1}$ is computable by a set-multilinear formula of product-depth at most Δ of size at most $d^\Delta 2^{10k\Delta d^{\mu(\Delta)}} \leq n^{O\left(\Delta d^{\mu(\Delta)}\right)}$, since $\Delta' + 1 \leq \Delta$ by definition. ◀

The following claim is the main technical result that helps in proving Claim 12. It is in the same spirit as [20, Claim 28], but we show the existence of a better partition with a more careful analysis. Our analysis holds for any $\alpha \in [0,1]$.

▷ Claim 13. Fix $\delta \leq \Delta' + 1$. Let $S \subseteq [d]$ with $|w_S| \leq k$ such that $|S_+| \leq p_\delta$ and $|S_-| \leq n_\delta$. Then there exists a partition of S as $S_1 \cup S_2 \cup \ldots S_r$ where the following conditions hold:

1. $|S_{i,+}| \leq p_{\delta-1}$ and $|S_{i,-}| \leq n_{\delta-1}$
2. $\sum_{i=1}^{r} |w_{S_i}| \leq 5kd^{\mu(\Delta)}$
3. $|w_{S_i}| \leq k$ for all $i \in [r]$

Proof of Claim 13. As long as possible, pick sets S_i with $|S_{i,+}| = p_{\delta-1}$ positive indices and $|S_{i,-}| = n_{\delta-1}$ negative indices. For all such sets picked, we have

$$|w_{S_i}| = \left|\sum\nolimits_{j\in S_i} w_j\right| = k \cdot |p_{\delta-1}\alpha - n_{\delta-1}| = k \cdot |p_{\delta-1}\alpha - n_{\delta-1}| \leq k \ . \tag{8}$$

Suppose the sets chosen after the procedure are $S_1, \ldots, S_m$, where $m = \min\left\{\left\lfloor\frac{|S_+|}{p_{\delta-1}}\right\rfloor, \left\lfloor\frac{|S_-|}{n_{\delta-1}}\right\rfloor\right\}$ and we are left with the set S'. Since we cannot pick the sets any more, we must have that $|S'_+| < p_{\delta-1}$ or $|S'_-| < n_{\delta-1}$ (or both). We analyze one case, others being analogous.

Say $m = \left\lfloor\frac{|S_+|}{p_{\delta-1}}\right\rfloor$ (i.e. $|S'_+| < p_{\delta-1}$). Also suppose $|S'_-| > n_{\delta-1}$. We pick a set S_{m+1} with $|S'_+|$ positive indices and $p \leq (|S_-| - m \cdot n_{\delta-1})$ negative indices such that

$$|w_{S_{m+1}}| = k\left|\alpha|S'_+| - p\right| = k\left|\alpha(|S_+| - m \cdot p_{\delta-1}) - p\right| \leq k. \tag{9}$$

Note that we can always choose $\alpha|S'_+| - 1 \leq p \leq \alpha|S'_+| + 1$ to satisfy the desired constraints. This follows from observing that $|p_{\delta-1}\alpha - n_{\delta-1}| \leq 1$ which gives $p_{\delta-1}\alpha - 1 \leq n_{\delta-1} \leq p_{\delta-1}\alpha + 1$. Now use the fact that $|S'_-| > n_{\delta-1}$.

The remaining set $T = S' \setminus S_{m+1}$ has only negative values which we split into singletons $S_{m+2}, \ldots, S_r$ (there are $(|S_-| - mn_{\delta-1} - p)$ of these sets). As these are singletons, for $m + 2 \leq j \leq r$ we trivially have $|w_{S_j}| \leq k$.

We also note that since $(|S_-| - m \cdot n_{\delta-1} - p)$ is positive, it is equal to $|m \cdot n_{\delta-1} + p - |S_-||$, which can be rewritten as $|(\alpha|S_+| - |S_-|) - (m(p_{\delta-1}\alpha - n_{\delta-1})) - (\alpha(|S_+| - m \cdot p_{\delta-1}) - p)|$. Using the triangle inequality, we can upper bound this quantity by the sum of $|\alpha|S_+| - |S_-||$, $|m(p_{\delta-1}\alpha - n_{\delta-1})|$ and $|\alpha(|S_+| - mp_{\delta-1}) - p|$. The first term is less than 1 since $|w_S| \leq k$ and the last term is less than 1 from (9). Putting it all together, we have

$$(|S_-| - m \cdot n_{\delta-1} - p) \leq |m(p_{\delta-1}\alpha - n_{\delta-1})| + 2. \tag{10}$$

Finally,

$$\begin{aligned}
\sum_{i=1}^{r} |w_{S_i}| &= \sum_{i=1}^{m} |w_{S_i}| + |w_{S_{m+1}}| + \sum_{i=m+2}^{r} |w_{S_i}| \\
&\leq km|p_{\delta-1}\alpha - n_{\delta-1}| + k + k(|S_-| - m \cdot n_{\delta-1} - p) \\
&\leq km|p_{\delta-1}\alpha - n_{\delta-1}| + k + k\,|m(p_{\delta-1}\alpha - n_{\delta-1})| + 2k \qquad \text{(using (10))} \\
&\leq k\left(2\left\lfloor\frac{|S_+|}{p_{\delta-1}}\right\rfloor |p_{\delta-1}\alpha - n_{\delta-1}| + 3\right) \leq k\left(2|S_+|\frac{|p_{\delta-1}\alpha - n_{\delta-1}|}{p_{\delta-1}} + 3\right) \\
&\leq k\left(2p_\delta \cdot \mathrm{fc}(\delta-1) + 3\right) \qquad \text{(By definition of fc)} \\
&\leq 5kd^{\mu(\Delta)}
\end{aligned}$$

where the last inequality is true because $\mathrm{fc}(\delta - 1) \leq 2d^{\mu(\Delta)}/p_\delta$ holds for $\delta \leq \Delta'$ by the definition of fc and p_δ; it also holds for $\delta = \Delta' + 1$ by the definition of Δ'. ◁

Armed with all this, the proof of Claim 12 becomes quite similar to the proof of Claim 27 in [20] (we refer the reader to the full version for details).

Handling more than two weights. To handle the case when there are multiple weights, we partition the index set $[d]$ into sets $\{S_i\}$ such that the sub-word indexed by each S_i contains at most two distinct weights (details in the full version). We can assume without loss of generality that all entries of w are integers as before.

▶ **Lemma 14.** *Let $w \in \{\alpha_1, \ldots, \alpha_\gamma\}^d$ ($|\alpha_i| \leq k$ for all i) be a word with $\gamma \leq d$ different weights and $|w_{[d]}| \leq k$. Then, the index set $[d]$ can be partitioned as $S_1 \cup \ldots \cup S_\eta$ with $\eta \leq 6\gamma$ such that for all $i \in [\eta]$, the sub-word $w_{|S_i}$ has at most two distinct weights and $|w_{S_i}| \leq k$.*

We can now use Claim 12 to construct polynomials with small set-multilinear formula size but large rank, even when the number of distinct set sizes is not two.

▷ Claim 15. Let $S \subseteq [d]$ and let $w \in \{\alpha_1, \ldots, \alpha_\gamma\}^d$ ($|\alpha_i| \leq k$ for all i) be a word with $\gamma \leq d$ different weights and $|w_S| \leq k$. Then, there exist $(J_+, J_-, \pi), (J'_+, J'_-, \pi')$ such that (S, J_+, J_-, π) and (S, J'_+, J'_-, π') are valid. For any fixed integer Δ and for all $\tau \in \{0,1\}^{|k_+ - k_-|}$, the polynomial $P_{(S,J_+,J_-,\pi,\tau)}$ can be computed by a set-multilinear formula of product-depth Δ and size at most $|S|^\Delta 2^{30k\gamma\Delta d^{\mu(\Delta)}}$ while the polynomial $P_{(S,J'_+,J'_-,\pi',\tau)}$ can be computed by a set-multilinear formula of product-depth Δ and size at most $|S|^\Delta 2^{5k\Delta d^{\mu(\Delta-1)}+6\gamma k}$.

The proof of Claim 15 is quite similar to that of Claim 12 and we prove it in the full version. Assuming the claim, we can finally prove Theorem 3:

Proof of Theorem 3. As $w_{[d]} \leq k$, applying Claim 15 to $S = [d]$, gives polynomials $P_\Delta, Q_\Delta \in \mathbb{F}_{sm}[\overline{X}(w)]$ with relative rank $\mathrm{relrk}_w(P_\Delta) = \mathrm{relrk}_w(Q_\Delta) = 2^{-|w_{[d]}|/2}$ (using the fact that this class of polynomials has maximum possible relative rank).

The polynomial P_Δ has product-depth Δ set-multilinear formula of size at most

$$d^\Delta 2^{30k\gamma\Delta d^{\mu(\Delta)}} \leq n^{O\left(\gamma\Delta d^{\mu(\Delta)}\right)}.$$

The polynomial Q_Δ has product-depth Δ set-multilinear formula of size at most

$$d^\Delta 2^{5k\Delta d^{\mu(\Delta-1)}+6\gamma k} \leq n^{O\left(\Delta d^{\mu(\Delta-1)}+\gamma\right)}.$$ ◀

References

1 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 67–75, 2008. doi:10.1109/FOCS.2008.32.

2 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoret. Comput. Sci.*, 22(3):317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

3 Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig. doi:10.1007/978-3-662-03338-8.

4 Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial derivatives in arithmetic complexity and beyond. *Found. Trends Theor. Comput. Sci.*, 6(1-2):front matter, 1–138 (2011), 2010. doi:10.1561/0400000043.

5 Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth multilinear formula lower bounds for iterated matrix multiplication with applications. *SIAM J. Comput.*, 48(1):70–92, 2019. doi:10.1137/18M1191567.

6 Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth-4 formulas computing iterated matrix multiplication. *SIAM J. Comput.*, 44(5):1173–1201, 2015. doi:10.1137/140990280.

7 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *J. ACM*, 61(6):Art. 33, 16, 2014. doi:10.1145/2629541.

8 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: a chasm at depth 3. *SIAM J. Comput.*, 45(3):1064–1079, 2016. doi:10.1137/140957123.

9 Nikhil Gupta, Chandan Saha, and Bhargav Thankey. A super-quadratic lower bound for depth four arithmetic circuits. In *35th Computational Complexity Conference*, volume 169 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. 23, 31. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2020. doi:10.4230/LIPIcs.CCC.2020.23.

10 K. A. Kalorkoti. A lower bound for the formula size of rational functions. *SIAM J. Comput.*, 14(3):678–687, 1985. doi:10.1137/0214050.

11 Neeraj Kayal. An exponential lower bound for the sum of powers of bounded degree polynomials, 2012. URL: https://eccc.weizmann.ac.il/report/2012/081/.

12 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. *SIAM J. Comput.*, 46(1):307–335, 2017. doi:10.1137/151002423.

13 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In *Symposium on Theory of Computing (STOC)*. ACM - Association for Computing Machinery, June 2014. doi:10.1145/2591796.2591847.

14 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. An almost cubic lower bound for depth three arithmetic circuits. In *43rd International Colloquium on Automata, Languages, and Programming*, volume 55 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 33, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016. doi:10.4230/LIPIcs.ICALP.2016.33.

15 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. On the size of homogeneous and of depth-four formulas with low individual degree. *Theory Comput.*, 14:Paper No. 16, 46, 2018. doi:10.4086/toc.2018.v014a016.

16 Pascal Koiran. Arithmetic circuits: the chasm at depth four gets wider. *Theoret. Comput. Sci.*, 448:56–65, 2012. doi:10.1016/j.tcs.2012.03.041.

17 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *55th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2014*, pages 364–373. IEEE Computer Soc., Los Alamitos, CA, 2014. doi:10.1109/FOCS.2014.46.

18 Mrinal Kumar and Shubhangi Saraf. The limits of depth reduction for arithmetic formulas: it's all about the top fan-in. *SIAM J. Comput.*, 44(6):1601–1625, 2015. doi:10.1137/140999220.

19 Deepanshu Kush and Shubhangi Saraf. Improved low-depth set-multilinear circuit lower bounds. to appear in CCC, 2022. URL: https://eccc.weizmann.ac.il/report/2022/064/.

20 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 804–814, 2022. doi:10.1109/FOCS52979.2021.00083.

21 Meena Mahajan. Algebraic complexity classes. In *Perspectives in computational complexity*, volume 26 of *Progr. Comput. Sci. Appl. Logic*, pages 51–75. Birkhäuser/Springer, Cham, 2014. doi:10.1007/978-3-319-05446-9.

22 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complexity*, 6(3):217–234, 1995. doi:10.1007/BF01294256.

23 Ran Raz. Separation of multilinear circuit and formula size. *Theory Comput.*, 2:121–135, 2006. doi:10.4086/toc.2006.v002a006.

24 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2):Art. 8, 17, 2009. doi:10.1145/1502793.1502797.

25 Ran Raz. Elusive functions and lower bounds for arithmetic circuits. *Theory Comput.*, 6:135–177, 2010. doi:10.4086/toc.2010.v006a007.

26 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. *Github Survey*, 2015. URL: https://github.com/dasarpmar/lowerbounds-survey.

27 Wolfgang M. Schmidt. *Diophantine approximations and Diophantine equations*, volume 1467 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1991. doi:10.1007/BFb0098246.

28 Victor Shoup and Roman Smolensky. Lower bounds for polynomial evaluation and interpolation problems. *Comput. Complexity*, 6(4):301–311, 1996/97. doi:10.1007/BF01270384.

29 Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Comput. Complexity*, 10(1):1–27, 2001. doi:10.1007/PL00001609.

30 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: a survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388 (2010), 2009. doi:10.1561/0400000039.

31 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. In *Mathematical foundations of computer science 2013*, volume 8087 of *Lecture Notes in Comput. Sci.*, pages 813–824. Springer, Heidelberg, 2013. doi:10.1007/978-3-642-40313-2_71.

32 Sébastien Tavenas, Nutan Limaye, and Srikanth Srinivasan. Set-multilinear and non-commutative formula lower bounds for iterated matrix multiplication. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 416–425. ACM, 2022. doi:10.1145/3519935.3520044.

33 Sébastien Tavenas, Srikanth Srinivasan, and Nutan Limaye. On the partial derivative method applied to lopsided set-multilinear polynomials. to appear in CCC, 2022. URL: https://eccc.weizmann.ac.il/report/2022/090/.

34 L. G. Valiant. Completeness classes in algebra. In *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing (Atlanta, Ga., 1979)*, pages 249–261. ACM, New York, 1979. doi:10.1145/800135.804419.

35 L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983. doi:10.1137/0212043.

Conflict-Free Coloring on Claw-Free Graphs and Interval Graphs

Sriram Bhyravarapu
The Institute of Mathematical Sciences, HBNI, Chennai, India

Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, India

Rogers Mathew
Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, India

Abstract

A *Conflict-Free Open Neighborhood coloring*, abbreviated CFON* coloring, of a graph $G = (V, E)$ using k colors is an assignment of colors from a set of k colors to a subset of vertices of $V(G)$ such that every vertex sees some color exactly once in its open neighborhood. The minimum k for which G has a CFON* coloring using k colors is called the *CFON* chromatic number* of G, denoted by $\chi^*_{ON}(G)$. The analogous notion for closed neighborhood is called CFCN* coloring and the analogous parameter is denoted by $\chi^*_{CN}(G)$. The problem of deciding whether a given graph admits a CFON* (or CFCN*) coloring that uses k colors is NP-complete. Below, we describe briefly the main results of this paper.

- For $k \geq 3$, we show that if G is a $K_{1,k}$-free graph then $\chi^*_{ON}(G)= O(k^2 \log \Delta)$, where Δ denotes the maximum degree of G. Dębski and Przybyło in [J. Graph Theory, 2021] had shown that if G is a line graph, then $\chi^*_{CN}(G)= O(\log \Delta)$. As an open question, they had asked if their result could be extended to claw-free ($K_{1,3}$-free) graphs, which are a superclass of line graphs. Since it is known that the CFCN* chromatic number of a graph is at most twice its CFON* chromatic number, our result positively answers the open question posed by Dębski and Przybyło.
- We show that if the minimum degree of any vertex in G is $\Omega(\frac{\Delta}{\log^{\epsilon} \Delta})$ for some $\epsilon \geq 0$, then $\chi^*_{ON}(G)= O(\log^{1+\epsilon} \Delta)$. This is a generalization of the result given by Dębski and Przybyło in the same paper where they showed that if the minimum degree of any vertex in G is $\Omega(\Delta)$, then $\chi^*_{ON}(G)= O(\log \Delta)$.
- We give a polynomial time algorithm to compute $\chi^*_{ON}(G)$ for interval graphs G. This answers in positive the open question posed by Reddy [Theoretical Comp. Science, 2018] to determine whether the CFON* chromatic number can be computed in polynomial time on interval graphs.
- We explore biconvex graphs, a subclass of bipartite graphs and give a polynomial time algorithm to compute their CFON* chromatic number. This is interesting as Abel et al. [SIDMA, 2018] had shown that it is NP-complete to decide whether a planar bipartite graph G has $\chi^*_{ON}(G) = k$ where $k \in \{1, 2, 3\}$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorics

Keywords and phrases Conflict-free coloring, Interval graphs, Bipartite graphs, Claw-free graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.19

Funding The first, second and third authors acknowledge SERB-DST for supporting this research via grants PDF/2021/003452, MTR/2020/000497 and MTR/2019/000550 respectively.

Acknowledgements We would like to thank anonymous referees for helpful comments and corrections, and for suggesting an improvement in the proof of Lemma 6. We would also like to thank Saket Saurabh for helpful discussions.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 19; pp. 19:1–19:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A *Conflict-Free Open Neighborhood coloring*, abbreviated CFON* coloring, of a graph $G = (V, E)$ using k colors is an assignment of colors from a set of k colors to a subset of vertices of $V(G)$ such that every vertex sees some color exactly once in its open neighborhood. The minimum k for which G has a CFON* coloring using k colors is called the *CFON* chromatic number* of G, denoted by $\chi^*_{ON}(G)$.[1] The analogous notion for closed neighborhood is called CFCN* coloring and the analogous parameter is denoted by $\chi^*_{CN}(G)$. It is known (see for instance, Equation 1.3 from [26]) that if G has no isolated vertices, then $\chi^*_{CN}(G)$ is at most twice $\chi^*_{ON}(G)$. Given a graph G and integer $k > 0$, the *CFON* coloring problem* is the problem of determining if $\chi^*_{ON}(G) \leq k$. The CFON* variant is considered to be harder than the CFCN* variant, see for instance, remarks in [22, 26].

The notion of conflict-free coloring was introduced by Even, Lotker, Ron and Smorodinsky in 2004, motivated by the frequency assignment problem in wireless communication [14]. The conflict-free coloring problem on graphs was introduced and first studied by Cheilaris [8] and Pach and Tardos [26]. Conflict-free coloring has found applications in the area of sensor networks [17, 25] and coding theory [23]. Since its introduction, the problem has been extensively studied, see for instance [1, 3, 5, 6, 8, 18, 19, 26, 28]. The decision version of the CFON* coloring problem and many of its variants are known to be NP-complete [1, 18]. In [18], Gargano and Rescigno showed that the optimization version of the CFON* coloring problem is hard to approximate within a factor of $n^{1/2-\epsilon}$, unless $\mathsf{P} = \mathsf{NP}$. Fekete and Keldenich [15] and Hoffmann et al. [21] studied a conflict-free variant of the chromatic Art Gallery Problem, which is about guarding a simple polygon P using a finite set of colored point guards such that each point $p \in P$ sees at least one guard whose color is distinct from all the other guards visible from p.

The conflict-free coloring problem has been studied on several graph classes like planar graphs, split graphs, geometric intersection graphs like interval graphs, unit disk intersection graphs and unit square intersection graphs, graphs of bounded degree, block graphs, etc. [1, 4, 6, 9, 16, 22, 26, 27]. The problem has been studied from parameterized complexity perspective. The problem is fixed-parameter tractable when parameterized by tree-width, neighborhood diversity, distance to cluster, or the combined parameters clique-width and the number of colors [2, 4, 6, 18, 27].

1.1 Our Contribution and Discussion

Below, we discuss the main results of this paper.

The complete bipartite graph $K_{1,3}$ is known as a *claw*. If a graph does not contain a claw as an induced subgraph, then it is called a *claw-free graph*. The *claw number* of a graph G is the largest integer k such that G contains an induced $K_{1,k}$. Dębski and Przybyło [10] showed that if G is a line graph with maximum degree Δ, then $\chi^*_{CN}(G)= O(\log \Delta)$. This bound is tight up to constants. Line graphs are a subclass of claw-free graphs. In [10], it was asked whether the above result can be extended to claw-free graphs. We do this by

[1] It is also known by the name "partial conflict-free chromatic number" as only a subset of vertices are assigned colors. The "(full) conflict-free chromatic number" of a graph, which requires assigning colors to all the vertices, is at most one more than its partial conflict-free chromatic number. We use the notations $\chi^*_{ON}(G)$ and $\chi^*_{CN}(G)$ to be consistent with our other papers on related topics. In our other papers, we use $\chi_{ON}(G)$ and $\chi_{CN}(G)$ to refer to the versions of the problem that require all the vertices to be assigned a color.

proving a more general result. We show that if G is $K_{1,k}$-free with maximum degree Δ, then $\chi^*_{ON}(G)= O(k^2 \log \Delta)$. Since $\chi^*_{CN}(G) \leq 2\chi^*_{ON}(G)$, we have $\chi^*_{CN}(G)= O(k^2 \log \Delta)$ as well. This result is presented in Section 3.2.

What is the maximum number of colors required to CFON* color a graph whose maximum degree is Δ? It can be seen that the graph obtained by subdividing every edge of a complete graph requires $\Delta + 1$ colors. It is known that for a graph G with maximum degree Δ, $\chi^*_{ON}(G)$ is at most $\Delta + 1$ [26]. Pach and Tardos [26] showed that if the minimum degree of any vertex in G is $\Omega(\log \Delta)$, then $\chi^*_{ON}(G)= O(\log^2 \Delta)$. In this direction, Dębski and Przybyło [10] showed that if the minimum degree of any vertex in G is $\Omega(\Delta)$, then the previous upper bound can be improved to show $\chi^*_{ON}(G)= O(\log \Delta)$. We extend the proof idea of [10] to generalize their result. We show that if the minimum degree of any vertex in G is $\Omega(\frac{\Delta}{\log^{\epsilon} \Delta})$ for some $\epsilon \geq 0$, then $\chi^*_{ON}(G)= O(\log^{1+\epsilon} \Delta)$. This result is presented in Section 3.3. A natural open question we have here is, can we get a stronger upper bound for the CFON* chromatic number of a graph with minimum degree $\omega(1)$? When the minimum degree is $o(\log \Delta)$, the only upper bound known is $O(\Delta)$ mentioned above due to [26]. In this situation our first result does give a better (than $O(\Delta)$) upper bound for CFON* chromatic number, if the claw number of the graph under consideration is $o\left(\sqrt{\frac{\Delta}{\log \Delta}}\right)$.

For an interval graph G, it has been shown that [4, 27] $\chi^*_{ON}(G) \leq 3$. It was shown in [4] that there exists an interval graph that requires 3 colors, making the above bound tight. It was asked in [27] if there is a polynomial time algorithm that given an interval graph G, computes $\chi^*_{ON}(G)$. We answer this in the affirmative and give polynomial time characterization algorithms for interval graphs G that decide if $\chi^*_{ON}(G) \in \{1, 2, 3\}$. These results are presented in Section 4.

For a bipartite graph G, it is easy to see that $\chi^*_{CN}(G) \leq 2$. On the contrary, there exist bipartite graphs G, for which $\chi^*_{ON}(G) = \Theta(\sqrt{n})$. It is NP-complete [1] to decide if a planar bipartite graph is CFON* colorable using k colors, where $k \in \{1, 2, 3\}$. We study the problem on some subclasses of bipartite graphs that include chain graphs, biconvex bipartite graphs, and bipartite permutation graphs. We show that three colors are sufficient to CFON* color a biconvex bipartite graph and give characterization algorithms to decide the CFON* chromatic number. The results are presented in Section 5.

2 Preliminaries

Throughout the paper, we consider simple undirected graphs. We denote the vertex set and the edge set of a graph $G = (V, E)$, by $V(G)$ and $E(G)$. For standard graph notations, we refer to the graph theory book by R. Diestel [11]. For a vertex $v \in G$, its *open neighborhood*, denoted by $N_G(v)$, is the set of neighbors of v in G. The *closed neighborhood* of v, denoted by $N_G[v]$, is $N_G(v) \cup \{v\}$. We use log to denote the logarithm to the base 2, and ln to denote the natural logarithm. Proofs of the results marked with ($\star$) are omitted due to space constraints.

3 Improved bounds for $\chi^*_{ON}(G)$ for graphs with bounded claw number

The graph $K_{1,k}$ is the complete bipartite graph on $k+1$ vertices with one vertex in one part and the remaining k vertices in the other part.

▶ **Definition 1** (Claw number). *The* claw number *of a graph G is the smallest k such that G is $K_{1,k+1}$-free. In other words, it is the largest k such that G contains an induced $K_{1,k}$.*

The complete bipartite graph $K_{1,3}$ is called a *claw*. A graph is called a *claw-free graph* if it does not contain a claw as an induced subgraph.

In this section, we prove two results: (i) an improved bound for $\chi^*_{ON}(G)$ in terms of the claw number and maximum degree of G, and (ii) an improved bound for $\chi^*_{ON}(G)$ for graphs with high minimum degree. We begin by stating a couple of results from probability theory which will be useful.

▶ **Lemma 2** (*The Local Lemma*, [13]). *Let $A_1, \ldots, A_n$ be events in an arbitrary probability space. Suppose that each event A_i is mutually independent of a set of all the other events A_j but at most d, and that $Pr[A_i] \leq p$ for all $i \in [n]$. If $4pd \leq 1$, then $Pr[\cap_{i=1}^{n} \overline{A_i}] > 0$.*

▶ **Theorem 3** (Chernoff Bound, Corollary 4.6 in [24]). *Let $X_1, \ldots, X_n$ be independent Poisson trials such that $Pr[X_i] = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. For $0 < \delta < 1$, $Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\mu\delta^2/3}$.*

3.1 Auxiliary lemmas

In this subsection, we state some auxiliary lemmas on conflict-free chromatic number of graphs and hypergraphs having certain structural characteristics that will be used to prove the main theorems in Sections 3.2 and 3.3. Before we begin, let us define the conflict-free chromatic number of a hypergraph.

▶ **Definition 4.** *Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a coloring $c : V \rightarrow [r]$ is a* conflict-free coloring *of $\mathcal{H}$ if for every hyperedge $E \in \mathcal{E}$, there is a vertex in E that receives a color under c that is distinct from the colors received by all the other vertices in E. The minimum r such that $c : V \rightarrow [r]$ is a conflict-free coloring of $\mathcal{H}$ is called the* conflict-free chromatic number *of $\mathcal{H}$. This is denoted by $\chi_{CF}(\mathcal{H})$.*

The following theorem on conflict-free coloring of hypergraphs is from [26]. The degree of a vertex in a hypergraph is the number of hyperedges it is part of.

▶ **Theorem 5** (Theorem 1.1(b) in [26]). *Let $\mathcal{H}$ be a hypergraph and let Δ be the maximum degree of any vertex in $\mathcal{H}$. Then, $\chi_{CF}(\mathcal{H}) \leq \Delta + 1$.*

We prove an upper bound for the conflict-free chromatic number of a "near uniform hypergraph" in Lemma 6 below.

▶ **Lemma 6.** *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph where (i) every hyperedge intersects with at most Γ other hyperedges, and (ii) for every hyperedge $E \in \mathcal{E}$, $r \leq |E| \leq \ell r$, where $\ell \geq 1$ is some integer and $r \geq 2\log(4\Gamma)$. Then, $\chi_{CF}(\mathcal{H}) \leq e\ell r$, where e is the base of natural logarithm.*

Proof. For each vertex in V, assign a color that is chosen independently and uniformly at random from a set of $e\ell r$ colors. We will first show that the probability of this coloring being bad for an edge is small, and then use Local Lemma to show the existence of conflict-free coloring for $\mathcal{H}$ using at most $e\ell r$ colors.

Consider a hyperedge $E \in \mathcal{E}$ with $m := |E|$. By assumption, we have $r \leq m \leq \ell r$. Let A_E denote the bad event that E is colored with $\leq |E|/2$ colors. Note that if A_E does not occur, then E is colored with $> |E|/2$ colors, hence there is at least one color that appears exactly once in E.

$$\begin{aligned}
Pr[A_E] &\leq \binom{e\ell r}{m/2}\left(\frac{m/2}{e\ell r}\right)^m \\
&\leq \left(\frac{e^2\ell r}{m/2}\right)^{m/2}\left(\frac{m/2}{e\ell r}\right)^m \qquad \text{(since } \binom{n}{k} \leq \left(\frac{en}{k}\right)^k\text{)} \\
&= \frac{(m/2)^{m/2}}{(\ell r)^{m/2}} \quad = \quad \left(\frac{m}{2\ell r}\right)^{m/2} \\
&\leq (1/2)^{m/2} \quad \leq \quad \frac{1}{4\Gamma}.
\end{aligned}$$

Here the penultimate inequality follows since $m \leq \ell r$, and the last inequality follows since $m \geq 2\log(4\Gamma)$.

We apply the Local Lemma (Lemma 2) on the events A_E for all hyperedges $E \in \mathcal{E}$. Since each hyperedge intersects with at most Γ other hyperedges, and $4 \cdot \frac{1}{4\Gamma} \cdot \Gamma \leq 1$, we get $Pr[\cap_{E\in\mathcal{E}}(\overline{A}_E)] > 0$. That is, there is a conflict free coloring of $\mathcal{H}$ that uses at most $e\ell r$ colors. This completes the proof of the lemma. ◀

Lemmas 7 and 8 prove upper bounds for $\chi^*_{ON}(G)$ when G satisfies certain degree restrictions.

▶ **Lemma 7.** *Let G be a graph with (i) $V(G) = X \uplus Y$, $X, Y \neq \emptyset$, (ii) every vertex in G has at most d_X neighbors in X, (iii) every vertex in Y has at least one neighbor in X, and (iv) every vertex in X has at most d_Y neighbors in Y. Then, there is a coloring of vertices of X with $d_X d_Y + d_X - d_Y + 1$ colors such that every vertex in Y sees some color exactly once among its neighbors in X.*

Proof. For each vertex $y \in Y$, we arbitrarily choose one of its neighbors in X. Let us call this neighbor $f(y)$. For each $y \in Y$, contract the edges $\{y, f(y)\}$ to obtain a resulting graph G_X. Note that the vertex set of G_X is $V(G_X) = X$. The maximum degree of a vertex in the new graph G_X is at most $(d_X - 1)d_Y + d_X$. Thus, we can do a proper coloring (such that no pair of adjacent vertices receive the same color) of G_X using $d_X d_Y + d_X - d_Y + 1$ colors. We note that this coloring of the vertices of X satisfies our requirement: in the original graph G, for each $y \in Y$, the neighbor $f(y)$ is colored distinctly from all the other neighbors of y in X. ◀

▶ **Lemma 8.** *Let G be a graph with (i) $V(G) = X \uplus Y$, $X, Y \neq \emptyset$, (ii) every vertex in Y has at most t_X neighbors in X, and (iii) every vertex in X has at least one neighbor in Y. Then, there is a coloring of the vertices of Y using at most $(t_X + 1)$ colors such that every vertex in X sees some color exactly once among its neighbors in Y.*

Proof. For every vertex $v \in X$, let $N^Y_G(v)$ denote the set $N_G(v) \cap Y$, i.e., the neighbors of v in Y in the graph G. Since every vertex in X has at least one neighbor in Y, we have, $|N^Y_G(v)| \geq 1$. We construct a hypergraph $\mathcal{H} = (V, \mathcal{E})$ from G as described below. We have (i) $V = Y$, and (ii) $\mathcal{E} = \{N^Y_G(v) \ : \ v \in X\}$. Since every vertex in Y has at most t_X neighbors in X in the graph G, the maximum degree of a vertex in the hypergraph $\mathcal{H}$ (that is, the maximum number of hyperedges a vertex in $\mathcal{H}$ is part of) is at most t_X. From Theorem 5, we have $\chi_{CF}(\mathcal{H}) \leq t_X + 1$. Observe that in this coloring of the vertices of Y using at most $(t_X + 1)$ colors, every vertex in X sees some color exactly once among its neighbors in Y. ◀

The following lemma, which will be used in the proof of Theorem 12, shows that given a graph with high minimum degree there exists a subset of vertices that, for every vertex, intersects its neighborhood at a small number of vertices.

▶ **Lemma 9.** *Let Δ denote the maximum degree of a graph G. It is given that every vertex in G has degree at least $\frac{c\Delta}{\log^{\epsilon}\Delta}$ for some $\epsilon \geq 0$ and c is a constant. Then, there exists $A \subseteq V(G)$ such that for every vertex $v \in V(G)$,*

$$75\log(2\Delta) < |N_G(v) \cap A| < \frac{125}{c}\log^{1+\epsilon}(2\Delta).$$

Proof. We construct a random subset A of $V(G)$ as described below. Each $v \in V(G)$ is independently chosen into A with probability $\frac{100\log^{1+\epsilon}(2\Delta)}{c\Delta}$. For a vertex $v \in V(G)$, let X_v be a random variable that denotes $|N_G(v) \cap A|$. Then, $\mu_v := E[X_v] = \frac{100\log^{1+\epsilon}(2\Delta)}{c\Delta}d_G(v) \geq 100\log(2\Delta)$. Since $d_G(v) \leq \Delta$, we also have $\mu_v \leq \frac{100\log^{1+\epsilon}(2\Delta)}{c}$. Let B_v denote the event that $|X_v - \mu_v| \geq \frac{\mu_v}{4}$. Applying Theorem 3 with $\delta = 1/4$, we get $Pr[B_v] = Pr[|X_v - \mu_v| \geq \frac{\mu_v}{4}] \leq 2e^{-\frac{\mu_v}{48}} \leq 2e^{-\frac{100\log(2\Delta)}{48}} = 2e^{-\frac{100\ln(2\Delta)}{48\ln 2}} < \frac{2}{(2\Delta)^3}$. The event B_v is mutually independent of all but those events B_u where $N_G(u) \cap N_G(v) \neq \emptyset$. Hence, every event B_v is mutually independent of all but at most Δ^2 other events. Applying Lemma 2 with $p = Pr[B_v] \leq \frac{2}{(2\Delta)^3}$ and $d = \Delta^2$, we have $4 \cdot \frac{2}{(2\Delta)^3} \cdot \Delta^2 \leq 1$. Thus, there is a non-zero probability that none of the events B_v occur. In other words, for every v, it is possible to have $\frac{3}{4}\mu_v < X_v < \frac{5}{4}\mu_v$. Using the upper and lower bounds of μ_v we computed above, we can say that there exists an A such that, for every v, $75\log(2\Delta) < |N_G(v) \cap A| < \frac{125}{c}\log^{1+\epsilon}(2\Delta)$. ◀

3.2 Graphs with bounded claw number

▶ **Theorem 10.** *Let G be a $K_{1,k}$-free graph with maximum degree Δ having no isolated vertices. Then, $\chi^*_{ON}(G) = O(k^2\log\Delta)$.*

Proof. Consider a proper coloring (such that no pair of adjacent vertices receive the same color) of G, $h : V(G) \to [\Delta+1]$, using $\Delta+1$ colors. Let $C_1, C_2, \ldots, C_{\Delta+1}$ be the color classes given by this coloring G. That is, $V(G) = C_1 \uplus C_2 \uplus \cdots \uplus C_{\Delta+1}$ is the partitioning of the vertex set of G given by the coloring, where each C_i is an independent set. We may assume that the coloring h satisfies the following property: for every $1 < i \leq \Delta+1$, every vertex v in C_i has at least one neighbor in every C_j, where $1 \leq j < i$ (otherwise, we can move v to a color class C_j, $j < i$, in which it has no neighbors without compromising on the "properness" of the coloring). Since G is $K_{1,k}$-free, we have the following observation.

▶ **Observation 11.** *For every $i \in [\Delta+1]$, a vertex in G has at most $k-1$ neighbors in C_i.*

Let $r = 2\log(4\Delta^2)$. We partition the vertex set of G into three parts, namely V_1, V_2, and V_3 as described below. We have $V_1 := C_1$. If $\Delta > r$, then $V_2 := C_2 \uplus C_3 \uplus \cdots \uplus C_{r+1}$ and $V_3 := C_{r+2} \uplus C_{r+3} \uplus \cdots \uplus C_{\Delta+1}$. Otherwise, $V_2 := C_2 \uplus C_3 \uplus \cdots \uplus C_{\Delta+1}$ and $V_3 := \emptyset$.

The rest of the proof is about constructing a coloring $f : V(G) \to \mathbb{N} \times \mathbb{N}$ that is a CFON* coloring of G. Let $N_1 = \{1, 2, \ldots, r_1\}$, $N_2 = \{r_1+1, r_1+2, \ldots, r_1+r_2\}$, and $N_3 = \{r_1+r_2+1, r_1+r_2+2, \ldots, r_1+r_2+r_3\}$, where $|N_1| = r_1 = (k-1)(k-2)r + k$, $|N_2| = r_2 = e(k-1)r$, and $|N_3| = r_3 = k$. We define three colorings f_1, f_2, and f_3 below.

We begin by describing the coloring $f_1 : V_1 \to N_1$. Let $G[V_1 \cup V_2]$ be the subgraph of G induced on $V_1 \cup V_2$. From Observation 11, every vertex in $G[V_1 \cup V_2]$ has at most $k-1$ neighbors in $V_1 = C_1$. Every vertex in V_2 has at least one neighbor in V_1 due to the property of our coloring h. From Observation 11, we can also say that every vertex in V_1 has at most

$r(k-1)$ neighbors in V_2. Applying Lemma 7 on $G[V_1 \cup V_2]$ with $X = V_1$, $Y = V_2$, $d_X = k-1$ and $d_Y = r(k-1)$, we can say that there is a coloring $f_1 : V_1 \to N_1$ of the vertices of V_1 with $(k-1)(k-2)r + k$ colors such that every vertex in V_2 sees some color exactly once among its neighbors in V_1.

We now describe the coloring $f_2 : V_2 \to N_2$. If $V_3 = \emptyset$, then, $\forall v \in V_2$, $f_2(v) = r_1 + 1$. Suppose $V_3 \neq \emptyset$. For a vertex v in G, let $N_G^{V_2}(v)$ denote the set of neighbors of v in V_2 in the graph G. We construct a hypergraph $\mathcal{H}_2 = (V_2, \mathcal{E}_2)$ as follows. We have $\mathcal{E}_2 = \{N_G^{V_2}(v) \ : \ v \in V_3\}$. Consider an arbitrary hyperedge $E \in \mathcal{E}_2$. In the graph G, since every vertex in V_3 has at least one neighbor in every color class C_i, $2 \leq i \leq r+1$, $|E| \geq r$. Using Observation 11, we can say that $|E| \leq (k-1)r$. As $|N_G^{V_2}(v)| \leq N_G(v) \leq \Delta, \forall v \in V(G)$, we have $|E| \leq \Delta$. This also implies that E intersects with at most Δ^2 other hyperedges in $\mathcal{E}_2$. Applying Lemma 6 with $\ell = (k-1)$ and $\Gamma = \Delta^2$, we have $\chi_{CF}(\mathcal{H}_2) \leq e(k-1)r$. Thus, there is a coloring $f_2 : V_2 \to N_2$ of the vertices V_2 such that every vertex in V_3 sees some color exactly once among its neighbors in V_2.

Finally, we describe the coloring $f_3 : V_2 \cup V_3 \to N_3$. From Observation 11, every vertex in $V_2 \cup V_3$ has at most $k-1$ neighbors in $V_1 = C_1$. Since there are no isolated vertices in G, every vertex in V_1 has at least one neighbor in $V_2 \cup V_3$. Applying Lemma 8 with $X = V_1$, $Y = V_2 \cup V_3$, and $t_X = k-1$, we get a coloring $f_3 : V_2 \cup V_3 \to N_3$ of the vertices of $V_2 \cup V_3$ using at most k colors such that every vertex in V_1 sees some color exactly once among its neighbors in $V_2 \cup V_3$.

We are now ready to define the coloring f.

$$f(v) = \begin{cases} (1, f_1(v)), \text{ if } v \in V_1 \\ (f_2(v), f_3(v)), \text{ if } v \in V_2 \\ (1, f_3(v)), \text{ if } v \in V_3. \end{cases}$$

We now argue that f is indeed a CFON* coloring of G. Consider a vertex $v \in V(G)$. If $v \in V_3$, v sees some color exactly once among its neighbors in V_2 under the coloring f_2. Let u be that neighbor of v in V_2 and $f_2(u)$ be that color that appears exactly once in the neighborhood of v in V_2. Since the codomains of f_1, f_2, and f_3 are pairwise disjoint sets, v does not see the same color among its neighbors in V_1 or in V_2. Further, since $f(u) = (f_2(u), f_3(u))$, the final coloring f only refines the color classes of V_2 given by f_2. Thus, the color $(f_2(u), f_3(u))$ appears exactly once among the neighbors of v in G. The cases when $v \in V_1$ and $v \in V_2$ also follow using similar arguments.

The coloring f uses at most $|N_1| + |N_2||N_3| + |N_3| = (k-1)(k-2)r + k + e(k-1)kr + k$ colors. Since $r = O(\log \Delta)$, this implies that $\chi_{CF}^{ON}(G) = O(k^2 \log \Delta)$. ◀

3.3 Graphs with high minimum degree

When a graph G has high minimum degree, the following theorem gives improved upper bounds for $\chi_{ON}^*(G)$ in terms of its maximum degree.

▶ **Theorem 12.** *Let G be a graph with maximum degree Δ. It is given that every vertex in G has degree at least $\frac{c\Delta}{\log^{\epsilon} \Delta}$ for some $\epsilon \geq 0$ and c is a constant. Then, $\chi_{ON}^*(G) = O(\log^{1+\epsilon} \Delta)$.*

Proof. Apply Lemma 9 to find an $A \subseteq V(G)$ such that for every $v \in V(G)$, $75 \log(2\Delta) < |N_G(v) \cap A| < \frac{125}{c} \log^{1+\epsilon}(2\Delta)$. Construct a hypergraph $\mathcal{H} = (A, \mathcal{E})$ where $\mathcal{E} = \{N_G(v) \cap A \ : \ v \in V(G)\}$. Every $E \in \mathcal{E}$ satisfies $2\log(4\Delta^2) < 75 \log(2\Delta) < |E| < \frac{125}{c} \log^{1+\epsilon}(2\Delta)$. Applying Lemma 6 with $r = 75 \log(2\Delta)$ and $\ell = \frac{5}{3c} \log^{\epsilon}(2\Delta)$, we get $\chi_{CF}(\mathcal{H}) \leq \frac{340}{c} \log^{1+\epsilon}(2\Delta)$. It is easy to see that this conflict-free coloring of $\mathcal{H}$ is indeed a CFON* coloring for G. ◀

4 Interval graphs

In this section, we show that the problem of determining the CFON* chromatic number of a given interval graph is polynomial time solvable. It was shown in [4, 27] that, for an interval graph G, $\chi^*_{ON}(G) \leq 3$ and that there exists an interval graph that requires three colors. The complexity of the problem on interval graphs was posed as an open question in the above papers. We show that CFON* coloring is polynomial time solvable. That is, given an interval graph G, in polynomial time we decide whether $\chi^*_{ON}(G)$ is 1, 2 or 3. We state it formally below.

▶ **Theorem 13.** *Given an interval graph G, there is a polynomial time algorithm that determines $\chi^*_{ON}(G)$.*

▶ Remark 14 (Notation). In the introduction, we defined CFON* coloring to be an assignment of colors to a *subset* of the vertices. For the sake of convenience, we will use the color 0 to denote uncolored vertices. That is, we will use an assignment $f : V(G) \rightarrow \{0, 1, 2\}$, to denote a coloring that assigns the colors 1 and 2 to some vertices. The vertices that are assigned 0 by f are the "uncolored" vertices. The "color" 0 cannot serve as a unique color in the neighborhood of any vertex.

▶ **Definition 15** (Interval Graphs). *A graph $G = (V, E)$ is called an* interval graph *if there exists a set of intervals on the real line such that the following holds: (i) there is a bijection between the intervals and the vertices and (ii) there exists an edge between two vertices if and only if the corresponding intervals intersect.*

The main ingredient of the algorithm is the use of *multi-chain ordering* property on interval graphs. Before defining the multi-chain ordering property, we look at some prerequisites.

▶ **Definition 16** (Chain Graph [12]). *A bipartite graph $G = (A, B)$ is a* chain graph *if and only if for any two vertices $u, v \in A$, either $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$. If G is a chain graph, it follows that for any two vertices $u, v \in B$, either $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$.*

As a consequence, we can order the vertices in B in the decreasing order of the degrees. We can break ties arbitrarily. If $b_1 \in B$ appears before $b_2 \in B$ in the ordering, then it follows that $N(b_2) \subseteq N(b_1)$.

▶ **Definition 17** (Multi-chain Ordering [7, 12]). *Given a connected graph $G = (V, E)$, we arbitrarily choose a vertex as $v_0 \in V(G)$ and construct distance layers $L_0, L_1, \ldots, L_p$ from v_0. The layer L_i, where $i \in [p]$, represents the set of vertices that are at a distance i from v_0. Note that p here denotes the largest integer such that L_p is non-empty.*

We say that these layers form a multi-chain ordering *of G if for every two consecutive layers L_i and L_{i+1}, where $i \in \{0, 1, \ldots, p-1\}$, we have that the vertices in L_i and L_{i+1}, and the edges connecting these layers form a chain graph.*

▶ **Theorem 18** (Theorem 2.5 of [12]). *All connected interval graphs admit multi-chain orderings.*

We give a characterization of interval graphs that require one color and two colors in polynomial time in Theorem 21 and Theorem 23 respectively. Given an interval graph G, the algorithms decide if G is CFON* colorable using one color or two colors. If G is not CFON* colorable using one color or two colors, we conclude that G is CFON* colorable using three colors (since it is known that for an interval graph G, $\chi^*_{ON}(G) \leq 3$). One of the key ideas used in Theorem 23 (to decide if G can be CFON* colored using two nonzero colors) is sort of a bootstrapping idea. After narrowing down the possibilities, we need to test if a given subgraph can be colored using the colors $\{0, 1\}$ so as to obtain a CFON* coloring. To solve this, we use Theorem 21.

Before we proceed to the main theorems of this section, we observe the following on a graph G that admits multi-chain ordering.

▶ **Observation 19.** *If G admits a multi-chain ordering, then every distance layer L_i, for $0 \leq i < p$ contains a vertex v such that $N(v) \supseteq L_{i+1}$.*

Proof. Consider a multi-chain ordering of G, starting with an arbitrary vertex. For any two consecutive distance layers L_i and L_{i+1}, it can be seen that each vertex in L_{i+1} has a neighbor in L_i. This, together with the fact that L_i and L_{i+1} form a chain graph, imply that there is a vertex $v \in L_i$ such that $N(v) \supseteq L_{i+1}$. ◀

▶ **Observation 20.** *In any CFON* coloring of G that uses one color, at most one vertex in each L_i is assigned the color 1.*

Proof. Consider a layer L_i of the graph. As per Observation 19, there is a $v \in L_i$ such that $N(v) \supseteq L_{i+1}$. If two vertices in L_{i+1} are colored 1, then the vertex $v \in L_i$ does not have a uniquely colored neighbor. Hence in all the layers $L_1, L_2, \ldots$ up to the last layer L_p, we have that at most one vertex is assigned the color 1. Since L_0 has only one vertex, the statement is trivially true for L_0. ◀

▶ **Theorem 21.** *Given an interval graph $G = (V, E)$, we can decide in $O(n^5)$ time if $\chi^*_{ON}(G) = 1$.*

Proof. Let $L_0, L_1, \ldots, L_p$ be the distance layers of G constructed from an arbitrarily chosen vertex v_0, satisfying the multi-chain ordering. If there is a CFON* coloring that uses 1 color, then from Observation 20, at most one vertex in each layer is assigned the color 1. There are two possibilities for a layer L_i: either it has no vertices colored 1, or it has exactly one vertex that is colored 1. In the former case, there is a unique coloring for L_i when none of the vertices in L_i are assigned the color 1. In the latter case, we have $|L_i|$ many colorings (for L_i) where each coloring has exactly one vertex with color 1 (and the rest are assigned 0). In total, we have at most $|L_i| + 1$ colorings for each L_i. We call all such colorings *valid*.

The task is to find if there is a sequence of colorings assigned to each layer of G such that we have a CFON* coloring. Notice that the vertices in L_i can possibly have neighbors in the layers L_{i-1}, L_i, and L_{i+1}. The question of deciding whether the vertices in L_i have a uniquely colored neighbor entirely depends on the colorings assigned to these three layers. We say that colorings assigned to three consective layers are *good* if the vertices in the central layer have uniquely colored neighbors. We use a dynamic programming based approach to verify the existence of a CFON* coloring for G.

We now construct a layered companion hypergraph $\mathcal{G} = (V', \mathcal{E})$ with vertices in $p + 1$ layers. Each layer T_i of $\mathcal{G}$ corresponds to the layer L_i of G where $i \in [p] \cup \{0\}$. Each vertex in layer T_i of $\mathcal{G}$ corresponds to a valid coloring of vertices in L_i of G. Hence the number of vertices in each layer T_i of $\mathcal{G}$ is equal to $|L_i| + 1$. We now explain how the hyperedges $\mathcal{E}$ of $\mathcal{G}$ are determined.

For $1 \leq i \leq p-1$, the vertices $x \in T_{i-1}$, $y \in T_i$, $z \in T_{i+1}$ form a hyperedge $\{x, y, z\}$ if the corresponding colorings, when assigned to L_{i-1}, L_i and L_{i+1} respectively, ensures that every vertex in L_i has a uniquely colored neighbor. We also have hyperedges $\{y, z\}$, where $y \in T_0$ and $z \in T_1$ are colorings such that when y and z are assigned to L_0 and L_1 respectively, the vertex in L_0 sees a uniquely colored neighbor. Similarly, we have hyperedges $\{x, y\}$, where $x \in T_{p-1}$ and $z \in T_p$ are colorings such that when x and y are assigned to L_{p-1} and L_p respectively, all the vertices in L_p see a uniquely colored neighbor.

Since the number of valid colorings is $|L_i| + 1$ for the layer L_i, the total number of valid colorings across all layers is at most $2n$. The total number of potential hyperedges to check is at most $O(n^3)$. Once we fix valid colorings x_{i-1}, x_i, x_{i+1} for L_{i-1}, L_i, L_{i+1} respectively, we can check in $O(|L_i| \cdot n) \leq O(n^2)$ time if $\{x_{i-1}, x_i, x_{i+1}\} \in \mathcal{E}$. Hence we need $O(n^5)$ time to construct $\mathcal{G}$.

To obtain a CFON* coloring for G, we need to construct a sequence of colorings $x_0 \in T_0$, $x_1 \in T_1, \ldots, x_p \in T_p$ such that $\{x_0, x_1\} \in \mathcal{E}$, $\{x_{i-1}, x_i, x_{i+1}\} \in \mathcal{E}$ for all $1 \leq i \leq p-1$, and finally $\{x_{p-1}, x_p\} \in \mathcal{E}$. For this, we use Lemma 22, stated and proved below. Since each $|T_i| = |L_i| + 1 \leq n + 1$, and number of layers is at most n, this takes at most $O(n^4)$ time. The construction of $\mathcal{G}$ takes $O(n^5)$ time and dominates the running time. ◀

▶ **Lemma 22.** *Suppose there is a layered hypergraph $\mathcal{G} = (V', \mathcal{E})$ with layers $T_0, T_1, T_2, \ldots, T_p$, where $|T_i| \leq \alpha$, for $0 \leq i \leq p$ and $p \leq \beta$. Suppose further that all the hyperedges in $\mathcal{E}$ contain one vertex each from three consecutive layers, or contain one vertex each from T_0 and T_1, or contain one vertex each from T_{p-1} and T_p. We can determine if there exists a sequence $x_0 \in T_0$, $x_1 \in T_1, \ldots, x_p \in T_p$ such that $\{x_0, x_1\} \in \mathcal{E}$, $\{x_{i-1}, x_i, x_{i+1}\} \in \mathcal{E}$ for all $1 \leq i \leq p-1$, and finally $\{x_{p-1}, x_p\} \in \mathcal{E}$ in $O(\alpha^3\beta)$ time.*

Proof. We start with the vertices in T_0. For each vertex $x_1 \in T_1$, we store a list of predecessors x_0 such that $\{x_0, x_1\} \in \mathcal{E}$. For $1 \leq i \leq p-1$, we do the following at each vertex $x_i \in T_i$. We look at the list of predecessors stored. If x_{i-1} is a listed predecessor of x_i, then we search for all the hyperedges $\{x_{i-1}, x_i, z\}$, where $z \in T_{i+1}$. If we find such a hyperedge $\{x_{i-1}, x_i, x_{i+1}\} \in \mathcal{E}$, then we store x_i as a predecessor in the list at x_{i+1}. Finally, for each $x_p \in T_p$, we check if there is a listed predecessor $z \in T_{p-1}$ of x_p such that $\{z, x_p\} \in \mathcal{E}$. If there is any such $x_p \in T_p$ for which this holds, then there exists a sequence as desired in the statement of the lemma.

Note that the general step involves going through a list of size at most α at each vertex x_i. For each listed predecessor x_{i-1}, there are potentially at most α hyperedges of the form $\{x_{i-1}, x_i, z\}$ to check, where $z \in T_{i+1}$. We need to do this for all the vertices (at most α of them) of T_i. This gives a time complexity of $O(\alpha^3)$ at the i-th layer. Since there are β layers, the total running time is $O(\alpha^3\beta)$. ◀

We now proceed to the next result that decides in polynomial time whether $\chi_{ON}(G) = 2$.

▶ **Theorem 23** ($\star$)**.** *Given an interval graph G, we can decide in $O(n^{20})$ time if $\chi_{ON}(G) = 2$.*

Sketch of Proof. The idea of this proof is similar to the proof of Theorem 21. For a layer $|L_i|$, we had $|L_i| + 1$ colorings to consider in Theorem 21. Unlike in Theorem 21, we have more colorings to consider since the vertices can get the colors $\{0, 1, 2\}$. We have the following types of colorings in each layer L_i:

Type 1: All the vertices in L_i are assigned the color 0. There is only one coloring of L_i of this type.

Type 2: Exactly one vertex is assigned the color 1 or 2 while the rest are assigned the color 0. The number of colorings is $2|L_i|$.

Type 3: Both the colors 1 and 2 appear exactly once and the rest are assigned the color 0. The number of colorings is $|L_i|(|L_i| - 1) \leq |L_i|^2$.

Type 4: One of the colors 1 or 2 appears at least twice while the other color appears exactly once. The remaining vertices are assigned the color 0.

Type 5: One of the colors 1 or 2 appears at least twice and all the other vertices are assigned the color 0.

Due to space constraints, the full proof is omitted. We describe a proof sketch highlighting the key ideas in the proof below.

- The above 5 types are exhaustive. We cannot have a "Type 6" coloring in L_{i+1} where there are at least two vertices with color 1 and at least two vertices with color 2. This is because Observation 19 implies the existence of a vertex $v \in L_i$ such that $N(v) \supseteq L_{i+1}$. This implies that v does not have a uniquely colored neighbor for such a coloring of L_{i+1}.
- The number of colorings of Types 1, 2, 3 are polynomial in $|L_i|$ while the number of colorings of Types 4 and 5 are exponential in $|L_i|$. Since we cannot consider an exponential number of colorings, we consider a polynomial subset of Type 4 and Type 5 colorings which are representatives of all possible Type 4 and Type 5 colorings.
- Given a Type 4 or Type 5 coloring, the key point is that it is enough to fix the colors of a few vertices that we will refer to as "left-important" and "right-important" vertices. This allows us to restrict the focus onto a reduced number of representative colorings.
- Because of the flexibility offered by the representative colorings, there are some cases where we have to explore further in order to decide if the graph is CFON* colorable using colors from $\{0, 1, 2\}$. This reduces to the problem of testing whether a given subgraph is CFON* colorable using colors from $\{0, 1\}$. We use Theorem 21 (with some minor changes) to accomplish this. This is the last, but critical step that we need to complete the proof. ◀

Using Theorems 21 and 23, we can now infer Theorem 13.

▶ Remark 24. Recently, the work of Gonzalez and Mann [20] (done simultaneously and independently from ours) on mim-width showed that the CFON* coloring problem is polynomial-time solvable on graph classes for which a branch decomposition of constant mim-width can be computed in polynomial time. This includes the class of interval graphs. We note that our work gives a more explicit algorithm without having to go through the machinery of mim-width. We also note that the mim-width algorithm, as presented in [20], requires a running time in excess of $\Omega(n^{300})$. Hence our algorithm is better in this regard as well.

5 Subclasses of Bipartite Graphs

It is known that there exist bipartite graphs G for which $\chi^*_{ON}(G) = \Theta(\sqrt{n})$, where n is the number of vertices of G. Abel et al. [1] showed that it is NP-complete to decide if k colors are sufficient to CFON* color a planar bipartite graph even when $k \in \{1, 2, 3\}$. This implies that CFON* coloring is NP-hard on bipartite graphs as well. In this section, we study CFON* coloring on some subclasses of bipartite graphs namely biconvex graphs and bipartite permutation graphs. We show that CFON* coloring is polynomial time solvable on these classes.

We first define biconvex graphs, followed Lemma 26 by a bound on the CFON* chromatic number. The proof of Lemma 26 is omitted.

▶ **Definition 25** (Biconvex Graph). *We say that an ordering σ of X in a bipartite graph $B = (X, Y, E)$ satisfies the* adjacency property *if for every vertex $y \in Y$, the neighborhood $N(y)$ is a set of vertices that are consecutive in the ordering σ of X. A bipartite graph (X, Y, E) is* biconvex *if there are orderings of X (with respect to Y) and Y (with respect to X) that fulfill the adjacency property.*

▶ **Lemma 26** (⋆). *If G is a biconvex graph, then $\chi^*_{ON}(G) \leq 3$.*

▶ **Theorem 27.** *The problem of determining the CFON* chromatic number of a given biconvex graph is solvable in polynomial time.*

Proof. Given a biconvex graph G, we show that $\chi^*_{ON}(G) \leq 3$. We use the fact that every induced subgraph of a biconvex graph admits multi-chain ordering [7, 12]. Let $G = (V, E)$ be a biconvex graph and let $V_0, V_1, \dots, V_q$ be a partition of vertices $V(G)$ respecting the multi-chain ordering conditions. Similar to interval graphs, we now characterize graphs that require one color and two colors. Note that the algorithms in Theorems 21 and 23 work for biconvex graphs too as the proof is based on the multi-chain ordering property and biconvex bipartite graphs admit multi-chain ordering property. In fact, the proof is a bit simpler because of the fact that each V_i is an independent set and we do not need to take care of the edges within a part V_i, as in the case of interval graphs. ◀

The class of bipartite permutation graphs [7] are a subclass of biconvex, and also admit multi-chain ordering property. Hence it follows from Theorem 27 that the problem is polynomial time solvable on bipartite permutation graphs.

▶ **Corollary 28.** *The problem of determining the CFON* chromatic number of a given bipartite permutation graph is solvable in polynomial time.*

6 Conclusion

In this paper, we study CFON* coloring on claw-free graphs, interval graphs and biconvex graphs.

We first show that if G is a $K_{1,k}$-free graph with maximum degree Δ, then $\chi^*_{ON}(G) = O(k^2 \log \Delta)$. We then show that if the minimum degree of G is $\Omega(\frac{\Delta}{\log^\epsilon \Delta})$ for some $\epsilon \geq 0$, then $\chi^*_{ON}(G) = O(\log^{1+\epsilon} \Delta)$. The tightness of these bounds is a natural open question.

We show that CFON* coloring is polynomial time solvable on interval graphs and biconvex graphs, critically using the fact that they admit multi-chain ordering property. Using a similar approach, it can be shown that the full coloring variant of the problem (i.e., CFON coloring) is polynomial time solvable on these graph classes. It is known that CFON* coloring is NP-hard on planar bipartite graphs and there exist bipartite graphs on n vertices that requires $\Theta(\sqrt{n})$ colors. It may be of interest to study the problem on other subclasses of bipartite graphs, such as convex bipartite graphs, chordal bipartite graphs and tree-convex bipartite graphs.

References

1 Zachary. Abel, Victor. Alvarez, Erik D. Demaine, Sándor P. Fekete, Aman. Gour, Adam. Hesterberg, Phillip. Keldenich, and Christian. Scheffer. Conflict-free coloring of graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2675–2702, 2018. doi:10.1137/17M1146579.

2 Akanksha Agrawal, Pradeesha Ashok, Meghana M. Reddy, Saket Saurabh, and Dolly Yadav. FPT algorithms for conflict-free coloring of graphs and chromatic terrain guarding. *CoRR*, abs/1905.01822, 2019. arXiv:1905.01822.

3 Amotz Bar-Noy, Panagiotis Cheilaris, Svetlana Olonetsky, and Shakhar Smorodinsky. Online conflict-free colorings for hypergraphs. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *Automata, Languages and Programming*, pages 219–230, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

4 Sriram Bhyravarapu, Tim A. Hartmann, Subrahmanyam Kalyanasundaram, and I. Vinod Reddy. Conflict-free coloring: Graphs of bounded clique width and intersection graphs. In *Combinatorial Algorithms - 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021, Proceedings*, pages 92–106, 2021. doi:10.1007/978-3-030-79987-8_7.

5 Sriram Bhyravarapu, Subrahmanyam Kalyanasundaram, and Rogers Mathew. A short note on conflict-free coloring on closed neighborhoods of bounded degree graphs. *J. Graph Theory*, 97(4):553–556, 2021. doi:10.1002/jgt.22670.

6 Hans L. Bodlaender, Sudeshna Kolay, and Astrid Pieterse. Parameterized complexity of conflict-free graph coloring. *CoRR*, abs/1905.00305, 2019. arXiv:1905.00305.

7 Andreas Brandstädt and Vadim V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Comb.*, 67, 2003.

8 Panagiotis Cheilaris. *Conflict-free Coloring.* PhD thesis, City University of New York, New York, NY, USA, 2009.

9 Ke Chen, Amos Fiat, Haim Kaplan, Meital Levy, Jiří Matoušek, Elchanan Mossel, János Pach, Micha Sharir, Shakhar Smorodinsky, Uli Wagner, and Emo Welzl. Online conflict-free coloring for intervals. *SIAM J. Comput.*, 36(5):1342–1359, December 2006.

10 Michał Dębski and Jakub Przybyło. Conflict-free chromatic number versus conflict-free chromatic index. *Journal of Graph Theory*, 2021. doi:10.1002/jgt.22743.

11 Reinhard Diestel. Graph theory 5th ed. *Graduate texts in mathematics*, 173, 2017.

12 Jessica A. Enright, Lorna Stewart, and Gábor Tardos. On list coloring and list homomorphism of permutation and interval graphs. *SIAM J. Discret. Math.*, 28(4):1675–1685, 2014. doi:10.1137/13090465X.

13 P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets*, 10:609–627, 1975.

14 Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, January 2004.

15 Sándor P Fekete, Stephan Friedrichs, Michael Hemmer, Joseph BM Mitchell, and Christiane Schmidt. On the chromatic art gallery problem. In *CCCG*, 2014.

16 Sándor P. Fekete and Phillip Keldenich. Conflict-free coloring of intersection graphs. *International Journal of Computational Geometry & Applications*, 28(03):289–307, 2018.

17 Luisa Gargano and Adele Rescigno. Collision-free path coloring with application to minimum-delay gathering in sensor networks. *Discrete Applied Mathematics*, 157:1858–1872, April 2009. doi:10.1016/j.dam.2009.01.015.

18 Luisa Gargano and Adele A. Rescigno. Complexity of conflict-free colorings of graphs. *Theor. Comput. Sci.*, 566(C):39–49, February 2015. doi:10.1016/j.tcs.2014.11.029.

19 Roman Glebov, Tibor Szabó, and Gábor Tardos. Conflict-free colouring of graphs. *Combinatorics, Probability and Computing*, 23(3):434–448, 2014.

20 Carolina Lucía Gonzalez and Felix Mann. On d-stable locally checkable problems on bounded mim-width graphs. *CoRR*, abs/2203.15724, 2022. doi:10.48550/arXiv.2203.15724.

21 Frank Hoffmann, Klaus Kriegel, Subhash Suri, Kevin Verbeek, and Max Willert. Tight bounds for conflict-free chromatic guarding of orthogonal art galleries. *Computational Geometry*, 73:24–34, 2018.

22 Chaya Keller and Shakhar Smorodinsky. Conflict-free coloring of intersection graphs of geometric objects. In *SODA*, 2017.

23 Prasad Krishnan, Rogers Mathew, and Subrahmanyam Kalyanasundaram. Pliable index coding via conflict-free colorings of hypergraphs. In *IEEE International Symposium on Information Theory, ISIT 2021, Melbourne, Australia, July 12-20, 2021*, pages 214–219. IEEE, 2021. doi:10.1109/ISIT45174.2021.9518120.

24 M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge Univ Pr, 2005.

25 Vinodh P Vijayan and E. Gopinathan. Design of collision-free nearest neighbor assertion and load balancing in sensor network system. *Procedia Computer Science*, 70:508–514, December 2015. `doi:10.1016/j.procs.2015.10.092`.

26 Janos Pach and Gábor Tardos. Conflict-free colourings of graphs and hypergraphs. *Combinatorics, Probability and Computing*, 18(5):819–834, 2009.

27 I. Vinod Reddy. Parameterized algorithms for conflict-free colorings of graphs. *Theor. Comput. Sci.*, 745:53–62, 2018. `doi:10.1016/j.tcs.2018.05.025`.

28 Shakhar Smorodinsky. *Conflict-Free Coloring and its Applications*, pages 331–389. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

Skolem Meets Schanuel

Yuri Bilu
Institut de Mathématiques de Bordeaux, Université de Bordeaux and CNRS, Talence, France

Florian Luca
School of Mathematics, University of the Witwatersrand, Johannesburg, South Africa
Research Group in Algebraic Structures & Applications, King Abdulaziz University, Saudi Arabia
Centro de Ciencias Matemáticas UNAM, Morelia, Mexico
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

Joris Nieuwveld
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

David Purser
University of Warsaw, Poland

James Worrell
Department of Computer Science, University of Oxford, UK

Abstract

The celebrated Skolem-Mahler-Lech Theorem states that the set of zeros of a linear recurrence sequence is the union of a finite set and finitely many arithmetic progressions. The corresponding computational question, the Skolem Problem, asks to determine whether a given linear recurrence sequence has a zero term. Although the Skolem-Mahler-Lech Theorem is almost 90 years old, decidability of the Skolem Problem remains open. The main contribution of this paper is an algorithm to solve the Skolem Problem for simple linear recurrence sequences (those with simple characteristic roots). Whenever the algorithm terminates, it produces a stand-alone certificate that its output is correct – a set of zeros together with a collection of witnesses that no further zeros exist. We give a proof that the algorithm always terminates assuming two classical number-theoretic conjectures: the Skolem Conjecture (also known as the Exponential Local-Global Principle) and the p-adic Schanuel Conjecture. Preliminary experiments with an implementation of this algorithm within the tool Skolem point to the practical applicability of this method.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Skolem Problem, Skolem Conjecture, Exponential Local-Global Principle, p-adic Schanuel Conjecture

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.20

Supplementary Material *Software*: `https://skolem.mpi-sws.org`

Funding *Yuri Bilu*: Supported by ANR project JINVARIANT, by the Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany, and by the Max Planck Institute for Mathematics, Bonn, Germany.
Joël Ouaknine: Also affiliated with Keble College, Oxford as `emmy.network` Fellow, and supported by DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 20; pp. 20:1–20:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 The Skolem Problem

A (rational) linear recurrence sequence (LRS) $\boldsymbol{u} = \langle u_n \rangle_{n=0}^{\infty}$ is a sequence of rational numbers satisfying the equation

$$u_{n+d} = c_1 u_{n+d-1} + \cdots + c_{d-1} u_{n+1} + c_d u_n \qquad (1)$$

for all $n \in \mathbb{N}$, where the coefficients $c_1, \ldots, c_d$ are rational numbers and $c_d \neq 0$. We say that the above recurrence has *order d*. We moreover say that an LRS is *simple* if the characteristic polynomial of its minimal-order recurrence has simple roots.

The celebrated theorem of Skolem, Mahler, and Lech (see [15]) describes the structure of the set $\{n \in \mathbb{N} : u_n = 0\}$ of zero terms of an LRS as follows:

▶ **Theorem 1.** *Given a linear recurrence sequence $\boldsymbol{u} = \langle u_n \rangle_{n=0}^{\infty}$, the set of zero terms is a union of finitely many arithmetic progressions, together with a finite set.*

The statement of Theorem 1 can be refined by considering the notion of *non-degeneracy* of an LRS. An LRS is non-degenerate if in its minimal recurrence the quotient of no two distinct roots of the characteristic polynomial is a root of unity. A given LRS can be effectively decomposed as an interleaving of finitely many non-degenerate sequences, some of which may be identically zero. The core of the Skolem-Mahler-Lech Theorem is the fact that a non-zero non-degenerate linear recurrence sequence has finitely many zero terms. Unfortunately, all known proofs of this last assertion are ineffective: it is not known how to compute the finite set of zeros of a given non-degenerate linear recurrence sequence. It is readily seen that the existence of a procedure to do so is equivalent to the existence of a procedure to decide whether an arbitrary given LRS has a zero term. The problem of deciding whether an LRS has a zero term is variously known as the Skolem Problem or the Skolem-Pisot Problem.

Decidability of the Skolem Problem is known only for certain special cases, based on the relative order of the absolute values of the characteristic roots. Say that a characteristic root λ is *dominant* if its absolute value is maximal among all the characteristic roots. Decidability is known in case there are at most 3 dominant characteristic roots, and also for recurrences of order at most 4 [26, 34]. However for LRS of order 5 it is not currently known how to decide the Skolem Problem.

The Skolem Problem, along with closely related questions such as the Positivity Problem, is intimately connected to various fundamental topics in program analysis and automated verification, such as the termination and model checking of simple while loops [3, 18, 27] or the algorithmic analysis of stochastic systems [1, 2, 5, 13, 28]. It also appears in a variety of other contexts, such as formal power series [29, 33] and control theory [9, 16]. The Skolem Problem is often used as a reference to establish hardness of other open decision problems; in addition to some of the previously cited papers, the articles [4, 14], for example, specifically invoke hardness of the Skolem Problem for simple LRS of order 5. Thus far, the only known complexity bound for the Skolem Problem is NP-hardness [10].

1.2 The Skolem Conjecture and the Bi-Skolem Problem

The notion of linear recurrence equally well makes sense for a bi-infinite sequence $\boldsymbol{u} = \langle u_n \rangle_{n=-\infty}^{\infty}$ of rational numbers: one defines $\boldsymbol{u}$ to be a *linear recurrent bi-sequence (LRBS)* if it satisfies the recurrence (1) for all $n \in \mathbb{Z}$. Note that every LRS $\boldsymbol{u}$ extends uniquely to an LRBS satisfying the same recurrence (one obtains such an extension by "running the recurrence backwards"). The notions of simplicity and non-degeneracy carry over in the obvious way to LRBS. We remark also that the Skolem-Mahler-Lech Theorem remains valid for LRBS – a non-degenerate LRBS has finitely many zeros. The analog of the Skolem Problem for LRBS is the *Bi-Skolem Problem*, which asks, for a given LRBS $\boldsymbol{u} = \langle u_n \rangle_{n=-\infty}^{\infty}$, whether there exists $n \in \mathbb{Z}$ with $u_n = 0$.

A major motivation to consider the Bi-Skolem Problem is the existence of the *Exponential Local-Global Principle*, a conjecture introduced by Thoralf Skolem in 1937 [32]. To formulate the conjecture we first make some observations about the value set of an LRBS. Given a non-zero integer b, let $\mathbb{Z}[\frac{1}{b}]$ be the subring of $\mathbb{Q}$ obtained by adjoining $\frac{1}{b}$ to $\mathbb{Z}$. We note that every rational LRBS takes values in $\mathbb{Z}[\frac{1}{b}]$ for some b. Indeed, if $\boldsymbol{u} = \langle u_n \rangle_{n=-\infty}^{\infty}$ satisfies recurrence (1) and $\mathbb{Z}[\frac{1}{b}]$ contains the coefficients $c_1, \ldots, c_d$, the reciprocal c_d^{-1} of the last coefficient, and the initial terms $u_0, \ldots, u_{d-1}$, then by running the recurrence forwards and backwards from the initial terms we see that $u_n \in \mathbb{Z}[\frac{1}{b}]$ for all $n \in \mathbb{Z}$.

▶ **Skolem Conjecture.** *Let $\boldsymbol{u}$ be a simple rational LRBS taking values in the ring $\mathbb{Z}[\frac{1}{b}]$ for some integer b. Then $\boldsymbol{u}$ has no zero iff, for some integer $m \geq 2$ with $\gcd(b, m) = 1$, we have that $u_n \not\equiv 0 \bmod m$ for all $n \in \mathbb{Z}$.*

In other words, the Skolem Conjecture asserts that if a simple LRBS fails to have a zero, then this is witnessed modulo m for some m. The truth of this conjecture immediately entails the existence of an algorithm to solve the Bi-Skolem Problem for simple LRBS: simply search in parallel either for a zero of the LRBS, or for a number m substantiating the absence of zeros. If the Skolem Conjecture holds, then the search must necessarily eventually terminate.

There exists a substantial body of literature on the Skolem Conjecture, including proofs of a variety of special cases. In particular, the Skolem Conjecture has been shown to hold for simple LRBS of order 2 [6], and for certain families of LRBS of order 3 [30, 31]. In a different but related vein, Bertók and Hajdu have shown that, in some sense, the Skolem Conjecture is valid in "almost all" instances [7, 8].

1.3 Main Results

It is immediate that the Bi-Skolem Problem reduces to the Skolem Problem: an LRBS $\langle u_n \rangle_{n=-\infty}^{\infty}$ has a zero term if and only if at least one of the one-way infinite sequences $\langle u_n \rangle_{n=0}^{\infty}$ and $\langle u_{-n} \rangle_{n=0}^{\infty}$, both of which are LRS, has a zero term. However it is open whether there is a reduction in the other direction (equivalently, it is open whether an oracle for the Bi-Skolem Problem can be used to determine *all* the zeros of a non-degenerate LRBS). Indeed, an oracle for the Bi-Skolem Problem would appear to be of little utility in deciding the Skolem Problem for an LRS whose bi-completion happens to harbour a zero at a negative index. It is likewise not known (in spite of the similar nomenclature) whether the truth of the Skolem Conjecture implies decidability of the Skolem Problem.

Our first main result is as follows:

▶ **Theorem 2.** *The Skolem Problem reduces to the Bi-Skolem Problem subject to the weak p-adic Schanuel Conjecture.*

Schanuel's Conjecture [21, Pages 30-31] is a unifying conjecture in transcendental number theory that plays a key role in the study of the exponential function over both the real and complex numbers. In particular, a celebrated paper of Macintyre and Wilkie [23] obtains decidability of the first-order theory of the structure $(\mathbb{R}; <, \cdot, +, \exp)$ assuming Schanuel's Conjecture over $\mathbb{R}$. A p-adic version of the Schanuel Conjecture, referring to the exponential function on the ring $\mathbb{Z}_p$ of p-adic integers, was formulated in [12]. This conjecture was shown in [24] to imply decidability of the first-order theory of the structure $(\mathbb{Z}_p; <, \cdot, +, \exp)$.

Since the reduction in Theorem 2 specialises to simple LRBS we obtain:

▶ **Theorem 3.** *The Skolem Problem for simple LRS is decidable subject to the weak p-adic Schanuel Conjecture and the Skolem Conjecture.*

The proof of Theorem 3 gives an algorithm that computes the set of zeros of a non-degenerate simple LRBS. The algorithm moreover produces an unconditional certificate that its output is correct, i.e., that all zeros have been found. This certificate consists of a partition of the input LRBS into finitely many subsequences such that each subsequence contains at most one zero. For a subsequence with no zero, the algorithm finds an integer m such that the subsequence is non-zero modulo m; for a subsequence with a zero, the algorithm provides a prime p such that p divides the non-zero terms a well-described (upper-bounded) number of times. The conjectural aspect of Theorem 3 solely concerns the proof that the algorithm terminates on all input sequences.

We have implemented our algorithm within the Skolem tool,[1] which enumerates the set of zeros of a given non-degenerate simple LRS, and produces an independent (conjecture-free) certificate that all zeros have been found. Preliminary experiments, which we present in Section 5, point to the practical applicability of our algorithm.

1.4 Related Work

The decidability of the Skolem Problem is generally considered to have been open since the early 1930s, as the p-adic techniques underpinning the Skolem-Mahler-Lech Theorem were well understood already at the time not to be effective. As noted earlier, a breakthrough establishing decidability at order 4 occurred in the mid-1980s [26, 34], making key use of Baker's Theorem on linear forms in logarithms of algebraic numbers. Very recently, we have shown that the Skolem Problem is decidable at order 5 assuming only the Skolem Conjecture; and in the same paper we also obtained unconditional decidability for reversible LRS[2] of order 7 or less [22]. A minor contribution of the present paper is to improve on the former result by establishing a Turing reduction from the Skolem Problem at order 5 to the Bi-Skolem Problem for simple LRBS of order 5; this is the content of Theorem 12.

2 Technical Background

2.1 Computation in Number Fields

A number field $\mathbb{K}$ is a finite-degree extension of $\mathbb{Q}$. For computational purposes, such a field can be represented in the form $\mathbb{Q}[X]/(g(X))$, where $g(X)$ is the minimal polynomial of a primitive element of $\mathbb{K}$. With such a representation it is straightforward to do arithmetic in $\mathbb{K}$, including solving systems of linear equations with coefficients in $\mathbb{K}$. Moreover, given a polynomial $f(X) \in \mathbb{Q}[X]$, one can compute a representation in the above form of the splitting field $\mathbb{K}$ of f over $\mathbb{Q}$, together with representations of the roots of f as elements of $\mathbb{K}$ [20].

In addition to basic arithmetic and linear algebra in $\mathbb{K}$, we wish to determine whether some given elements $\lambda_1, \dots, \lambda_s \in \mathbb{K}$ are multiplicatively independent and, if not, to exhibit $a_1, \dots, a_s \in \mathbb{Z}$ such that $\lambda_1^{a_1} \cdots \lambda_s^{a_s} = 1$. For this we can use the following result, which shows that if such a multiplicative relation exists then there exists one in which the exponents $a_1, \dots, a_s$ have absolute value at most B for some bound B computable from the height of the λ_i and the degree of the number field $\mathbb{K}$.

[1] Skolem can be experimented with online at `https://skolem.mpi-sws.org/`.

[2] An integer LRS is *reversible* if its completion as an LRBS only takes on integer values.

▶ **Theorem 4** (Masser [25]). *Let $\mathbb{K}$ be a number field of degree D over $\mathbb{Q}$. For $s \geq 1$ let $\lambda_1, \ldots, \lambda_s$ be non-zero elements of $\mathbb{K}$ having absolute logarithmic Weil height at most h over $\mathbb{Q}$. Then the group of multiplicative relations*

$$L = \{(k_1, \ldots, k_s) \in \mathbb{Z}^s : \lambda_1^{k_1} \cdots \lambda_s^{k_s} = 1\} \tag{2}$$

is generated (as an additive subgroup of $\mathbb{Z}^s$) by a collection of vectors all of whose entries have absolute value at most $(csh)^{s-1}D^{s-1}\dfrac{(\log(D+2))^{3s-3}}{(\log\log(D+2))^{3s-4}}$, for some absolute constant c.

2.2 p-adic Numbers

Let p be a prime. Define the *p-adic valuation* $v_p : \mathbb{Q} \to \mathbb{Z} \cup \{\infty\}$ by $v_p(0) = \infty$ and $v_p\left(p^\nu \cdot \frac{a}{b}\right) = \nu$ for all $a, b \in \mathbb{Z} \setminus \{0\}$ such that $\gcd(ab, p) = 1$. In other words, $v_p(x)$ gives the exponent of p as a divisor of $x \in \mathbb{Q}$. The map v_p determines an absolute value $|\cdot|_p$ on $\mathbb{Q}$, where $|x|_p := p^{-v_p(x)}$ (with the convention that $|0|_p = p^{-\infty} = 0$). Due to the fact that $v_p(a+b) \geq \min(v_p(a), v_p(b))$, we have the strong triangle equality: $|a+b|_p \leq \max(|a|_p, |b|_p)$ for all $a, b \in \mathbb{Q}$. In other words, $|\cdot|_p$ is a *non-Archimedean* absolute value. The field $\mathbb{Q}_p$ of *p-adic numbers* is the completion of $\mathbb{Q}$ with respect to $|\cdot|_p$. The absolute value $|\cdot|_p$ extends to a non-Archimedean absolute value on $\mathbb{Q}_p$. The ring of *p-adic integers* is $\mathbb{Z}_p := \{x \in \mathbb{Q}_p : |x|_p \leq 1\}$. The ring $\mathbb{Z}_p$ contains a unique maximal ideal $p\mathbb{Z}_p$, with the quotient $\mathbb{Z}_p/p\mathbb{Z}_p$ being isomorphic to $\mathbb{F}_p$ (the finite field with p elements). When we refer to elements of $\mathbb{Z}_p$ modulo p we refer to their image under this quotient map.

Given a sequence of numbers $\langle a_n \rangle_{n=0}^\infty$ in $\mathbb{Z}_p$, the infinite sum $\sum_{n=0}^\infty a_n$ converges to an element of $\mathbb{Z}_p$ if and only if $|a_n|_p \to 0$ (equivalently, $v_p(a_n) \to \infty$) as $n \to \infty$. It follows that given a sequence $\langle a_n \rangle_{n=0}^\infty$ in $\mathbb{Z}_p$ with $|a_n|_p \to 0$, the corresponding power series $f(X) = \sum_{j=0}^\infty a_j X^j$ defines a function $f : \mathbb{Z}_p \to \mathbb{Z}_p$.

Consider a monic polynomial $g(X) \in \mathbb{Z}[X]$ with non-zero discriminant $\Delta(g)$. Let p be a prime that does not divide $\Delta(g)$. Denote by $\overline{g}(X) \in \mathbb{F}_p[X]$ the polynomial obtained from g by replacing each coefficient with its residue modulo p. It is well known that a sufficient condition for g to split completely over $\mathbb{Z}_p$ is that $\overline{g}$ split over $\mathbb{F}_p$. Indeed, in this situation one can use Hensel's Lemma [17, Theorem 3.4.1] to "lift" each of the roots of $\overline{g}$ in $\mathbb{F}_p$ to a distinct root in $\mathbb{Z}_p$. Moreover, by the Chebotarev density theorem [19] there are infinitely many primes p for which $\overline{g}$ splits over $\mathbb{F}_p$. Hence there are infinitely many primes p such that g splits over $\mathbb{Z}_p$. Note that the last statement holds even without the assumption that $\Delta(g) \neq 0$, since $g \in \mathbb{Z}[X]$ splits over $\mathbb{Z}_p$ whenever $\frac{g}{\gcd(g,g')} \in \mathbb{Z}[X]$ splits over $\mathbb{Z}_p$ (and the latter has non-zero discriminant).

Let p be an odd prime.[3] The *p-adic exponential* is defined as $\exp(x) = \displaystyle\sum_{k=0}^\infty \frac{x^k}{k!}$, which converges for all $x \in p\mathbb{Z}_p$. The *p-adic logarithm* is defined as $\log(x) = \displaystyle\sum_{k=0}^\infty (-1)^{k+1}\frac{(x-1)^k}{k}$, which converges for all $x \in 1 + p\mathbb{Z}_p$. For $x, y \in p\mathbb{Z}_p$ we have $\exp(x+y) = \exp(x)\exp(y)$ and for $x, y \in 1 + p\mathbb{Z}_p$ we have $\log(xy) = \log(x) + \log(y)$. Indeed we have that exp and log yield mutually inverse isomorphisms between the additive group $p\mathbb{Z}_p$ and multiplicative group $1 + p\mathbb{Z}_p$.

[3] We omit the prime $p = 2$ to avoid special cases in the facts below.

Schanuel's Conjecture for the complex numbers is a powerful unifying principle in transcendence theory. We will need the following p-adic version of the weak form of Schanuel's Conjecture, which can be found, e.g., as [12, Conjecture 3.10].

▶ **Conjecture 5** (Weak p-adic Schanuel Conjecture)**.** *Let $\alpha_1, \dots, \alpha_s \in 1+p\mathbb{Z}_p$ be algebraic over $\mathbb{Q}$ and such that $\log \alpha_1, \dots, \log \alpha_s$ are linearly independent over $\mathbb{Q}$. Then $\log \alpha_1, \dots, \log \alpha_s$ are algebraically independent over $\mathbb{Q}$, i.e., for every non-zero polynomial $P \in \mathbb{Q}_p[X_1, \dots, X_s]$ whose coefficients are algebraic over $\mathbb{Q}$, we have that $P(\log \alpha_1, \dots, \log \alpha_s) \neq 0$.*

A known special case of Conjecture 5 is the following result of Brumer [11], which is a p-adic analog of Baker's Theorem on linear independence of logarithms of algebraic numbers.

▶ **Theorem 6.** *Let $\alpha_1, \dots, \alpha_s \in 1+p\mathbb{Z}_p$ be algebraic over $\mathbb{Q}$ and such that $\log \alpha_1, \dots, \log \alpha_s$ are linearly independent over $\mathbb{Q}$. Then $\beta_0 + \beta_1 \log \alpha_1 + \cdots + \beta_s \log \alpha_s \neq 0$ for all $\beta_0, \dots, \beta_s \in \mathbb{Q}_p$ that are algebraic over $\mathbb{Q}$ and not all zero.*

3 p-adic Power-Series Representation of LRBS

Let $\boldsymbol{u} = \langle u_n \rangle_{n=-\infty}^{\infty}$ be an LRBS of rational numbers satisfying the linear recurrence

$$u_{n+d} = c_1 u_{n-d-1} + \cdots + c_d u_n \quad (n \in \mathbb{Z}),$$

where $c_d \neq 0$. For the purposes of computing the zeros of $\boldsymbol{u}$ we can assume without loss of generality that the coefficients $c_1, \dots, c_d$ of the recurrence are integers. (It is easy to see that for any integer ℓ such that $\ell c_i \in \mathbb{Z}$ for $i \in \{1, \dots, d\}$, the scaled sequence $\langle \ell^n u_n \rangle_{n=-\infty}^{\infty}$ satisfies an integer recurrence.) Write $g(X) := X^d - c_1 X^{d-1} - \cdots - c_d$ for the *characteristic polynomial* of the recurrence and let

$$A := \begin{pmatrix} c_1 & \cdots & c_{d-1} & c_d \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix}, \quad \alpha := \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}, \quad \text{and} \quad \beta := \begin{pmatrix} u_{d-1} \\ \vdots \\ u_1 \\ u_0 \end{pmatrix}.$$

We now have the matrix-exponential representation $u_n = \alpha A^n \beta$ for all $n \in \mathbb{Z}$. (Here A is known as the *companion matrix* of the recurrence.)

The key tool in our approach – which also underlies the proof of the Skolem-Mahler-Lech Theorem – is the representation of the LRBS $\boldsymbol{u}$ in terms of a power series $f(X) = \sum_{j=0}^{\infty} a_j X^j$ with coefficients in $\mathbb{Z}_p$. In defining f we work with an odd prime p such that (i) p does not divide the constant term c_d of the recurrence; (ii) p does not divide the discriminant $\Delta\left(\frac{g}{\gcd(g,g')}\right)$; (iii) the characteristic polynomial g splits over $\mathbb{Z}_p$. As explained in Section 2.2, there are infinitely many such primes. Moreover, for a particular prime p that does not divide $\Delta\left(\frac{g}{\gcd(g,g')}\right)$, we can easily verify whether g splits over $\mathbb{Z}_p$, since this is equivalent to $\frac{g}{\gcd(g,g')}$ splitting over $\mathbb{F}_p$.

Write $\lambda_1, \dots, \lambda_s \in \mathbb{Z}_p$ for the distinct roots of g. Let $\mathbb{K}$ be the subfield of $\mathbb{Q}_p$ generated by $\lambda_1, \dots, \lambda_s$. Then $\mathbb{K}$ is a number field and thus we can compute symbolically in $\mathbb{K}$ as described in Section 2.1. It is well known that the sequence $\boldsymbol{u}$ admits an exponential-polynomial representation

$$u_n = \sum_{i=1}^{s} Q_i(n) \lambda_i^n \quad (n \in \mathbb{Z}), \tag{3}$$

where $Q_i \in \mathbb{K}[X]$ has degree strictly less than the multiplicity of λ_i as a root of g. The coefficients of each polynomial Q_i can be computed as the unique solution of the system of linear equations that arises by substituting $n = 0, \dots, d-1$ in Equation (3) (where, recall, d is the order of the recurrence).

The companion matrix has determinant $\det(A) = \pm c_d$, which is non-zero modulo p; hence A is invertible modulo p. Let L be the least positive integer such that $A^L \equiv I \bmod p$. Being an eigenvalue of A^L, $\lambda_i^L \equiv 1 \bmod p$ for all $i \in \{1, \dots, s\}$ and hence the p-adic logarithm $\log \lambda_i^L$ is defined for all $i \in \{1, \dots, s\}$. We thus obtain the following representation of the subsequence $\langle u_{Ln} \rangle_{n=-\infty}^{\infty}$ in terms of the p-adic exponential and logarithm functions:

$$u_{Ln} = \sum_{i=1}^{s} Q_i(Ln)\lambda_i^{Ln} = \sum_{i=1}^{s} Q_i(Ln)\exp(n \log \lambda_i^L)\,.$$

Now consider the power series $f(X) = \sum_{j=0}^{\infty} a_j X^j$ such that

$$f(x) := \sum_{i=1}^{s} Q_i(Lx)\exp(x \log \lambda_i^L) \tag{4}$$

for all $x \in \mathbb{Z}_p$. Then we have $u_{Ln} = f(n)$ for all $n \in \mathbb{Z}$. Moreover, since $a_j = \frac{1}{j!} f^{(j)}(0)$, by taking derivatives in (4) we obtain the following expression for the coefficients of f:

$$a_j = \frac{1}{j!} \sum_{i=1}^{s} \sum_{k=0}^{j} \binom{j}{k} L^k Q_i^{(k)}(0) \left(\log \lambda_i^L\right)^{j-k}\,. \tag{5}$$

In the remainder of this section we recall an alternative formula for the coefficients of f as p-adically convergent sums of rational numbers. This provides a simple method to compute $v_p(a_j)$ that avoids computing p-adic approximations of the characteristic roots, as would be needed if we were to directly use (5).

Recall that we have $A^L \equiv I \mod p$. Let us say that $A^L = I + pB$ for some integer matrix B. Then we have:

$$\begin{aligned}
u_{Ln} &= \alpha A^{Ln} \beta \\
&= \alpha (I + pB)^n \beta \\
&= \sum_{k=0}^{n} \binom{n}{k} p^k \alpha B^k \beta \\
&= \sum_{k=0}^{n} \frac{n(n-1)\dots(n-k+1)}{k!} p^k \alpha B^k \beta \\
&= \sum_{k=0}^{\infty} \frac{n(n-1)\dots(n-k+1)}{k!} p^k \alpha B^k \beta \\
&= \sum_{k=0}^{\infty} \sum_{j=0}^{\infty} c_{k,j} n^j \frac{p^k}{k!} \quad \text{for certain } c_{k,j} \in \mathbb{Z} \text{ with } c_{k,j} = 0 \text{ for } j > k \\
&= \sum_{j=0}^{\infty} \sum_{k=j}^{\infty} c_{k,j} n^j \frac{p^k}{k!}\,.
\end{aligned}$$

It remains to see why one can reverse the order of summation in the last line above and why the resulting sums converge in $\mathbb{Z}_p$. For this we can apply [17, Proposition 4.1.4], for which we require that the summand $c_{k,j} n^j \frac{p^k}{k!}$ converge to 0 as $j \to \infty$ and converge to 0 uniformly in j as $k \to \infty$. But this precondition follows from the fact that $v_p(k!) < \frac{k}{p-1}$, from which we have $v_p\left(c_{k,j} n^j \frac{p^k}{k!}\right) \geq \frac{(p-2)k}{p-1}$ for all $k \geq j$.

Now consider the power series $\widetilde{f}(X) := \sum_{j=0}^{\infty} b_j X^j$ where

$$b_j := \sum_{k=j}^{\infty} c_{k,j} \frac{p^k}{k!} \in \mathbb{Z}_p \,. \tag{6}$$

By the above considerations we have that $v_p(b_j) \geq \frac{(p-2)j}{p-1}$ and hence $\widetilde{f}$ converges on $\mathbb{Z}_p$ and satisfies $\widetilde{f}(n) = u_{Ln}$ for all $n \in \mathbb{Z}$. In particular, the power series f and $\widetilde{f}$ agree on $\mathbb{Z}$ and hence (e.g., by [17, Proposition 4.4.3]) are identical, i.e., $a_j = b_j$ for all $j \in \mathbb{N}$. Thus we can use Equation (6) to exactly compute $v_p(a_j)$ for any j such that $a_j \neq 0$.

4 Computing all the Zeros of an LRBS

In this section we show, assuming the weak p-adic Schanuel Conjecture, that the set of all zeros of a non-degenerate LRBS is computable using an oracle for the Bi-Skolem Problem. In particular, this gives a Turing reduction of the Skolem Problem to the Bi-Skolem Problem.

▶ **Proposition 7.** *Let $f : \mathbb{Z}_p \to \mathbb{Z}_p$ be given by a convergent p-adic power series $f(X) = \sum_{k=0}^{\infty} a_k X^k$, with coefficients in $\mathbb{Z}_p$. Suppose that ℓ is a positive integer such that $a_0 = \cdots = a_{\ell-1} = 0$ and $a_\ell \neq 0$. Then, writing $\nu := v_p(a_\ell)$, we have $f(p^{\nu+1}x) \neq 0$ for all non-zero $x \in \mathbb{Z}_p$.*

Proof. Let $x \in \mathbb{Z}_p$ be non-zero. For every $m > \ell$ we have

$$\begin{aligned} v_p(a_\ell(p^{\nu+1}x)^\ell) &= \nu + \ell(\nu+1) + v_p(x^\ell) \\ &< m(\nu+1) + v_p(x^m) \quad (\text{since } \ell < m \text{ and } x \neq 0) \\ &\leq v_p(a_m(p^{\nu+1}x)^m) \,. \end{aligned}$$

It follows that for all $m \geq \ell$,

$$v_p\left(\sum_{k=0}^{m} a_k (p^{\nu+1}x)^k\right) = v_p(a_\ell(p^{\nu+1}x)^\ell) \,.$$

Letting m tend to infinity, we have $v_p(f(p^{\nu+1}x)) = v_p(a_\ell(p^{\nu+1}x)^\ell) < \infty$ and we conclude that $f(p^{\nu+1}x) \neq 0$. ◀

▶ **Proposition 8.** *Let $\boldsymbol{u} = \langle u_n \rangle_{n=-\infty}^{\infty}$ be a non-zero LRBS consisting of rational numbers. Assuming the weak p-adic Schanuel Conjecture, one can compute a positive integer M such that $u_{Mn} \neq 0$ for all $n \in \mathbb{Z} \setminus \{0\}$.*

Proof. As explained in Section 3, there exists a prime p and a positive integer L such that $u_{Ln} = f(n)$ for all $n \in \mathbb{Z}$, for the p-adic power series $f(X) = \sum_{j=0}^{\infty} a_j X^j$ whose coefficients are given by the formula (4). Recall that in this formula the λ_i are the characteristic roots of $\boldsymbol{u}$ and the Q_i are the coefficients appearing in the exponential polynomial formula (3).

Pick a maximal multiplicatively independent subset of characteristic roots. Without loss of generality we can write this set as $\{\lambda_1, \ldots, \lambda_t\}$ for some $t \leq s$. As discussed in Section 2, given the characteristic polynomial of the recurrence, we can compute such a set, as well as integers m_i and $n_{i,j}$ for $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, t\}$, where the m_i are non-zero, such that for all $i \in \{1, \ldots, s\}$ we have

$$\lambda_i^{m_i} = \lambda_1^{n_{i,1}} \cdots \lambda_t^{n_{i,t}} \,.$$

Raising the left- and right-hand sides in the above equation to the power L and then taking (p-adic) logarithms, we get that

$$\log \lambda_i^L = \frac{n_{i,1}}{m_i} \log \lambda_1^L + \cdots + \frac{n_{i,t}}{m_i} \log \lambda_t^L$$

for all $i \in \{1, \ldots, s\}$. In other words, for all $i \in \{1, \ldots, s\}$ we have that $\log \lambda_i^L = \ell_i(\log \lambda_1^L, \ldots, \log \lambda_t^L)$ for an effectively computable linear form $\ell_i(X_1, \ldots, X_t)$ with rational coefficients.

For $j \in \mathbb{N}$, define $F_j \in \mathbb{K}[X_1, \ldots, X_t]$ by

$$F_j(X_1, \ldots, X_t) := \frac{1}{j!} \sum_{i=1}^{s} \sum_{k=0}^{j} \binom{j}{k} L^k Q_i^{(k)}(0) \ell_i(X_1, \ldots, X_t)^{j-k} .$$

Then by Equation (5) we have

$$a_j = F_j \left(\log \lambda_1^L, \ldots, \log \lambda_t^L\right) . \tag{7}$$

We claim that $a_j \neq 0$ if F_j is not identically zero. Since the coefficients of F_j are algebraic over $\mathbb{Q}$ and the set $\{\log \lambda_1, \ldots, \log \lambda_t\}$ is linearly independent over $\mathbb{Q}$, the claim follows immediately from Equation 7 and the weak p-adic Schanuel Conjecture (Conjecture 5).

We can now use the following procedure to compute a positive integer M such that $u_{Mn} \neq 0$ for all $n \in \mathbb{Z}$:

1. Successively compute the polynomials $F_0, F_1, \ldots$.
2. Let j_0 be the least index j such that F_j is not identically zero. Compute $\nu := v_p(a_{j_0})$ using the series (6). The weak p-adic Schanuel Conjecture implies that $a_{j_0} \neq 0$ and hence the computation of $v_p(a_{j_0})$ terminates.
3. Set $M := Lp^{\nu+1}$. Applying Proposition 7, we have $u_{Mn} \neq 0$ for all non-zero integers n.

Note that j_0 is well defined in Line 2, since if all the a_j were zero, then $\boldsymbol{u}$ would be the identically zero sequence, contradicting our initial assumption. ◀

A couple of remarks about the proof of Proposition 8 are in order.

▶ Remark 9. Observe that the p-adic Schanuel Conjecture is only required for termination of the procedure described at the end of the proof. If the procedure terminates then it is certain that a_{j_0} is the first non-zero coefficient of the power series (6) and hence the outputted value of M is guaranteed to be such that $u_{Mn} \neq 0$ for all non-zero integers n.

▶ Remark 10. Examining the expression (5) and noting that $Q_i^{(k)}(0) = 0$ for $k > \deg(Q_i)$, we see that the sequence $\langle j! a_j \rangle_{j=0}^{\infty}$ is given by an exponential polynomial corresponding to a (non-rational) LRS of order d and hence at least one of $a_0, a_1, \ldots, a_{d-1}$ is non-zero. This means that the index j_0 in Line 2 of the above procedure is at most $d - 1$.

▶ **Theorem 11.** *Assuming the weak p-adic Schanuel Conjecture, there is a Turing reduction from the Skolem Problem to the Bi-Skolem Problem.*

Proof. We present a recursive procedure that uses an oracle for the Bi-Skolem Problem to compute all the zeros of a non-degenerate LRBS that is not identically zero.

Given a non-degenerate LRBS $\boldsymbol{u} = \langle u_n \rangle_{n=-\infty}^{\infty}$, we use the oracle for the Bi-Skolem Problem to determine whether there exists $n \in \mathbb{Z}$ with $u_n = 0$. If the oracle outputs that no such n exists then the procedure terminates. Otherwise one searches for $n_0 \in \mathbb{Z}$ such that $u_{n_0} = 0$; clearly the search is guaranteed to terminate. Having found n_0, by reindexing the sequence $\boldsymbol{u}$ we can suppose that $n_0 = 0$. Now we use Proposition 8 to compute a positive integer M such that $u_{Mn} \neq 0$ for all $n \neq 0$. We then divide the sequence $\boldsymbol{u}$ into M subsequences $\boldsymbol{u}^{(0)}, \dots, \boldsymbol{u}^{(M-1)}$, where for $j \in \{0, \dots, M-1\}$, the j-th subsequence is given by $u_n^{(j)} = u_{Mn+j}$ for all $n \in \mathbb{Z}$. We know that $n = 0$ is the only zero of $\boldsymbol{u}^{(0)}$. We now recursively call the procedure to find all zeros of the remaining subsequences $\boldsymbol{u}^{(1)}, \dots, \boldsymbol{u}^{(M-1)}$. Observe that the computation must terminate since each recursive call involves discovering a new zero of the original sequence $\boldsymbol{u}$, and by the version of the Skolem-Mahler-Lech Theorem for LRBS, there are only finitely many such zeros. ◀

If we restrict to recurrences of order at most 5 then we obtain an unconditional version of Theorem 11.

▶ **Theorem 12.** *There is a Turing reduction from the Skolem Problem for LRS of order at most* 5 *to the Bi-Skolem Problem for simple LRBS of order at most* 5.

Proof. As summarised in Appendix A, the Skolem Problem can be decided for all LRS $\langle u_n \rangle_{n=0}^{\infty}$ of order at most 5 except those that (after scaling) have a closed form $u_n = \sum_{i=1}^{5} \alpha_i \lambda_i^n$ satisfying the following three conditions, where $\lambda_1, \dots, \lambda_5$ are algebraic integers that generate a number field $\mathbb{K} := \mathbb{Q}(\lambda_1, \dots, \lambda_5)$:

1. $\alpha_1 \neq -\alpha_3$;
2. $\lambda_1 \lambda_2 = \lambda_3 \lambda_4$;
3. there is a prime ideal $\mathfrak{p}$ in $\mathcal{O}_{\mathbb{K}}$ that divides λ_1 and λ_3 but not $\lambda_2, \lambda_4, \lambda_5$.

The theorem at hand is proven using the procedure described in the proof of Theorem 11, which uses as a subroutine the procedure described in Proposition 8. To avoid relying on the weak p-adic Schanuel Conjecture, it suffices to give an unconditional proof of the termination of the latter procedure when invoked on LRBS whose closed-form representation satisfies the above three conditions. In other words, we must show (unconditionally) that for such LRBS one can compute a positive integer M such that $u_{Mn} \neq 0$ for all $n \in \mathbb{Z} \setminus \{0\}$.

Let p be a prime satisfying Conditions (i)–(iii) listed in Section 3. In particular, we have an embedding of $\mathbb{K}$ into $\mathbb{Q}_p$. Recall from Section 3 that there exists a positive integer L such that $u_{Ln} = f(n)$ for a p-adic power series $f(X) = \sum_{j=0}^{\infty} a_j X^j$ such that $a_1 = \sum_{i=1}^{5} \alpha_i \log \lambda_i^L$. The termination of the procedure described in the proof of Proposition 8 will be assured if $a_1 \neq 0$. We claim that for an LRBS satisfying the above three conditions, one has $a_1 = \sum_{i=1}^{5} \alpha_i \log \lambda_i^L \neq 0$.

To prove the claim, suppose for a contradiction that $\sum_{i=1}^{5} \alpha_i \log \lambda_i^L = 0$. Raising to the L-th power and then taking logarithms in Condition 2 above, we also have $\log \lambda_1^L + \log \lambda_2^L - \log \lambda_3^L - \log \lambda_4^L = 0$. Combining the two previous equations to cancel $\log \lambda_1^L$ we have

$$(\alpha_2 - \alpha_1) \log \lambda_2^L + (\alpha_3 + \alpha_1) \log \lambda_3^L + (\alpha_4 + \alpha_1) \log \lambda_4^L + \alpha_5 \log \lambda_5^L = 0 \,. \tag{8}$$

From Condition 1 ($\alpha_1 \neq -\alpha_3$), we have that the coefficient of $\log \lambda_3^L$ in Equation (8) is non-zero. Applying Theorem 6, possibly several times, we eventually obtain an equation $\sum_{i=2}^{5} \beta_i \log \lambda_i^L = 0$ such that the β_i are integers and $\beta_3 \neq 0$. Equivalently, we have a multiplicative relation among the characteristic roots that involves λ_3 but not λ_1. But this contradicts Condition 3 and the proof is concluded. ◀

▶ **Theorem 13.** *The Skolem Problem for simple LRS is decidable assuming the Skolem Conjecture and the weak p-adic Schanuel Conjecture. The Skolem Problem for LRS of order at most 5 is decidable assuming the Skolem Conjecture.*

▶ Remark 14. Given that the Skolem Conjecture remains open in general, it is worth remarking that the proof of Theorem 13 sustains the following more general formulation: Consider a class $\mathcal{C}$ of simple LRBS that is closed under taking subsequences and under translations. If the Skolem Conjecture holds for $\mathcal{C}$ then, assuming the weak p-adic Schanuel Conjecture, the Skolem Problem is decidable over LRS in $\mathcal{C}$.

5 The Skolem Tool

We have implemented our algorithm in the Skolem tool, which finds all zeros (at both positive and negative indices) for simple integer LRS, and produces independent certificates guaranteeing that there are no further zeros. Even though we do not have complexity bounds, Skolem can efficiently handle many interesting examples, including several from the literature for which no proof technique was previously known to apply. Our tool is available online at `https://skolem.mpi-sws.org` and includes several built-in examples.

The implementation is written in Python, using the SageMath computer-algebra extension. This allows for the efficient and exact manipulation of mathematical objects, including elements of $\mathbb{Z}_p$. Python handles integers of arbitrary sizes seamlessly, making it especially suitable for our purposes, since even small examples can give rise to very large numbers within the inner workings of our algorithm.

▶ **Example 15.** Consider the LRS from [22, Example 2.4]:

$$u_{n+5} = 9u_{n+4} - 10u_{n+3} + 522u_{n+2} - 4745u_{n+1} + 4225u_n$$

with initial terms (for $n = 0, 1, 2, 3, 4$) of $\langle -30, -27, 0, 469, 1762\rangle$. It is shown in [22] to have a unique zero at index 2 by being non-zero modulo 12625 at all indices larger than 2. The Skolem tool establishes this in a simpler way: after finding $u_2 = 0$, the tool calculates that there are no zeros in $\langle u_{2+14n}\rangle_{n=-\infty}^{\infty}$ for $n \neq 0$. Then the tool computes that $\langle u_{k+14n}\rangle_{n=-\infty}^{\infty}$ is non-zero modulo 29 for each even $k \neq 2$, and non-zero modulo 2 for each odd k (where $0 \leq k \leq 13$). Observe that the computed modulo classes, and thus the resulting certificate, is much smaller than those arising from 12625 as used in [22].

▶ **Example 16.** Consider the LRS from [22, Example 2.5]:

$$u_{n+6} = 6u_{n+5} - 26u_{n+4} + 66u_{n+3} - 130u_{n+2} + 150u_{n+1} - 125u_n$$

with initial terms (for $n = 0, 1, 2, 3, 4, 5$) of $\langle 0, 3, 11, -12, -125, -177\rangle$, which was established at the time of writing to lie beyond the reach of existing known techniques. The Skolem tool is able to show using the methods developed in the present paper that there are indeed no further zeros (other than $u_0 = 0$).

▶ **Example 17.** Consider the reversible order-8 LRS from [22, Example 3.5]:

$$u_{n+8} = 6u_{n+7} - 25u_{n+6} + 66u_{n+5} - 120u_{n+4} + 150u_{n+3} - 89u_{n+2} + 18u_{n+1} - u_n$$

with initial terms (for $n = 0, \ldots, 7$) of $\langle 0, 0, -48, -120, 0, 520, 624, -2016\rangle$, which likewise was established at the time to lie beyond the reach of existing techniques. Skolem shows that there are no zeros other than those lying at indices 0, 1, and 4.

Table 1 Table showing the distribution of outcomes by order. The line between orders 6 and 7 shows the boundary beyond which more than 50% of runs timeout. The second experiment shows the timeout rate when the timeout is increased to $60s \cdot \textit{order}$ (“degenerate” and “not-simple” counts omitted as the distribution is similar to the 60s timeout experiment and unaffected by the timeout).

	Timeout of 60 seconds						Timeout of $60 \cdot \textit{order}$ seconds			
Order	Total	Success	Degenerate	Not simple	Timeout	Timeout %	Total	Success	Timeout	Timeout %
2	9250	8836	358	50	0	0.00%	1245	1200	0	0.00%
3	8995	8919	74	2	0	0.00%	1327	1322	0	0.00%
4	9195	9157	35	2	1	0.01%	1395	1392	0	0.00%
5	9188	8700	15	3	470	5.12%	1303	1290	11	0.84%
6	9172	4905	10	6	4251	46.35%	1318	952	366	27.77%
7	9213	1339	12	0	7862	85.34%	1328	310	1016	76.51%
8	9157	378	10	0	8769	95.76%	1249	73	1173	93.92%
9	9143	87	4	3	9049	98.97%	1330	18	1312	98.65%
10	9047	25	8	1	9013	99.62%	1294	7	1286	99.38%
Total	82360	42346	526	67	39415	47.86%	11789	6564	5164	43.80%

5.1 Testing

The Skolem tool was tested on a suite of random LRS, with the order taken uniformly between 2 and 10 and the coefficients taken uniformly at random between -20 and 20. Tests were run for 48 hours using a 60-second timeout[4], generating 82367 test instances.[5]

The results are presented in Tables 1 and 2. In particular, from order 7 onwards the tool is unable to handle more than half of the instances within one minute, with the timeout percentage jumping significantly from order 6. Both degenerate and non-simple LRS instances are very sparse, and as expected the higher the order the fewer such instances were randomly produced.

The experiments were re-run using a timeout of $60 \cdot \textit{order}$ seconds (i.e., ranging from 2–10 minutes) in order to determine whether the 60-second timeout was too strict. The timeout percentage does decrease, but the overall pattern shows that the vast majority of LRS of order 7 and above could not be handled to completion before the timeout.

Table 2 presents statistical information. In the main experiment the average time is below 9 seconds for order-6 examples that succeed within 60 seconds. However, the decrease from order-8 onwards is explained by there being significantly fewer examples succeeding within 60 seconds. On average there are very few zeros (as can be expected) and those that do occur are almost always those occurring within the initial terms (the largest zero is nearly always at index less than the order).

The average maximum jump step used (i.e., M such that $\langle u_{Mn} \rangle$ has no zeros for $n \neq 0$ and $u_0 = 0$) is observed on average to grow with the order (except at order 10 with only 25 successful samples).

[4] Testing was conducted using SageMath 9.5 in Docker on a Dell PowerEdge M620 blade equipped with 2× 3.3 GHz Intel Xeon E5-2667 v2 (2×8 cores, 32 with hyper-threading) and 256GB ram. Testing was restricted to 16 parallel threads (50% of the computer’s resources) for institutional reasons.

[5] 7 instances were discarded: 6 happened to be the zero sequence, one resulted in an exception (outside of the main tool code) which was later fixed.

Table 2 Table listing statistical information for successful runs, by order. Line between orders 6 and 7 shows the boundary beyond which more than 50% of runs timeout, resulting in skewed analysis for the subsequent rows. For the second experiment, with timeout of $60 \cdot \mathit{order}$ seconds, only the mean time is shown as there are fewer data points.

Order	mean time (seconds)	mean count of zeros	max count of zeros	mean max zero	max zero index	mean tree depth	mean max jump	mean time (seconds) $60s \cdot \mathit{order}$
2	0.03	0.06	1	0.63	6	1.06	5.11	0.03
3	0.05	0.08	3	1.03	5	1.08	13.56	0.06
4	0.19	0.10	3	1.58	7	1.10	37.05	0.22
5	4.82	0.11	2	2.05	6	1.11	107.39	10.07
6	8.95	0.20	2	2.58	7	1.20	254.12	55.36
7	11.72	0.30	2	2.80	9	1.30	482.34	70.19
8	8.07	0.35	2	3.51	8	1.35	533.92	68.21
9	7.38	0.38	1	4.24	8	1.38	689.33	138.40
10	5.71	0.40	1	5.20	9	1.40	249.60	112.11

References

1 M. Agrawal, S. Akshay, B. Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of Markov chains. *J. ACM*, 62(1):2:1–2:34, 2015.

2 S. Akshay, T. Antonopoulos, J. Ouaknine, and J. Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015.

3 S. Almagor, T. Karimov, E. Kelmendi, J. Ouaknine, and J. Worrell. Deciding ω-regular properties on linear recurrence sequences. *Proc. ACM Program. Lang.*, 5(POPL), 2021.

4 C. Baier, F. Funke, S. Jantsch, T. Karimov, E. Lefaucheux, F. Luca, J. Ouaknine, D. Purser, M. A. Whiteland, and J. Worrell. The Orbit Problem for parametric linear dynamical systems. In *32nd International Conference on Concurrency Theory, CONCUR 2021*, volume 203 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

5 G. Barthe, C. Jacomme, and S. Kremer. Universal equivalence and majority of probabilistic programs over finite fields. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 155–166. ACM, 2020.

6 B. Bartolome, Y. Bilu, and F. Luca. On the exponential local-global principle. *Acta Arith.*, 159(2):101–111, 2013.

7 Cs. Bertók and L. Hadju. A Hasse-type principle for exponential Diophantine equations and its applications. *Math. Comput.*, 85:849–860, 2016.

8 Cs. Bertók and L. Hadju. A Hasse-type principle for exponential Diophantine equations over number fields and its applications. *Monatshefte Math.*, 187:425–436, 2018.

9 V. Blondel and J. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000. `doi:10.1016/S0005-1098(00)00050-9`.

10 V. D. Blondel and N. Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and Its Applications*, 351–352, 2002.

11 A. Brumer. On the units of algebraic number fields. *Mathematika*, 14:121–124, 1967.

12 F. Calegari and B. Mazur. Nearly ordinary Galois deformations over arbitrary number fields. *J. Inst. Math. Jussieu*, 8(1):99–177, 2009.

13 K. Chatterjee and L. Doyen. Stochastic processes with expected stopping time. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–13. IEEE, 2021.

14 V. Chonev, J. Ouaknine, and J. Worrell. The Polyhedron-Hitting Problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 940–956. SIAM, 2015.

15 G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward. *Recurrence Sequences.* American Mathematical Society, 2003.

16 N. Fijalkow, J. Ouaknine, A. Pouly, J. Sousa Pinto, and J. Worrell. On the decidability of reachability in linear time-invariant systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019*, pages 77–86. ACM, 2019.

17 F. Gouvea. *p-adic Numbers: An Introduction, Second Edition.* Universitext. Springer, 1997.

18 T. Karimov, E. Lefaucheux, J. Ouaknine, D. Purser, A. Varonka, M. A. Whiteland, and J. Worrell. What's decidable about linear loops? *Proc. ACM Program. Lang.*, 6(POPL), 2022.

19 J. C. Lagarias and A. M. Odlyzko. *Effective versions of the Chebotarev density theorem.* Academic Press, London, 1977.

20 S. Landau. Factoring polynomials over algebraic number fields. *SIAM Journal on Computing*, 14(1):184–195, 1985.

21 Serge Lang. *Introduction to Transcendental Numbers.* Addison-Wesley, 1966.

22 R. Lipton, F. Luca, J. Nieuwveld, J. Ouaknine, D. Purser, and J. Worrell. On the Skolem Problem and the Skolem Conjecture. *To appear, Proceedings of the 37th Annual ACM/IEEE Symposium on Logic and Computer Science, LICS'22*, 2022.

23 A. 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.

24 N. Mariaule. p-adic exponential ring, p-adic Schanuel's conjecture and decidability. *PhD Thesis, University of Manchester*, 2014.

25 D. W. Masser. Linear relations on algebraic groups. In *New Advances in Transcendence Theory.* Cambridge University Press, 1988.

26 M. Mignotte, T. N. Shorey, and R. Tijdeman. The distance between terms of an algebraic recurrence sequence. *Journal für die reine und angewandte Mathematik*, 349, 1984.

27 J. Ouaknine and J. Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, 2015.

28 J. Piribauer and C. Baier. On Skolem-hardness and saturation points in Markov decision processes. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 138:1–138:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

29 G. Rozenberg and A. Salomaa. *Cornerstones of Undecidability.* Prentice Hall, 1994.

30 A. Schinzel. Abelian binomials, power residues and exponential congruences. *Acta Arith.*, 32(3):245–274, 1977.

31 A. Schinzel. On the congruence $u_n \equiv c \pmod p$ where u_n is a recurring sequence of the second order. *Acta Acad. Paedagog. Agriensis Sect. Math.*, 30:147–165, 2003.

32 Th. Skolem. Anwendung exponentieller Kongruenzen zum Beweis der Unlösbarkeit gewisser diophantischer Gleichungen. *Avhdl. Norske Vid. Akad. Oslo I*, 12:1–16, 1937.

33 M. Soittola. On D0L synthesis problem. In A. Lindenmayer and G. Rozenberg, editors, *Automata, Languages, Development.* North-Holland, 1976.

34 N. K. Vereshchagin. The problem of appearance of a zero in a linear recurrence sequence (in Russian). *Mat. Zametki*, 38(2), 1985.

A Hard Cases of the Skolem Problem at Order 5

As explained in [22], the Skolem Problem is known to be decidable for all LRS of order at most 5 except for those sequences $\boldsymbol{u} = \langle u_n \rangle_{n=0}^{\infty}$ having an exponential-polynomial representation

$$u_n = \alpha_1 \lambda_1^n + \overline{\alpha_1}\overline{\lambda_1}^n + \alpha_2 \lambda_2^n + \overline{\alpha_2}\overline{\lambda_2}^n + \alpha_3 \lambda_3^n \tag{9}$$

such that $\alpha_1, \alpha_2, \alpha_3, \lambda_1, \lambda_2, \lambda_3 \in \overline{\mathbb{Q}}$ satisfy $|\lambda_1| = |\lambda_2| > |\lambda_3|$ and $\lambda_1, \lambda_2, \lambda_3$ are not all units. It is further shown in [22] that by scaling sequences of this form we can assume that there exists a prime ideal $\mathfrak{p}$ in the ring of integers of the number field generated by $\lambda_1, \overline{\lambda_1}, \lambda_2, \overline{\lambda_2}, \lambda_3$ such that $\mathfrak{p}$ divides λ_1 and λ_2, but not $\overline{\lambda_1}, \overline{\lambda_2}$ and λ_3.

Here we make the further observation that for non-degenerate LRS of the form (9), under the assumption that $|\alpha_1| = |\alpha_2|$, there is a computable upper bound on n such that $u_n = 0$.

By scaling we can assume without loss of generality that $|\lambda_1| = |\lambda_2| = 1$ and $|\alpha_1| = |\alpha_2| = 1$. Thus we can write $\lambda_1 = e^{i\theta_1}$ and $\lambda_2 = e^{i\theta_2}$ for $\theta_1, \theta_2 \in [0, 2\pi)$ and we can put $\alpha_1 = e^{i\phi_1}$ and $\alpha_2 = e^{i\phi_2}$ for $\phi_1, \phi_2 \in [0, 2\pi)$. Then we have

$$\begin{aligned} u_n &= \alpha_1\lambda_1^n + \overline{\alpha_1}\overline{\lambda_1}^n + \alpha_2\lambda_2^n + \overline{\alpha_2}\overline{\lambda_2}^n + \alpha_3\lambda_3^n \\ &= 2\cos(n\theta_1 + \phi_1) + 2\cos(n\theta_2 + \phi_2) + \alpha_3\lambda_3^n \\ &= 4\left(\cos\left(\frac{n(\theta_1+\theta_2)+\phi_1+\phi_2}{2}\right)\cos\left(\frac{n(\theta_1-\theta_2)+\phi_1-\phi_2}{2}\right)\right) + \alpha_3\lambda_3^n\,. \end{aligned}$$

By non-degeneracy of $\boldsymbol{u}$, the terms $\cos\left(\frac{n(\theta_1+\theta_2)+\phi_1+\phi_2}{2}\right)$ and $\cos\left(\frac{n(\theta_1-\theta_2)+\phi_1-\phi_2}{2}\right)$ are respectively zero for at most one value of $n \in \mathbb{N}$. Furthermore, using Baker's Theorem on linear forms in logarithms (see [26, 34] for details), each of these terms has a lower bound (when non-zero) of the form $\frac{c}{n^d}$ for explicitly computable constants c and d. Since $|\lambda_3| < 1$ it follows that $u_n \neq 0$ for all $n \geq n_0$ for some effective threshold n_0.

Deepening the (Parameterized) Complexity Analysis of Incremental Stable Matching Problems

Niclas Boehmer
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Klaus Heeger
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Rolf Niedermeier
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Abstract

When computing stable matchings, it is usually assumed that the preferences of the agents in the matching market are fixed. However, in many realistic scenarios, preferences change over time. Consequently, an initially stable matching may become unstable. Then, a natural goal is to find a matching which is stable with respect to the modified preferences and as close as possible to the initial one. For Stable Marriage/Roommates, this problem was formally defined as Incremental Stable Marriage/Roommates by Bredereck et al. [AAAI '20]. As they showed that Incremental Stable Roommates and Incremental Stable Marriage with Ties are NP-hard, we focus on the parameterized complexity of these problems. We answer two open questions of Bredereck et al. [AAAI '20]: We show that Incremental Stable Roommates is W[1]-hard parameterized by the number of changes in the preferences, yet admits an intricate XP-algorithm, and we show that Incremental Stable Marriage with Ties is W[1]-hard parameterized by the number of ties. Furthermore, we analyze the influence of the degree of "similarity" between the agents' preference lists, identifying several polynomial-time solvable and fixed-parameter tractable cases, but also proving that Incremental Stable Roommates and Incremental Stable Marriage with Ties parameterized by the number of different preference lists are W[1]-hard.

2012 ACM Subject Classification Theory of computation → W hierarchy; Theory of computation → Fixed parameter tractability; Theory of computation → Design and analysis of algorithms; Theory of computation → Algorithmic game theory

Keywords and phrases Stable Marriage, Stable Roommates, adapting to changing preferences, NP-hardness, W[1]-hardness, XP, FPT, master lists, incremental algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.21

Related Version *Full Version*: `https://arxiv.org/abs/2208.01563`

Acknowledgements NB was supported by the DFG project MaMu (NI 369/19) and by the DFG project ComSoc-MPMS (NI 369/22). KH was supported by the DFG Research Training Group 2434 "Facets of Complexity" and by the DFG project FPTinP (NI 369/16).

1 Introduction

Efficiently adapting solutions to changing inputs is a core issue in modern algorithmics [3, 6, 9, 16, 27]. In particular, in *incremental* combinatorial problems, roughly speaking, the goal is to build new solutions incrementally while adapting to changes in the input. Typically, one wants to avoid (if possible) too radical changes in the solution relative to perhaps moderate changes in the input. The corresponding study of incremental algorithms attracted research on numerous problems and scenarios [24], including among many others shortest path computation [40], flow computation [31], clustering problems [9, 35], and graph coloring [28].

In this paper, we study the problem of adapting stable matchings under preferences to change. Consider for instance the following two scenarios: First, as reported by Feigenbaum et al. [18], school seats in public schools are centrally assigned in New York. Ahead of the

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 21; pp. 21:1–21:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

start of the new year, all interested students are asked to submit their preferences over public schools. Then, a stable matching of students to public schools is computed and transmitted. However, in the past, shortly before the start of the new year typically around 10% of students changed their preferences and decided to attend a private school instead, leaving the initially implemented matching unstable and triggering lengthy decentralized ad hoc updates. Second, consider the assignment of freshmen to double bedrooms in college accommodation. After the orientation weeks, it is quite likely that students got to know each other (and in particular their roommates) better and thus their initially uninformed preferences changed, making the matching unstable.

In our work, we focus on the problem of finding a stable matching after the "change" that is as close as possible to a given initially stable matching. The closeness condition here is due to the fact that in most applications reassignments come at some cost which we want to minimize (e.g., in the above New York example, reassigning students might make it necessary for the family to reallocate within the city). We build upon the work of Bredereck et al. [7], who performed a first systematic study of incremental versions of stable matching problems, and the recent (partially empirical) follow-up work by Boehmer et al. [5], who proved among others that different types of changes are "equivalent" to each other. The central focus of our studies lies on the Stable Roommates (SR) problem: given a set of agents with each agent having preferences over other agents, the task is to find a stable matching, i.e., a matching so that there are no two agents preferring each other to their assigned partner. We also consider a famous special case of SR, namely Stable Marriage (SM), where the set of agents is partitioned into two sets, and each agent may only be matched to an agent from the other set. Formally, in the incremental versions of SR and SM, called Incremental Stable Roommates (ISR) and Incremental Stable Marriage (ISM), we are given two preference profiles containing the preferences of each agent before and after the "change" and a matching that is stable in the preference profile before the change. Then, the task is to find a matching that is stable after the change and as close as possible to the given matching, i.e., has a minimum symmetric difference to it.

Related Work. Bredereck et al. [7] formally introduced Incremental Stable Marriage [with ties] (ISM/[ISM-T]) and Incremental Stable Roommates [with ties] (ISR/[ISR-T]). They showed that ISM without ties (in the preference lists) is solvable in polynomial time by a simple reduction to finding a stable matching maximizing the weight of the included agent pairs (which is solvable in polynomial time [17]). In contrast to this, ISR is NP-complete [12, Theorem 4.2], yet admits an FPT-algorithm for the parameter k, that is, the maximum allowed size of the symmetric difference between the two matchings [7]. With ties, Bredereck et al. [7] showed that ISM-T and ISR-T are NP-complete and W[1]-hard for k even if the two preference profiles only differ in a single swap in some agent's preference list. As ISR-T can be considered as a generalization of ISM-T, their results motivate us to focus on the NP-hard ISR and ISM-T problems (which are somewhat incomparable problems). Recently, Boehmer et al. [5] followed up on the work of Bredereck et al. [7], proving that different types of changes such as deleting agents or performing swaps of adjacent agents in some preference list are "equivalent". Moreover, they introduced incremental variants of further stable matching problems and performed empirical studies.

More broadly considered, matching problems involving preferences in the presence of change are of high current interest in several application domains. Many such works fall into the category of "dynamic matchings" [1, 2, 14, 15, 34]. However, different from our work, there the focus is typically on adapting classic stability notions to dynamic settings while we rather aim for "reestablishing" (classic) stability at minimal change cost.

Table 1 Overview of our main results where each row contains results for one parameterization. Note that ISM is polynomial-time solvable as proven by Bredereck et al. [7].

	ISR	ISM-T
$\lvert\mathcal{P}_1 \oplus \mathcal{P}_2\rvert$	W[1]-h. (Th. 1) & XP (Th. 2)	NP-h. for $\lvert\mathcal{P}_1 \oplus \mathcal{P}_2\rvert = 1$ (Th. 5)
#ties+k	FPT wrt. k (Th. 1 in [7])	W[1]-h. even if $\lvert\mathcal{P}_1 \oplus \mathcal{P}_2\rvert = 1$ (Th. 5) XP (even for parameter #agents with ties) (Pr. 6)
#outliers	FPT (Th. 10)	?
#master lists	W[1]-h. even for complete preferences (Th. 11/12)	

Closer to our work, there are several papers on adapting a given matching to change (while minimizing the number of reassignments): First, Gajulapalli et al. [20] designed a polynomial-time (and incentive-compatible) algorithm for an incremental variant of the many-to-one version of Stable Marriage (known as Hospital Residents) where new agents are added. Second, Feigenbaum et al. [18] considered an incremental variant of Hospital Residents where some agents may leave the system. They designed a "fair", Pareto-efficient, and strategy-proof algorithm for finding a matching before and after the change. Both these works are closest related to the polynomial-time solvable ISM problem, which we do not study. Third, Bhattacharya et al. [3] studied one-to-one matching markets where agents are added and deleted over time and for some agents the set of acceptable partners may change. Their focus is on updating the matching in each step such that the number of reassignments is small while maintaining a small unpopularity factor. So in contrast to our work, they do not maintain that the matching is stable but (close to) popular.

Also motivated by temporally evolving preferences, several papers study the robustness of stable matchings subject to changing preferences [4, 11, 21, 22, 23, 36]. By selecting a robust initial stable matching, one can increase the odds that it remains stable after some changes.

Our Contributions. Focusing on the two NP-hard problems ISR and ISM-T, we significantly extend the work of Bredereck et al. [7] on incremental stable matchings, particularly answering their two main open questions. Moreover, we strengthen several of their results. In addition, we analyze the impact of the degree of "similarity" between the agents' preference lists. Doing so, from a conceptual perspective, we complement work of Meeks and Rastegari [38]. They studied the influence of the number of agent types on the computational complexity of stable matching problems (two agents are of the same type if they have the same preferences and all other agents are indifferent between them). By way of contrast, we consider the smaller so far unstudied parameter "number of different preference lists".

Next, we present a brief summary of the structure of the paper (for each section marking the main studied problem(s)) and our main contributions (see Table 1 for an overview):

Section 3 (ISR). Motivated by the observation that ISM-T is NP-hard even if just one swap has been performed, Bredereck et al. [7] asked for the parameterized complexity of ISR with respect to the difference $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ between the two given preference profiles. We design and analyze an involved algorithm solving ISR in polynomial time if $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ is constant (in other words, this is an XP-algorithm). Our algorithm relies on the observation that if we know how certain agents are matched in the matching to be constructed and we adapt the given matching accordingly, then we can find an optimal solution by propagating these changes through the matching until a new stable matching is reached; a general approach that might be of independent interest. We complement this result by proving that ISR parameterized by $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ is W[1]-hard.

Section 4 (ISM-T). Bredereck et al. [7] considered the total number of ties to be a promising parameter to potentially achieve fixed-parameter tractability results. We prove that this is not the case as ISM-T is W[1]-hard with respect to k *plus* the total number of ties even if $|\mathcal{P}_1 \oplus \mathcal{P}_2| = 1$. Notably, this result strengthens the W[1]-hardness with respect to k for $|\mathcal{P}_1 \oplus \mathcal{P}_2| = 1$ of Bredereck et al. [7] for ISM-T, while presenting a fundamentally different yet less technical proof. On the positive side, we devise an XP-algorithm for the number of agents with at least one tie in their preferences.

Section 5 (ISR; ISM-T). We study different cases where the agents have "similar" preferences. For instance, we consider the case where all but s agents have the same preference list (we call these s agents *outliers*), or the case where each agent has one out of only p different *master preference lists*. We devise an algorithm that enumerates all stable matchings in an SR instance in FPT time with respect to s, implying an FPT algorithm for ISR parameterized by s. In contrast to this and to a simple FPT algorithm for the number of agent types [5], we prove that ISR and ISM-T are W[1]-hard with respect to the number p of different preference lists.

2 Preliminaries

The input of Stable Roommates with Ties (SR-T) is a set $A = \{a_1, \dots, a_{2n}\}$ of agents. Each agent $a \in A$ has a subset $\mathrm{Ac}(a) \subseteq A \setminus \{a\}$ of agents it *accepts* and a preference relation $\succsim_a$ which is a weak order over the agents $\mathrm{Ac}(a)$. Without loss of generality, we assume that acceptance is symmetric, i.e., for two agents $a, a' \in A$, $a' \in \mathrm{Ac}(a)$ implies $a \in \mathrm{Ac}(a')$. We collect the preferences of all agents in a *preference profile* $\mathcal{P}$. For two agents $a', a'' \in \mathrm{Ac}(a)$, agent a *weakly prefers* a' to a'' if $a' \succsim_a a''$. If a weakly prefers a' to a'' but does not weakly prefer a'' to a', then a *strictly prefers* a' to a'', and we write $a' \succ_a a''$. If a weakly but not strictly prefers a' to a'', then a is *indifferent* between a' and a'' and we write $a' \sim_a a''$; in other words, a' and a'' are *tied*. If a strictly prefers a' to a'' or $a' = a''$ holds, then we write $a' \succeq_a a''$. We say that an agent a has strict preferences, which we denote as $\succ_a$, if $\succsim_a$ is a strict order, and, in this case, we use the terms "strictly prefer" and "prefer" interchangeably. For two preference relations $\succsim$ and $\succsim'$ defined over the same set, the swap distance between $\succsim$ and $\succsim'$ is the number of agent pairs that are ordered differently by the two relations, i.e., $|\{(a,b) : a \succ b \wedge b \succsim' a\}| + |\{(a,b) : a \sim b \wedge \neg(a \sim' b)\}|$; for two preference relations over different sets, we define the swap distance to be infinite. For two preference profiles $\mathcal{P}_1$ and $\mathcal{P}_2$ containing the preferences of the same agents, $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ denotes the total swap distance between the two preference relations of an agent summed over all agents.[1]

A *matching* M is a set of pairs $\{a, a'\}$ with $a \neq a' \in A$, $a \in \mathrm{Ac}(a')$, and $a' \in \mathrm{Ac}(a)$, where each agent appears in at most one pair. In a matching M, an agent a is *matched* if a is part of one pair from M; otherwise, a is *unmatched*. A *perfect matching* is a matching in which all agents are matched. For a matching M and an agent $a \in A$, we denote by $M(a)$ the partner of a in M, i.e., $M(a) = a'$ if $\{a, a'\} \in M$ and $M(a) := \square$ if a is unmatched in M. All agents $a \in A$ strictly prefer any agent from $\mathrm{Ac}(a)$ to being unmatched, i.e., $a' \succ_a \square$ for all $a \in A$ and $a' \in \mathrm{Ac}(a)$.

[1] Notably, by the equivalence theorem of Boehmer et al. [5, Theorem 1], all our results (except for Theorem 5 where the constant $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ increases by a small number) still hold if $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ instead denotes the number of agents whose preferences changed, the number of deleted agents (i.e., the number of agents with empty preferences in $\mathcal{P}_2$ and non-empty preferences in $\mathcal{P}_1$), or the number of added agents (i.e., the number of agents with empty preferences in $\mathcal{P}_1$ and non-empty preferences in $\mathcal{P}_2$).

Two agents $a \neq a' \in A$ *block* a matching M if a and a' accept each other and strictly prefer each other to their partners in M, i.e., $a \in \mathrm{Ac}(a')$, $a' \in \mathrm{Ac}(a)$, $a' \succ_a M(a)$, and $a \succ_{a'} M(a')$. A matching M is *stable* if it is not blocked by any agent pair.[2] An agent pair $\{a, a'\}$ is called a *stable pair* if there is a stable matching M with $\{a, a'\} \in M$. For two matchings M and M', we denote by $M \triangle M'$ the set of pairs that appear only in M or only in M', i.e., $M \triangle M' = \{\{a, a'\} \mid (\{a, a'\} \in M \wedge \{a, a'\} \notin M') \vee (\{a, a'\} \notin M \wedge \{a, a'\} \in M')\}$. The incremental variant of Stable Roommates [with Ties] is defined as follows.

> Incremental Stable Roommates [with Ties] (ISR/[ISR-T])
> **Input:** A set A of agents, two preference profiles $\mathcal{P}_1$ and $\mathcal{P}_2$ containing the strict [weak] preferences of all agents, a stable matching M_1 in $\mathcal{P}_1$, and an integer k.
> **Question:** Is there a matching M_2 that is stable in $\mathcal{P}_2$ such that $|M_1 \triangle M_2| \leq k$?

We also consider the incremental variant of Stable Marriage. Instances of Stable Marriage are instances of Stable Roommates where the set of agents is partitioned into two sets U and W such that agents from one of the sets only accept agents from the other set, i.e., $\mathrm{Ac}(m) \subseteq W$ for all $m \in U$ and $\mathrm{Ac}(w) \subseteq U$ for all $w \in W$. Following traditional conventions, we refer to the agents from U as men and to the agents from W as women. All definitions from above still analogously apply to Stable Marriage. Thus, in Incremental Stable Marriage [with Ties] (ISM/[ISM-T]), we are given a set $A = U \cup W$ of agents and two preference profiles $\mathcal{P}_1$ and $\mathcal{P}_2$ containing the strict [weak] preferences of all agents, where each $m \in U$ accepts only agents from W and the other way round.

Lastly, in Stable Roommates, the preferences of an agent $a \in A$ are *complete* if $\mathrm{Ac}(a) = A \setminus \{a\}$. In Stable Marriage, the preferences of an agent $a \in U \cup W$ are *complete* if $\mathrm{Ac}(a) = W$ for $a \in U$ or if $\mathrm{Ac}(a) = U$ for $a \in W$. If the preferences of an agent are not complete, then they are *incomplete*.

We defer the proofs (or their completions) of all results marked by $(\star)$ to a full version.

3 Incremental Stable Roommates Parameterized by $|\mathcal{P}_1 \oplus \mathcal{P}_2|$

Bredereck et al. [7] showed that ISR-T and ISM-T are NP-hard even if $\mathcal{P}_1$ and $\mathcal{P}_2$ differ only by a single swap. While Bredereck et al. showed that ISR (without ties) is NP-hard, they asked whether it is fixed-parameter tractable parameterized by $|\mathcal{P}_1 \oplus \mathcal{P}_2|$. We show that ISR is W[1]-hard with respect to $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ (Section 3.1), yet admits an intricate polynomial-time algorithm for constant $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ (Section 3.2), thus still clearly distinguishing it from the case with ties.

3.1 W[1]-Hardness

This section is devoted to proving that ISR with respect to $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ is W[1]-hard:

▶ **Theorem 1** $(\star)$. *ISR parameterized by $|\mathcal{P}_1 \oplus \mathcal{P}_2|$ is W[1]-hard.*

To prove the theorem, we reduce from the W[1]-hard Multicolored Clique problem parameterized by the solution size ℓ [39]. In Multicolored Clique, we are given an ℓ-partite graph $G = (V^1 \cup V^2 \cup \dots \cup V^\ell, E)$ and the question is whether there is a clique X

[2] This definition of stability in the presence of ties is the by far most frequently studied variant known as *weak stability*. *Strong stability* and *super stability* are the two most popular alternatives. Notably, ISM-T (as defined later) becomes polynomial-time solvable for both strong and super stability, as for these two stability notions a stable matching maximizing a given weight function on all pairs of agents can be found in polynomial time [19, 32, 33].

of size ℓ in G with $X \cap V^c \neq \emptyset$ for all $c \in [\ell]$. To simplify notation, we assume that $V^c = \{v_1^c, \ldots, v_\nu^c\}$ for all $c \in [\ell]$ and that the given graph G is r-regular for some $r \in \mathbb{N}$. We refer to the elements of $[\ell]$ as *colors* and say that a vertex v has color $c \in [\ell]$ if $v \in V^c$. The structure of the reduction is as follows. For each color $c \in [\ell]$, there is a vertex-selection gadget, encoding which vertex from V^c is part of the multicolored clique. Furthermore, there is one edge gadget for each edge. Unless both endpoints of an edge are selected by the corresponding vertex-selection gadgets, the matching M_2 selected in the edge gadget contributes to the difference $M_1 \triangle M_2$ between M_1 and M_2. We set k (that is, the upper bound on $|M_1 \triangle M_2|$) such that at least $\binom{\ell}{2}$ edges need to have both endpoints in the selected set of vertices, implying that the selected set of vertices forms a clique.

Vertex-Selection Gadget. For each color $c \in [\ell]$, we add a vertex selection gadget. For each vertex $v_i^c \in V^c$, we add one 4-cycle consisting of agents $a_{i,1}^c$, $a_{i,2}^c$, $a_{i,3}^c$, and $a_{i,4}^c$. Further, in $\mathcal{P}_2$, two agents s^c and $\bar{s}^c$ are "added" to the gadget (more formally, s^c and $\bar{s}^c$ are matched to dummy agents t^c and $\bar{t}^c$ in all stable matching in $\mathcal{P}_1$ but cannot be matched to t^c and $\bar{t}^c$ in a stable matching in $\mathcal{P}_2$). We construct the vertex-selection gadget such that the agents s^c and $\bar{s}^c$ have to be matched to agents from the same 4-cycle in a stable matching in $\mathcal{P}_2$. This encodes the selection of the vertex corresponding to this 4-cycle to be part of the multicolored clique. Lastly, we add a second 4-cycle consisting of agents $\bar{a}_{i,1}^c$, $\bar{a}_{i,2}^c$, $\bar{a}_{i,3}^c$, and $\bar{a}_{i,4}^c$ for each vertex $v_i^c \in V^c$ to achieve that $M_1 \triangle M_2$ contains the same number of pairs from the vertex-selection gadget, independent of which vertex is selected to be part of the clique.

Apart from agents s^c and $\bar{s}^c$, all agents from the vertex-selection gadget only find agents from the gadget acceptable, while s^c and $\bar{s}^c$ also find agent $a_{e,1}$ (this agent will be introduced in the next paragraph "Edge Gadget") for each edge e incident to a vertex from V^c acceptable. For $v_i^c \in V^c$, let $A_{\delta(v_i^c),1}$ denote the set of agents $a_{e,1}$ such that e is an edge incident to v_i^c, i.e., $A_{\delta(v_i^c),1} := \{a_{e,1} \mid e \in E \wedge e \cap v_i^c \neq \emptyset\}$, and let $[A_{\delta(v_i^c),1}]$ denote an arbitrary strict order of $A_{\delta(v_i^c),1}$. For all $c \in [\ell]$ and $i \in [n]$, each vertex-selection gadget contains the following agents with the indicated preferences in $\mathcal{P}_1$:

$$
\begin{aligned}
&s^c : t^c \succ a_{1,1}^c \succ \bar{a}_{1,1}^c \succ [A_{\delta(v_1^c),1}] \succ a_{2,1}^c \succ \bar{a}_{2,1}^c \succ [A_{\delta(v_2^c),1}] \succ \cdots \succ a_{n,1}^c \\
&\quad \succ \bar{a}_{n,1}^c \succ [A_{\delta(v_1^n),1}] \\
&\bar{s}^c : \bar{t}^c \succ a_{n,4}^c \succ \bar{a}_{n,4}^c \succ [A_{\delta(v_n^c),1}] \succ a_{n-1,4}^c \succ \bar{a}_{n-1,4}^c \succ [A_{\delta(v_{n-1}^c),1}] \succ \cdots \succ a_{1,4}^c \\
&\quad \succ \bar{a}_{1,4}^c \succ [A_{\delta(v_1^c),1}]
\end{aligned}
$$

$$
t^c : s^c \succ u^c, \qquad \bar{t}^c : \bar{s}^c \succ \bar{u}^c, \qquad u^c : t^c, \qquad \bar{u}^c : \bar{t}^c
$$

$$
\begin{aligned}
&a_{i,1}^c : a_{i,2}^c \succ s^c \succ a_{i,4}^c, \quad a_{i,2}^c : a_{i,3}^c \succ a_{i,1}^c, \quad a_{i,3}^c : a_{i,4}^c \succ a_{i,2}^c, \quad a_{i,4}^c : a_{i,1}^c \succ \bar{s}^c \succ a_{i,3}^c \\
&\bar{a}_{i,1}^c : \bar{a}_{i,2}^c \succ s^c \succ \bar{a}_{i,4}^c, \quad \bar{a}_{i,2}^c : \bar{a}_{i,3}^c \succ \bar{a}_{i,1}^c, \quad \bar{a}_{i,3}^c : \bar{a}_{i,4}^c \succ \bar{a}_{i,2}^c, \quad \bar{a}_{i,4}^c : \bar{a}_{i,1}^c \succ \bar{s}^c \succ \bar{a}_{i,3}^c
\end{aligned}
$$

In $\mathcal{P}_2$, only the preferences of agents t^c and $t^{\bar{c}}$ change to $u^c \succ s^c$, respectively, $\bar{u}^c \succ \bar{s}^c$. See Figure 1 for an example. Notably, in each of the added 4-cycles, there exist two matchings of the four agents that are stable within the cycle in both $\mathcal{P}_1$ and $\mathcal{P}_2$ (i.e., $\{\{a_{i,1}^c, a_{i,2}^c\}, \{a_{i,3}^c, a_{i,4}^c\}\}$ or $\{\{a_{i,1}^c, a_{i,4}^c\}, \{a_{i,2}^c, a_{i,3}^c\}\}$ and $\{\{\bar{a}_{i,1}^c, \bar{a}_{i,2}^c\}, \{\bar{a}_{i,3}^c, \bar{a}_{i,4}^c\}\}$ or $\{\{\bar{a}_{i,1}^c, \bar{a}_{i,4}^c\}, \{\bar{a}_{i,2}^c, \bar{a}_{i,3}^c\}\}$ for $c \in [\ell]$ and $i \in [\nu]$). Matching M_1 contains for the 4-cycles consisting of agents $\{a_{i,t}^c \mid t \in [4]\}$ the edges $\{a_{i,1}^c, a_{i,2}^c\}$ and $\{a_{i,3}^c, a_{i,4}^c\}$ and for the 4-cycles consisting of agents $\{\bar{a}_{i,t}^c \mid t \in [4]\}$ the edges $\{\bar{a}_{i,1}^c, \bar{a}_{i,4}^c\}$ and $\{\bar{a}_{i,2}^c, \bar{a}_{i,3}^c\}$.

Edge Gadget. For each edge $e = \{v_i^c, v_j^{\hat{c}}\}$, we add an edge gadget. This gadget consists of a 4-cycle with agents $a_{e,1}$, $a_{e,2}$, $a_{e,3}$, and $a_{e,4}$, admitting two different matchings that are stable *within* the gadget in both $\mathcal{P}_1$ and $\mathcal{P}_2$. The matching M_1 contains $\{a_{e,1}, a_{e,4}\}$ and $\{a_{e,3}, a_{e,2}\}$

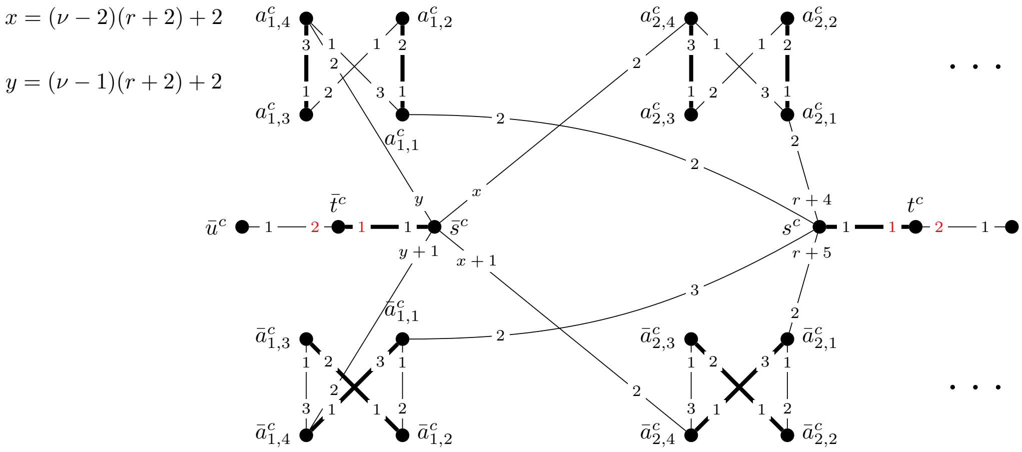


Figure 1 An example for the vertex-selection gadget from Theorem 1. For an edge between two agents a and a', the number x closer to agent a means that a ranks a' at position x, i.e., there are $x-1$ agents which a prefers to a'. For example, the preferences of $a^c_{1,4}$ are $a^c_{1,1} \succ \bar{s}^c \succ a^c_{1,3}$. The depicted preferences are those in $\mathcal{P}_1$. The preferences in $\mathcal{P}_2$ arise from swapping the red numbers. The matching M_1 is marked in bold.

in this 4-cycle and remains stable in M_2 if all of s^c, $\bar{s}^c$, $s^{\hat{c}}$, and $\bar{s}^{\hat{c}}$ are matched at least as good as the respective agents corresponding to v^c_i and $v^{\hat{c}}_j$. This notably only happens if the vertex-selection gadgets of V^c and $V^{\hat{c}}$ "select" the endpoints of e. Otherwise, the agents in this component need to be matched as $\{a_{e,1}, a_{e,2}\}$ and $\{a_{e,3}, a_{e,4}\}$ in M_2, thereby contributing four pairs to $M_1 \triangle M_2$. For each $e = \{v^c_i, v^{\hat{c}}_j\} \in E$, the agent's preferences are as follows:

$$a_{e,1} : a_{e,2} \succ s^c \succ \bar{s}^c \succ s^{\hat{c}} \succ \bar{s}^{\hat{c}} \succ a_{e,4}, \qquad a_{e,2} : a_{e,3} \succ a_{e,1},$$
$$a_{e,3} : a_{e,4} \succ a_{e,2}, \qquad a_{e,4} : a_{e,1} \succ a_{e,3}.$$

The Reduction. To complete the description of the parameterized reduction, we set $M_1 := \{\{s^c, t^c\}, \{\bar{s}^c, \bar{t}^c\} \mid c \in [\ell]\} \cup \{\{a^c_{i,1}, a^c_{i,2}\}, \{a^c_{i,3}, a^c_{i,4}\}, \{\bar{a}^c_{i,1}, \bar{a}^c_{i,4}\}, \{\bar{a}^c_{i,3}, \bar{a}^c_{i,2}\} \mid c \in [\ell], i \in [\nu]\} \cup \{\{a_{e,1}, a_{e,4}\}, \{a_{e,3}, a_{e,2}\} \mid e \in E\}$ and $k := \ell \cdot (4\nu + 5) + 4(|E| - \binom{\ell}{2})$.

For the correctness of the reduction one can show that in M_2 for each $c \in [\ell]$ there is some $i^* \in [\nu]$ such that the matching M_2 contains pairs $\{s^c, a^c_{i^*,1}\}$, $\{\bar{s}^c, a^c_{i^*,4}\}$ (this corresponds to selecting vertex $v^c_{i^*}$ for color c). Then, the only agents $a_{e,1}$ for an edge $e \in E$ incident to some vertex from V^c that both s^c and $\bar{s}^c$ do not prefer to their partner in M_2 are those in $A_{\delta(v^c_{i^*}),1}$. This implies that for all edges $e = \{v^c_i, v^{\hat{c}}_j\}$ with both endpoints selected we can match $a_{e,1}$ worse than s^c, $\bar{s}^c$, $s^{\hat{c}}$, and $\bar{s}^{\hat{c}}$. Thus, we can select $\{a_{e,1}, a_{e,4}\}, \{a_{e,2}, a_{e,3}\}$ as in M_1 in the respective edge gadget. In contrast, for all other edges we have to select the other matching in the edge gadget. To upper-bound the overall symmetric difference, one needs to further prove that for all $j < i^*$, matching M_2 contains $\{\{\bar{a}^c_{j,1}, \bar{a}^c_{j,2}\}, \{\bar{a}^c_{j,3}, \bar{a}^c_{j,4}\}\}$, and that for all $j > i^*$, matching M_2 contains $\{\{a^c_{j,1}, a^c_{j,4}\}, \{a^c_{j,2}, a^c_{j,3}\}\}$. Thus, independent of the selected vertex, each vertex-selection gadget contributes $4\nu + 5$ pairs to $M_1 \triangle M_2$.

3.2 XP-Algorithm

Complementing the above W[1]-hardness result, we now sketch an intricate XP-algorithm for ISR parameterized by $|\mathcal{P}_1 \oplus \mathcal{P}_2|$, resulting in the following theorem:

▶ **Theorem 2** (⋆). *ISR can be solved in $\mathcal{O}(2^{4|\mathcal{P}_1 \oplus \mathcal{P}_2|} \cdot n^{5|\mathcal{P}_1 \oplus \mathcal{P}_2|+3})$ time.*

Our algorithm works in two phases: In the initialization phase, we make some guesses how the stable matching M_2 looks like and accordingly change the original stable matching M_1. These changes and guesses then impose certain constraints how good/bad some agents must be matched in M_2. Subsequently, in the propagation phase, we locally resolve blocking pairs caused by the initial changes by "propagating" these constraints through the instance until a new stable matching is reached. This matching is then guaranteed to be as close as possible to the original stable matching. We believe that our technique to propagate changes through a matching is also of independent interest and might find applications elsewhere. In the following, we sketch the main ingredients of our algorithm. We say that we *update a matching M to contain a pair $e = \{a, b\}$* if we delete all pairs containing a or b from M and add pair e to M.

Initialization Phase (First Part of Description). Our algorithm maintains a matching M. At the beginning, we set $M := M_1$. Before we change M, we make some guesses on how the output matching M_2 shall look like. These guesses are responsible for the exponential part of the running time (the rest of our algorithm runs in polynomial time). The guesses result in some changes to M and, for some agents $a \in A$, in a "best case" and "worst case" to which partner a can be matched in M_2. Consequently, we will store in $\mathrm{bc}(a)$ the *best case* how a may be matched in M_2, i.e., the most-preferred (by a in $\mathcal{P}_2$) agent b for which we cannot exclude that a is matched to b in a stable matching in $\mathcal{P}_2$ respecting the guesses. Similarly, $\mathrm{wc}(a)$ stores the *worst case* to which a can be matched. We initialize $\mathrm{bc}(a) = \mathrm{wc}(a) = \perp$ for all $a \in A$, encoding that we do not know a best or worst case yet.

To be more specific, among others, in the initialization phase we guess for each agent $a \in A$ with modified preferences as well as for $M_1(a)$ how they are matched in M_2 and update M to include the guessed pairs. Moreover, as an unmatched agent a shall always have $\mathrm{bc}(a) \neq \perp$ or $\mathrm{wc}(a) \neq \perp$, we guess for all agents a that became unmatched by this whether they prefer $M_1(a)$ to $M_2(a)$ (in which case we set $\mathrm{bc}(a) := M_1(a)$) or $M_2(a)$ to $M_1(a)$ (in which case we set $\mathrm{wc}(a) := M_1(a)$). Our algorithm also makes further guesses in the initialization phase. However, in order to understand the purpose of these additional guesses, it is helpful to first understand the propagation phase in some detail. Thus, we postpone the description of the additional guesses to the end of this section.

Propagation Phase. After the initialization phase, blocking pairs for the current matching M force the algorithm to further change M and force a propagation of best and worst cases through the instance until a stable matching is reached. As our updates to M are in some sense "minimally invasive" and exhaustive, once M is stable in $\mathcal{P}_2$, it is guaranteed to be the stable matching in $\mathcal{P}_2$ which is closest to M_1 among all matchings respecting the initial guesses. At the core of the technique lies the simple observation that in an SR instance for each stable pair $\{c, d\}$ and each stable matching N not including $\{c, d\}$ exactly one of c and d prefers the other to its partner in N:

▶ **Lemma 3** ([26, Lemma 4.3.9]). *Let N be a stable matching and $e = \{c, d\} \notin N$ be a stable pair in an SR instance. Then either $N(c) \succ_c d$ and $c \succ_d N(d)$ or $d \succ_c N(c)$ and $N(d) \succ_d c$.*

From this we can draw conclusions in the following spirit: Assuming that for a stable pair $\{c, d\}$ in $\mathcal{P}_2$ we have that $\mathrm{wc}(c) \succ_c d$, i.e., c is matched better than d in M_2, it follows from Lemma 3 that d is matched worse than c in M_2, implying that we can safely set $\mathrm{bc}(d) = c$.

Algorithm 1 Simplified propagation step performed for a pair $\{a, b\}$ of two matched agents blocking M with $\mathrm{bc}(b) = M(b)$ or for an unmatched agent a with $\mathrm{wc}(a) \neq \bot$.

```
1: if a is unmatched then Let $e = \{a, c\}$ be the stable pair in $\mathcal{P}_2$ such that $c \succ_a^{\mathcal{P}_2} \mathrm{wc}(a)$
   and $\mathrm{bc}(c) \succ_c^{\mathcal{P}_2} a$ (or $\mathrm{bc}(c) = \bot$) and $c$ is worst-ranked by $a$ among all such pairs.
2: else Let $e = \{a, c\}$ be the stable pair in $\mathcal{P}_2$ such that $c \succ_a^{\mathcal{P}_2} M(a)$ and $\mathrm{bc}(c) \succ_c^{\mathcal{P}_2} a$ (or
   $\mathrm{bc}(c) = \bot$) and $c$ is worst-ranked by $a$ among all such pairs.
3: if no such pair exists then Reject this guess.
4: else Update $M$ such that it contains $e$, set $\mathrm{wc}(a) := c$ and $\mathrm{bc}(c) := a$.
5:     if $M(a) \neq \square$ then $\mathrm{bc}(M(a)) := a$.
6:     if $M(c) \neq \square$ then $\mathrm{wc}(M(c)) := c$.
```

In the following, we will now explain simplified versions of some parts of the propagation phase, while leaving out others (in fact, for the full algorithm and proof of correctness an extensive analysis is needed; see our full version).

Assume for a moment that the current matching M is perfect and that there is a blocking pair for M in $\mathcal{P}_2$ (see Algorithm 1 for a pseudocode-description of the following procedure). Because M_1 is stable in $\mathcal{P}_1$, all pairs that currently block M_1 either involve an agent with changed preferences or resulted from previous changes made to M. Using this, one can show that at least one of the two agents from a blocking pair $\{a, b\}$, say b, will have $a \succ_b \mathrm{bc}(b) = M(b)$. Thus, we know that b is matched worse than a in any stable matching in $\mathcal{P}_2$ respecting our current guesses. Accordingly, for $\{a, b\}$ not to block M_2, agent a has to be matched to b or better and, in particular, better than $M(a)$ in the solution. As a consequence, we update the worst case of a to be the next agent c which a prefers to $M(a)$ such that $\{a, c\}$ is a stable pair in $\mathcal{P}_2$, i.e., we set $\mathrm{wc}(a) := c$ (see Line 4). This change is then further propagated through the instance. Note that from Lemma 3 it follows that if $\{a', a''\}$ is a stable pair in $\mathcal{P}_2$ and agent a'' is the worst possible partner of a' in a stable matching in $\mathcal{P}_2$ (or a' prefers its worst possible partner to a''), then agent a'' cannot be matched better than agent a' in a stable matching in $\mathcal{P}_2$. Thus, by setting $\mathrm{wc}(a') := a''$ we also get $\mathrm{bc}(a'') := a'$. Consequently, applying this to our previous update $\mathrm{wc}(a) = c$, we can also set $\mathrm{bc}(c) := a$ (see Line 4). Moreover, recall that a prefers c to a's current partner $M(a)$ in M. Thus, assuming that in a stable matching M^* in $\mathcal{P}_2$ one of a and $M(a)$ prefers the other to its partner in M^* and the other prefers its partner in M^*, we can use the same argument again and set $\mathrm{bc}(M(a)) := a$ (see Line 5; we will discuss in the last paragraph in this section why this assumption can be made). For the update in Line 6 a similar reasoning applies.

So far we assumed that all agents are matched (which indeed needs to be the case for M_2 because we can delete all agents not matched by M_2 in a preprocessing step). Using this, whenever there is an unmatched agent a, one can show that it cannot be matched to $\mathrm{bc}(a)$ or $\mathrm{wc}(a)$. Thus, if $\mathrm{wc}(a) \neq \bot$, then we match a to the next-better agent c before $\mathrm{wc}(a)$ in its preferences such that $\{a, c\}$ is a stable pair in $\mathcal{P}_2$ and set $\mathrm{wc}(a) = c$. Subsequently, we propagate this change as in the above described case of a blocking pair (see Algorithm 1). Otherwise, we have $\mathrm{bc}(a) \neq \bot$ and we match a to the next-worse agent b after $\mathrm{bc}(a)$ in the preferences of a such that $\{a, b\}$ is a stable pair in $\mathcal{P}_2$. Here, a slightly more complicated subsequent propagation step is needed (as described in our full version).

Repeating these steps, i.e., matching so far unmatched agents and resolving blocking pairs, eventually either results in a conflict (i.e., an agent preferring its worst case to its best case, or changing a pair which we guessed to be part of M_2) or in a stable matching. In the first case, we conclude that no stable matching obeying our guesses exists, while in the latter case, we found a stable matching obeying the guesses and minimizing the symmetric difference to M_1 among all such matchings. The reason for the optimality of this matching is that every matching obeying our initial guesses has to obey the best and worst cases at the termination of the algorithm, and the computed matching M contains all pairs from M_1 which comply with the best and worst cases.

Initialization Phase (Second Part of Description). In addition to the guesses described above, the algorithm guesses the set F of pairs from M_1 for which both endpoints prefer M_2 to M_1. Similarly, the algorithm guesses the set H of pairs from M_2 for which both endpoints prefer M_1 to M_2. Notably, one can prove that the cardinality of both F and H can be upper-bounded by $|\mathcal{P}_1 \oplus \mathcal{P}_2|$, ensuring XP-running time. The reason why we need to guess the set F is that pairs $\{a, b\}$ from M_1 may not be stable pairs in $\mathcal{P}_2$. In this case, a preferring $M(a)$ to b does not imply that b prefers a to $M(b)$. Thus, if we would treat the pairs from F as "normal" pairs, we would propagate an incorrect worst case in Line 5. Note that all pairs from $M \setminus M_1$ are stable pairs in $\mathcal{P}_2$, as we only add pairs that are stable over time (see Line 4). The reason why we need to guess the set H is more subtle but also due to fact that pairs from H might cause problems for our propagation step (see our full version). To incorporate our guesses, for each $\{a, b\} \in F$, we delete $\{a, b\}$ from M and set $\text{wc}(a) = b$ and $\text{wc}(b) = a$, while for each $\{a, b\} \in H$ we update M to include H. We remark that from the proof of Theorem 1 it follows that ISR is NP-complete even if we know for each agent a whose preferences changed as well as $M_1(a)$ how they are matched in M_2 and the set of pairs $F \subseteq M_1$ for which both endpoints prefer M_2 to M_1. This indicates that guessing the set H might be necessary for an XP-algorithm.

4 Incremental Stable Marriage with Ties Parameterized by the Number of Ties

Bredereck et al. [7] raised the question how the total number of ties influences the computational complexity of ISM-T. Note that the number of ties in a preference relation is the number of equivalence classes of the relation containing more than one agent. For instance the preference relation $a \sim b \sim c \succ d \sim e \succ f$ contains two ties, where the first tie has size three and the second tie has size two. In this section, following a fundamentally different and significantly simpler path than Bredereck et al., we show that their W[1]-hardness result for ISM-T parameterized by k for $|\mathcal{P}_1 \oplus \mathcal{P}_2| = 1$ still holds if we parameterize by k *plus* the number of ties. To prove this, we introduce a natural extension of ISM called Incremental Stable Marriage with Forced Edges (ISMFE). ISMFE differs from ISM in that as part of the input we are additionally given a subset $Q \subseteq M_1$ of the initial matching, and the question is whether there is a stable matching M_2 for the changed preferences with $|M_1 \triangle M_2| \leq k$ containing all pairs from Q, i.e., $Q \subseteq M_2$.

We first show that ISMFE with ties is intractable even if $|Q| = 1$ by reducing from a W[1]-hard local search problem related to finding a perfect stable matching with ties [37]:

▶ **Proposition 4** ($\star$). *ISMFE with ties parameterized by k and the summed number of ties in $\mathcal{P}_1$ and $\mathcal{P}_2$ is W[1]-hard, even if $|\mathcal{P}_1 \oplus \mathcal{P}_2| = 1$ and $|Q| = 1$ and only women have ties in their preferences.*

Second, we reduce ISMFE with ties to ISM-T. The general idea of this parameterized reduction is to replace a forced pair $\{m, w\} \in Q$ by a gadget consisting of $6(k + 1)$ agents. In M_1, we match the agents from the gadget in a way such that if m and w are matched differently in M_2, then, compared to M_1, the matching in the whole gadget needs to be changed, thereby exceeding the given budget k. This reduction implies:

▶ **Theorem 5** (⋆)**.** *ISM-T parameterized by k and the summed number of ties in $\mathcal{P}_1$ and $\mathcal{P}_2$ is W[1]-hard, even if $|\mathcal{P}_1 \oplus \mathcal{P}_2| = 1$ and only women have ties in their preferences.*

We remark that our reduction also implies the W[1]-hardness of ISM-T parameterized by k if each tie has size at most two and $|\mathcal{P}_1 \oplus \mathcal{P}_2| = 1$.

On the algorithmic side, parameterized by the number of agents with at least one tie in their preferences in $\mathcal{P}_2$, ISM-T lies in XP. The idea of our algorithm is to first guess the partners of all agents in M_2 with a tie in their preferences in $\mathcal{P}_2$ and subsequently reduce the problem to an instance of Weighted Stable Marriage, which is polynomial-time solvable [17]. Note that parameterizing by the summed size of all ties results in fixed-parameter tractability, as we can iterate over all possibilities of breaking the ties and subsequently apply the algorithm for ISM.

▶ **Proposition 6** (⋆)**.** *ISM-T parameterized by the number of agents with at least one tie in their preferences in $\mathcal{P}_2$ lies in XP. ISM-T parameterized by the summed size of all ties in $\mathcal{P}_2$ is fixed-parameter tractable.*

5 Master Lists

After having shown in the previous section that ISM-T and ISR mostly remain intractable even if we restrict several problem-specific parameters, in this section we analyze the influence of the structure of the preference profiles by considering what happens if the agents' preferences are similar to each other. The arguably most popular approach in this direction is to assume that there exists a single central order (called master list) and that all agents derive their preferences from this order. This approach has already been applied to different stable matching problems in the quest for making them tractable [8, 13, 29, 30]. Specifically, we analyze in Section 5.1 the case where the preferences of all agents follow a single master list, in Section 5.2 the case where all but few agents have the same preference list, and in Section 5.3 the case where each agent has one of few different preference lists (which generalizes the setting considered in Section 5.2).

5.1 One Master List

In an instance of Stable Marriage/Roommates with agent set A, we say that the preferences of agent $a \in A$ can be *derived* from some preference list $\succsim^*$ over agents A if the preferences of a are $\succsim^*$ restricted to $\mathrm{Ac}(a)$. If the preferences of all agents in $\mathcal{P}_2$ can be derived from the same strict preference list (which is typically called *master list*), then there is a unique stable matching in $\mathcal{P}_2$ which iteratively matches the so-far unmatched top-ranked agent in the master list to the highest ranked agent it accepts:

▶ **Observation 7.** *If all preferences in $\mathcal{P}_2$ can be derived from the same strict preference list, then ISR can be solved in linear time.*

This raises the question what happens when the master list is not a strict but a weak order. If the preferences of the agents may be incomplete, then reducing from the NP-hard Weakly Stable Pair problem (the question is whether there is a stable matching in an SM-T/SR-T instance containing a given pair [29, Lemma 3.4]), one can show that even assuming that all preferences are derived from a weak master list is not sufficient to make ISM-T or ISR-T polynomial-time solvable.

▶ **Observation 8** (⋆)**.** *ISM-T and ISR-T are NP-hard even if all preferences in $\mathcal{P}_1$ and $\mathcal{P}_2$ can be derived from the same weak preference list.*

In contrast to this, if we assume that the preferences of agents in $\mathcal{P}_2$ are complete and derived from a weak master list, then we can solve ISM-T and ISR-T in polynomial time. While for ISM-T this follows from a characterization of stable matchings in such instances as the perfect matchings in a bipartite graph due to Irving et al. [29, Lemma 4.3], for ISR-T this characterization does not directly carry over. Thus, we need a new algorithm: Assume that the master list consists of q ties (possibly containing just one agent) and let $A_i \subseteq A$ be the set of agents from the ith tie for $i \in [q]$. Distinguishing between several cases, we build the matching M_2 by dealing for increasing $i \in [q]$ with each tie separately while greedily maximizing the overlap of the so-far constructed matching with M_1. Our algorithm exploits the observation that in a stable matching, for $i \in [q]$, all agents from A_i are matched to agents from A_i except if (i) $|\bigcup_{j \in [i-1]} A_j|$ is odd in which case one agent from A_i is matched to an agent from A_{i-1}, or (ii) if $|\bigcup_{j \in [i]} A_j|$ is odd in which case one agent from A_i is matched to an agent from A_{i+1}.

▶ **Proposition 9** (⋆). *If the preferences of agents in $\mathcal{P}_2$ are complete and derived from a weak master list, then ISM-T/ISR-T can be solved in polynomial time.*

5.2 Few Outliers

Next, we consider the case that almost all agents derive their complete preferences from a single strict preference list (we will call these agents *followers*), while the remaining agents (we will call those agents *outliers*) have arbitrary preferences. ISR is fixed-parameter tractable with respect to the number of outliers, as we show that all stable matchings in a Stable Roommates instance can be enumerated in FPT time with respect to this parameter:

▶ **Theorem 10** (⋆). *Given a Stable Roommates instance $(A, \mathcal{P})$ and a partitioning $F \uplus S$ of the agents A such that all agents from F have complete preferences that can be derived from the same strict preference list, one can enumerate all stable matchings in $(A, \mathcal{P})$ in $\mathcal{O}(n^2 \cdot |S|^{|S|})$ time. Consequently, ISR is solvable in $\mathcal{O}(n^2 \cdot |S|^{|S|})$ time, where $|S|$ is the number of outliers in $\mathcal{P}_2$.*

If the master list may contains ties, then enumerating stable matchings becomes a lot more complicated, as here we have much more flexibility on how the agents are matched. Thus, we leave it open whether there exists a similar fixed-parameter tractability result for a weak master list (both in the roommates and marriage setting).

5.3 Few Master Lists

Motivated by the positive result from Section 5.2, in this section we consider the smaller parameter “number of different preference lists”. Recall that Observation 7 states that if the preference lists of all agents are derived from a strict master list in a Stable Roommates instance, then there exists only one stable matching (even if the preferences of the agents may be incomplete). This raises the question what happens if there exist “few” master lists and each agent derives its preferences from one of the lists. To the best of our knowledge, the parameter “number of master lists” has not been considered before. However, it nicely complements (and lower-bounds) the parameter “number of agent types” as studied by Meeks and Rastegari [38]. Two agents are of the same type if they have the same preferences and all other agents are indifferent between them. Notably, Boehmer et al. [5, Proposition 5] proved that ISM-T is fixed-parameter tractable with respect to the number of agent types. Their algorithm also works for ISR-T.

If the preferences of agents are incomplete, then as proven in Observation 8, ISM-T is already NP-hard for just one weak master list. Moreover, note that a reduction of Cseh and Manlove [12, Theorem 4.2] implies that ISR with incomplete preferences is NP-hard even if the preferences of each agent are derived from one of two weak preference lists. Consequently, in this subsection we focus on the case with complete preferences.

In contrast to the two fixed-parameter tractability results for the number of outliers (Theorem 10) and the number of agent types [5], we show that parameterized by the number p of master lists, ISR is W[1]-hard even if the preferences of agents are complete:

▶ **Theorem 11** (⋆). *ISR is W[1]-hard parameterized by the minimum number p such that in $\mathcal{P}_2$ the preferences of each agent can be derived from one of p strict preference lists, even if in $\mathcal{P}_1$ as well as in $\mathcal{P}_2$ all agents have complete preferences.*

Containment of this problem in XP is an intriguing open question; in other words, is there a polynomial-time algorithm if the number of master lists is constant?

Recalling that ISM-T is polynomial-time solvable if agents have complete preferences derived from one weak master list (Proposition 9), we now ask the same question for ISM-T. Using a similar but slightly more involved reduction than for Theorem 11, we show that this problem is W[1]-hard with respect to the number of master lists.

▶ **Theorem 12** (⋆). *ISM-T is W[1]-hard parameterized by the minimum number p such that in $\mathcal{P}_2$ the preferences of each agent can be derived from one of p weakly ordered preference lists, even if in $\mathcal{P}_1$ as well as in $\mathcal{P}_2$ all agents have complete preferences.*

Again, it remains open whether ISM-T for a constant number of master lists is polynomial-time solvable or NP-hard.

6 Conclusion

Among others, answering two open questions of Bredereck et al. [7], we have contributed to the study of the computational complexity of adapting stable matchings to changing preferences. From a broader algorithmic perspective, in particular, the “propagation” technique from our XP-algorithm for the number of swaps, and the study of the number of different preference lists/master lists as a new parameter together with the needed involved constructions for the two respective hardness proofs could be of interest.

There are several possibilities for future work. As direct open questions, for the parameterization by the number of outliers, we do not know whether ISM-T or ISR-T are fixed-parameter tractable. Moreover, it remains open whether ISR or ISM-T with complete preferences is polynomial-time solvable for a constant number of master lists.

Finally, it would also be interesting to analyze a variation of ISR or ISM where the matching in $\mathcal{P}_1$ is not given, i.e., we have to find two matchings M_1 and M_2 with $|M_1 \triangle M_2| \leq k$ such that M_1 is stable in $\mathcal{P}_1$ and M_2 is stable in $\mathcal{P}_2$. Notably, this is a special case of a multistage [16, 25] variant of stable matching problems and Chen et al. [10] already proved that this problem is NP-hard for $k = 0$ in the bipartite case.

References

1 Mohammad Akbarpour, Shengwu Li, and Shayan Oveis Gharan. Thickness and information in dynamic matching markets. *J. Political Econ.*, 128(3):783–815, 2020. 2

2 Mariagiovanna Baccara, SangMok Lee, and Leeat Yariv. Optimal dynamic matching. *Theor. Econ.*, 15(3):1221–1278, 2020. 2

3 Sayan Bhattacharya, Martin Hoefer, Chien-Chung Huang, Telikepalli Kavitha, and Lisa Wagner. Maintaining near-popular matchings. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP '15)*, pages 504–515. Springer, 2015. 1, 3

4 Niclas Boehmer, Robert Bredereck, Klaus Heeger, and Rolf Niedermeier. Bribery and control in stable marriage. *J. Artif. Intell. Res.*, 71:993–1048, 2021. 3

5 Niclas Boehmer, Klaus Heeger, and Rolf Niedermeier. Theory of and experiments on minimally invasive stability preservation in changing two-sided matching markets. *CoRR*, abs/2112.05777, 2021. Accepted at *AAAI'22*. `arXiv:2112.05777`. 2, 4, 12, 13

6 Niclas Boehmer and Rolf Niedermeier. Broadening the research agenda for computational social choice: Multiple preference profiles and multiple solutions. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '21)*, pages 1–5. ACM, 2021. 1

7 Robert Bredereck, Jiehua Chen, Dušan Knop, Junjie Luo, and Rolf Niedermeier. Adapting stable matchings to evolving preferences. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1830–1837. AAAI Press, 2020. Full version available at arxiv.org/abs/1907.01375. 2, 3, 4, 5, 10, 13

8 Robert Bredereck, Klaus Heeger, Dušan Knop, and Rolf Niedermeier. Multidimensional stable roommates with master list. In *Proceedings of the 16th International Conferenc of the Web and Internet Economics (WINE '20)*, pages 59–73. Springer, 2020. 11

9 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. 1

10 Jiehua Chen, Rolf Niedermeier, and Piotr Skowron. Stable marriage with multi-modal preferences. In *Proceedings of the 2018 ACM Conference on Economics and Computation (EC '18)*, pages 269–286. ACM, 2018. 13

11 Jiehua Chen, Piotr Skowron, and Manuel Sorge. Matchings under preferences: Strength of stability and tradeoffs. *ACM Trans. Economics and Comput.*, 9(4):20:1–20:55, 2021. 3

12 Ágnes Cseh and David F. Manlove. Stable marriage and roommates problems with restricted edges: Complexity and approximability. *Discret. Optim.*, 20:62–89, 2016. 2, 13

13 Lin Cui and Weijia Jia. Cyclic stable matching for three-sided networking services. *Comput. Networks*, 57(1):351–363, 2013. 11

14 Ettore Damiano and Ricky Lam. Stability in dynamic matching markets. *Games Econ. Behav.*, 52(1):34–53, 2005. 2

15 Laura Doval. Dynamically stable matching. *CoRR*, abs/1906.11391v5, 2021. `arXiv:1906.11391`. 2

16 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP '14)*, pages 459–470. Springer, 2014. 1, 13

17 Tomás Feder. A new fixed point approach for stable networks and stable marriages. *J. Comput. Syst. Sci.*, 45(2):233–284, 1992. 2, 11

18 Itai Feigenbaum, Yash Kanoria, Irene Lo, and Jay Sethuraman. Dynamic matching in school choice: Efficient seat reallocation after late cancellations. *Manag. Sci.*, 66(11):5341–5361, 2020. 1, 3

19 Tamás Fleiner, Robert W. Irving, and David F. Manlove. Efficient algorithms for generalized stable marriage and roommates problems. *Theor. Comput. Sci.*, 381(1-3):162–176, 2007. 5

20 Karthik Gajulapalli, James A. Liu, Tung Mai, and Vijay V. Vazirani. Stability-preserving, time-efficient mechanisms for school choice in two rounds. In *Proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '20)*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. 3

21 Begum Genc, Mohamed Siala, Barry O'Sullivan, and Gilles Simonin. Finding robust solutions to stable marriage. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI '17)*, pages 631–637. ijcai.org, 2017. 3

22 Begum Genc, Mohamed Siala, Barry O'Sullivan, and Gilles Simonin. Robust stable marriage. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI '17)*, pages 4925–4926. AAAI Press, 2017. 3

23 Begum Genc, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. Complexity study for the robust stable marriage problem. *Theor. Comput. Sci.*, 775:76–92, 2019. 3

24 Michel X. Goemans and Francisco Unda. Approximating incremental combinatorial optimization problems. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization (APPROX/RANDOM 2017)*, pages 6:1–6:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. 1

25 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP '14)*, pages 563–575. Springer, 2014. 13

26 Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem – Structure and Algorithms.* Foundations of computing series. MIT Press, 1989. 8

27 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms (invited talk). In *Proceedings of the 1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND '22)*, pages 1:1–1:47. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. 1

28 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theor. Comput. Sci.*, 494:86–98, 2013. 1

29 Robert W. Irving, David F. Manlove, and Sandy Scott. The stable marriage problem with master preference lists. *Discret. Appl. Math.*, 156(15):2959–2977, 2008. 11, 12

30 Naoyuki Kamiyama. Many-to-many stable matchings with ties, master preference lists, and matroid constraints. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19)*, pages 583–591. IFAAMAS, 2019. 11

31 S. Kumar and P. Gupta. An incremental algorithm for the maximum flow problem. *J. Math. Model. Algorithms*, 2(1):1–16, 2003. 1

32 Adam Kunysz. An algorithm for the maximum weight strongly stable matching problem. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC '18)*, pages 42:1–42:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. 5

33 Adam Kunysz, Katarzyna E. Paluch, and Pratik Ghosal. Characterisation of strongly stable matchings. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*, pages 107–119. SIAM, 2016. 5

34 Ce Liu. Stability in repeated matching markets. *CoRR*, abs/2007.03794v2, 2021. `arXiv:2007.03794`. 2

35 Junjie Luo, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Parameterized dynamic cluster editing. *Algorithmica*, 83(1):1–44, 2021. 1

36 Tung Mai and Vijay V. Vazirani. Finding stable matchings that are robust to errors in the input. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA '18)*, pages 60:1–60:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. 3

37 Dániel Marx and Ildikó Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties. *Algorithmica*, 58(1):170–187, 2010. 10

38 Kitty Meeks and Baharak Rastegari. Solving hard stable matching problems involving groups of similar agents. *Theor. Comput. Sci.*, 844:171–194, 2020. 3, 12

39 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003. 5

40 G. Ramalingam and Thomas W. Reps. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms*, 21(2):267–305, 1996. 1

Tree Exploration in Dual-Memory Model

Dominik Bojko
Department of Fundamentals of Computer Science,
Wroclaw University of Science and Technology, Poland

Karol Gotfryd
Department of Fundamentals of Computer Science,
Wroclaw University of Science and Technology, Poland

Dariusz R. Kowalski
School of Computer and Cyber Sciences, Augusta University, GA, USA

Dominik Pająk
Department of Pure Mathematics, Wroclaw University of Science and Technology,
Infermedica, Poland

Abstract

We study the problem of online tree exploration by a deterministic mobile agent. Our main objective is to establish what features of the model of the mobile agent and the environment allow linear exploration time. We study agents that, upon entering a node, do not receive as input the edge via which they entered. In such model, deterministic memoryless exploration is infeasible, hence the agent needs to be allowed to use some memory. The memory can be located at the agent or at each node. The existing lower bounds show that if the memory is either only at the agent or only at the nodes, then the exploration needs superlinear time. We show that tree exploration in dual-memory model, with constant memory at the agent and logarithmic in the degree at each node is possible in linear time when one of the two additional features is present: fixed initial state of the memory at each node (so called clean memory) or a single movable token. We present two algorithms working in linear time for arbitrary trees in these two models. On the other hand, in our lower bound we show that if the agent has a single bit of memory and one bit is present at each node, then the exploration may require quadratic time even on paths, if the initial memory at nodes could be set arbitrarily (so called dirty memory). This shows that having clean node memory or a token allows linear time exploration of trees in the dual-memory model, but having neither of those features may lead to quadratic exploration time even on a simple path.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Online algorithms

Keywords and phrases Graph exploration, agent, memory, tree, deterministic algorithms, lower bound

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.22

Related Version *Full Version*: `https://arxiv.org/abs/2112.13449` [6]

Funding *Dominik Pająk*: Supported by Polish National Science Centre grant UMO-2019/33/B/ST6/02988.

1 Introduction

Consider a mobile entity deployed inside an undirected graph with the objective to visit all its nodes without any a priori knowledge of the topology or size of the graph. This problem, called a graph exploration, is among the basic problems investigated in the context of a mobile agent in a graph [1, 10, 24, 26, 33] and has applications to robot navigation [35] and searching the World Wide Web [7]. In this paper we focus on tree exploration by a deterministic agent. Clearly, to explore the whole tree of n nodes, any agent needs time $\Omega(n)$. A question arises:

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 22; pp. 22:1–22:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

"What are the minimum agent capabilities required to complete the tree exploration in linear time?" Many practical applications may not allow the agent to easily backtrack its moves due to technical, security or privacy reasons. Thus, in this work we assume that the agent, upon entering a node, does ***not*** receive any information about the edge via which it entered.

We consider three components of the input to the agent: memory at the agent, memory at each node, and movable tokens (cf. e.g. [4, 3, 10, 11]). A deterministic agent given some input always chooses the same outgoing edge. Hence at a node of degree d needs at least d different inputs in order to be able to choose all of d outgoing edges. Thus in order to ensure the sufficient number of possible inputs, the number of possible values of the memory and possible present or absent states of the tokens, must exceed d. Hence, the sum of the memory at the agent, memory at the node and the number of tokens must be at least $\lceil \log d_v \rceil$ bits, for any node of degree d_v, where log denotes base-2 logarithm (cf. [34, Observation 1.1]).

In many applications, mobile agents are meant to be small, simple and inexpensive devices or software, cf. [11, 24]. These restrictions limit the size of the memory with which they can be equipped. Thus it is crucial to analyse the performance of the agent equipped with the minimum size of the memory. In this paper we assume the asymptotically minimum necessary memory size of $O(\log d_v)$, where only a constant number of bits is stored at the agent and logarithmic number (in the degree) at each node.

We consider two additional features of the model, namely – clean memory or a single token, and we show that each of these assumptions alone allows linear-time tree exploration. We also prove that when both features are absent, then the linear time exploration with small memory may be impossible even on a simple path.

2 Model

The agent is located in an initially unknown tree $T = (V, E)$ with $n = |V|$ nodes, where n is not known to the agent. The agent can traverse one of the edges incident to its current location within a single step, in order to visit a neighbouring node at its other end. The nodes of the tree are unlabelled, however in order for the agent to locally distinguish the edges outgoing from its current position, we assume that the tree is port-labelled. This means that at each node v with some degree d_v, its outgoing edges are uniquely labelled with numbers from $\{1, 2, \ldots, d_v\}$. Throughout the paper we assume that the port labels are assigned by an adversary that knows the algorithm used by the agent in advance and wants to maximize the exploration time.

Memory. The agent is endowed with some number of memory bits (called *agent memory* or *internal memory*), which it can access and modify. In our results, we assume that the agent has $O(1)$ bits of memory.

Each node $v \in V$ contains some number of memory bits as well, which can be modified by the agent when visiting that node. We will call these bits *node memory* or *local memory*. We assume that node v contains $\Theta(\log d_v)$ bits of memory. Hence, each node is capable of storing a constant number of pointers to its neighbours in the tree.

General Exploration Algorithm. Upon entering a node v, the agent a receives, as input, its current state s_a, the state s_v of the current node v, the number tok_v of tokens at v, the number tok_a of tokens at the agent and the degree d_v of the current node. It outputs its new state s'_a, new state s'_v of node v, new state $\text{tok}'_a, \text{tok}'_v$ of the tokens at the agent/node, respectively, and port p_{out} via which it exits node v; hence, the algorithm defines a transition:

$$(s_a, s_v, \mathrm{tok}_a, \mathrm{tok}_v, d_v) \to (s'_a, s'_v, \mathrm{tok}'_a, \mathrm{tok}'_v, p_{\mathrm{out}}),$$

that must satisfy $\mathrm{tok}_a + \mathrm{tok}_v = \mathrm{tok}'_a + \mathrm{tok}'_v$. In models without the tokens, the state transition of the algorithm can be simplified to:

$$(s_a, s_v, d_v) \to (s'_a, s'_v, p_{\mathrm{out}})$$

Starting state. We assume that for a given deterministic algorithm, there is one starting state s of the agent (obtained from the agent memory), in which the agent is at the beginning of (any execution of) the algorithm. The agent can also use this state later (it can transition to s in subsequent steps of the algorithm). The starting location of the agent is chosen by an adversary accordingly to agent's algorithm (to maximize the agent's exploration time).

Our models. In this paper we study the following three models:

In CleanMem, there is a fixed state $\hat{s}$ and each node v is initially in the state $s_v = \hat{s}$, regardless of the degree of the vertex. In this model the agent does not have access to any tokens.

In DirtyMem, the memory at the nodes is in arbitrary initial states, which are chosen by an adversary. In this model the agent does not have access to any tokens.

In Token, the agent is initially equipped with a single token. If the agent holds a token, it can drop it at a node upon visiting that node. When visiting a node, the agent receives as input whether the current node already contains a token. In this case the agent additionally outputs whether it decides to pick up the token, i.e., deduct from tok_v and add to tok_a. Clearly, since the agent has only one token, then after dropping it at some node, it needs to pick it up before dropping it again at some other node. In this model we assume that the initial state of the memory at each node is chosen by an adversary (like in DirtyMem model).

3 Our results

Upper bounds. While a lot of the existing literature focuses on feasibility of exploration, we show that it is possible to complete the tree exploration in the minimum possible linear time using (asymptotically) minimal memory. We show two algorithms in models CleanMem and Token, exploring arbitrary unknown trees in the optimal time $O(n)$ if constant memory is located at the agent and logarithmic memory is located at each node. Our results show that in the context of tree exploration in dual-memory model, the assumption about clean memory (fixed initial state of node memory) can be "traded" for a token.

It is worth noting that in both our algorithms, the agent returns to the starting position and terminates after completing the exploration, which is a harder task than a perpetual exploration. If the memory is only at the agent, exploration of trees with stop at the starting node requires $\Omega(\log n)$ bits of memory [14]; if the agent is only required to terminate at any node then still a superconstant $\Omega(\log \log \log n)$ memory is required [14]; while exploration without stop is feasible using $O(\log \Delta)$ agent memory [14] (where Δ is the maximum degree of a node). All these results assume that the agent could receive the port number via which it entered the current node, while our algorithms operate without this information.

Lower bound. To explore a path with a single bit of memory and with arbitrary initial state at each node (note that $\log d_v = 1$ for internal nodes on the path), one can employ the Rotor-Router algorithm and achieve exploration time of $O(n^2)$ [41] (there is no need for agent memory), which is time and memory optimal in the model with only node memory [34].

We want to verify the hypothesis that dual-memory allows to achieve linear time of exploration of trees. In our lower bound we analyse path exploration with one bit at each node and additional one bit of memory at the agent. We prove that in this setting, in DirtyMem model an exploration of the path requires $\Omega(n^2)$ steps in the worst case. Interestingly, since time $O(n^2)$ in already achievable with only a single bit at each node (and no memory at the agent), our lower bound shows that adding a single bit at the agent does not reduce the exploration time significantly.

3.1 Previous and related work

Only node memory. One approach for exploration using only node memory is Rotor-Router [41]. It is a simple strategy, where upon successive visits to each node, the agent is traversing the outgoing edges in a round-robin fashion. Its exploration time is $\Theta(mD)$ for any graph with m edges and diameter D [2, 41]. It is easy to see that this algorithm can be implemented in port-labelled graphs with zero bits of memory at the agent and only $\lceil \log d_v \rceil$ bits of memory at each node v with degree d_v (note that this is the minimum possible amount of memory for any correct exploration algorithm using only memory at the nodes). Allowing even unbounded memory at each node still leads to $\Omega(n^3)$ time for some graphs, and $\Omega(n^2)$ time for paths [34]. These lower bounds hold even if the initial state of the memory at each node is clean (i.e., each node starts in some fixed state $\hat{s}$ like in CleanMem model). Hence, only node memory is insufficient for subquadratic time exploration of trees.

Only agent memory. When the agent is not allowed to interact with the environment, then such agent, when exploring regular graphs, does not acquire any new information during exploration. Hence such an algorithm is practically a sequence of port numbers that can be defined prior to the exploration process. In the model with agent memory, the agent is endowed with some number of bits of memory that the agent can access and modify at any step. Thus for the case of the path, the lower bound (for unlimited agent memory) can be infered from a lower bound $\Omega(n^{1.51})$ for Universal Traversal Sequences (UTS) [9]. If this number of bits is logarithmic in n (otherwise, the exploration is infeasible [14]), then in this model it is possible to implement UTS (the agent only remembers the position in the sequence). The best known upper bound is $O(n^3)$ [1] and the best known constructive upper bound is $O(n^{4.03})$ [31]. Memory $\Theta(D \log \Delta)$ is sufficient and sometimes required to explore any graph with maximum degree Δ [24]. For directed graphs, memory $\Omega(n \log \Delta)$ at the agent is sometimes required to explore any graph with maximum outdegree Δ, while memory $O(n\Delta \log \Delta)$ is always sufficient [23].

Finite state automata. An agent equipped with only constant number of bits of persistent memory can be regarded as a finite state automaton (see e.g. [8, 24, 25]). Movements of such agent, typically modelled as a finite Moore or Mealy automaton, are completely determined by a state transition function $f(s, p, d_v) = (s', p')$, where s, s' are the agent's states and p, p' are the ports through which the agent enters and leaves the node v. A finite state automaton cannot explore an arbitrary graph in the setting, where the nodes have no unique labels [39]. Fraigniaud et al. [24] showed that for any $\Delta \geqslant 3$ and any finite state agent with k states, one can construct a planar graph with maximum degree Δ and at most $k + 1$ nodes, that cannot be explored by the agent. It is, however, possible in this model to explore (without stop) the trees, assuming that the finite state agent has access to the incoming port number (cf. [14]).

Two types of memory. Sudo et al. [40] consider exploration of general graphs with two types of memory, where both memories may have arbitrary initial states. They show that $O(\log n)$ bits at the agent and at each node allows exploration in time $O(m + nD)$. Cohen et al. [8] studied the problem of exploring an arbitrary graph by a finite state automaton, which is capable of assigning $O(1)$-bit labels to the nodes. They proposed an algorithm, that – assuming the agent knows the incoming port numbers – dynamically labels the nodes with three different labels and explores (with stop) an arbitrary graph in time $O(mD)$ (if the graph can be labelled offline, both the preprocessing stage and the exploration take $O(m)$ steps). They also show that with 1-bit labels and $O(\log \Delta)$ bits of agent memory it is possible to explore (with stop) all bounded-degree graphs of maximum degree Δ in time $O(\Delta^{10}m)$.

Tokens. Deterministic, directed graph exploration in polynomial time using tokens has been considered in [3], where it was shown that a single token is sufficient if the agent has an upper bound on the number of vertices n and $\Theta(\log \log n)$ tokens are sufficient and necessary otherwise. In [4] the authors proposed a probabilistic polynomial time algorithm that allows two cooperating agents without the knowledge of n to explore any strongly connected directed graph. They also proved that for a single agent with $O(1)$ tokens this is not possible in polynomial time in n with high probability. Using one pebble, the exploration with stop requires an agent with $\Omega(\log n)$ bits of memory [25]. The same space bound remains true for perpetual exploration [24]. Disser et al. [15] show that for a single agent with $O(1)$ memory, to explore any undirected anonymous graph (i.e. with unlabelled vertices and port-labelled edges) with n vertices, $\Theta(\log \log n)$ distinguishable pebbles are necessary and sufficient. They proposed an algorithm based on universal exploration sequences (UXS, cf. [30, 38]), which runs in polynomial time and the agent terminates after returning to the starting vertex with all pebbles. For the lower bound they show that an agent with $O((\log n)^{1-\varepsilon})$ bits of memory (for any constant $\varepsilon > 0$) needs $\Omega(\log \log n)$ pebbles to explore all undirected graphs.

Randomized and collaborative exploration. Although we focus on deterministic graph exploration by a single agent, there is a vast body of literature on randomized exploration techniques, see e.g. [1, 12, 18, 19]. A classical and well-studied processes are random walks, where the agent in each step moves to a neighbour chosen uniformly at random (or does not move with some constant probability). Exploration using this method takes expected time $\Omega(n \log n)$ [20] and $O(n^3)$ [21], where for each of these bounds there exists a graph class for which it is tight. Randomness alone cannot ensure linear time of tree exploration, since expected time $\Omega(n^2)$ is required even for paths [32]. However approaches using memory at the agent [28], local information on explored neighbours [5], or local information on degrees [36] have shown that there are many methods to speedup random walks. Finally, to achieve fast exploration, usage of multiple agents is possible, cf. [12, 13, 15, 16, 17, 22, 27, 29, 37].

4 Upper bounds

4.1 Exploration in CleanMem model

To simplify descriptions, let us denote the starting position of the agent as Root. In this section we show an algorithm exploring any tree in $O(n)$ steps using $O(1)$ bits of memory at the agent and $O(\log d_v)$ bits of memory at each node v of degree d_v in CleanMem model. The memory at each node is organized as follows. It contains two port pointers (of $\lceil \log d_v \rceil$ bits each):

- v.parent – at some point of the execution, contains the port number leading to Root,
- v.last – points to the last port taken by the agent during the exploration

and two flags (of 1 bit):

- v.root – indicates whether the node is Root of the exploration,
- v.visited – indicates whether the node has already been visited.

At each node, the initial state of last pointers is 1, initial state of parent is NULL and initial state of each flag is False. The agent's memory contains one variable State that can take one of five possible values: Initial, Roam, Down, Up, Terminated.

For simplicity of the pseudocode we use a flag *parentSet*, which controls if parent is correctly set. This flag does not need to be stored because it is set and accessed in the same round. It is possible to write a more complicated pseudocode without this variable.

In our pseudocode, we use a procedure MOVE(p), in which the agent traverses an edge labelled with port p from its current location. When the agent changes its state to Terminated, it does not make any further moves.

Algorithm's description. For the purpose of the analysis, assume that tree T is rooted at the initial position of the agent (the Root). The main challenge in designing an exploration algorithm in this model is that the agent located at some node v may not know which port leads to the parent of v. Indeed, if the agent knew which port leads to its parent, it could perform a DFS traversal. In Algorithm 1 the agent would traverse the edges corresponding to outgoing ports in order $1, 2, 3, \ldots, d_v$ (skipping the port leading to its parent) and take the edge to its parent after completing the exploration of the subtrees rooted at its current position. Note that it is possible to mark, which outgoing ports have already been traversed by the agent using only a single pointer at each node hence 0 bits at the agent and $\lceil \log d_v \rceil$ bits at each node allow for linear time exploration in this case.

Since in our model, the agent does not know, which port leads to the parent, we need a second pointer parent at each node. To set it correctly, we first observe that in the model CleanMem, using flag visited, it is possible to mark the nodes that have already been visited. Notice that when the agent traverses some edge outgoing from v in the tree for the first time and enters to a node that has already been visited, then this node is certainly the parent of v. We can utilize this observation to establish correctly v.parent pointer using state Down. When entering to a node in this state, we know that the previously taken edge (port number of this edge is stored in v.last) leads to the parent of v. Our algorithm also ensures, that after entering a subtree rooted at v, the agent leaves it once, and the next time it enters this subtree it correctly sets v.parent and in subsequent steps it visits all the nodes of the subtree. Finally, having correctly set pointers parent at all the nodes, allows the agent to efficiently return to the starting node after completing the exploration and flag root allows the agent to terminate the algorithm at the starting node.

▶ **Theorem 1.** *Algorithm 1 explores any tree and terminates at the starting node in $O(n)$ steps in CleanMem model.*

Proof. First note that the algorithm marks the starting node with flag root (line 2). The algorithm never returns to state Initial, hence only this node will be marked with the root flag. The only line, where the agent terminates the algorithm is line 14 hence the agent can only terminate in the starting node. We need to show that the agent will terminate in every tree and before the termination it will visit all the vertices and the time of the exploration will be $O(n)$. Let us denote the starting node as Root and for any node v different from Root, will call the single neighbour of v that is closer to Root as the parent of v.

Algorithm 1 Tree exploration in CleanMem model.

```
   // Agent is at some node $v$ and the outgoing ports are $\{1, 2, \ldots, d_v\}$
1  if $State = Initial$ then
2      $v$.root ← True, State ← Roam, $v$.visited ← True;
3      MOVE($v$.last);
4  else
5      if $State = Down$ then
6          $v$.parent ← $v$.last, State ← Roam;     // mark edge as leading to parent
7          $v.parentSet$ ← True;
8      else
9          $v.parentSet$ ← False
10     if (($State = Roam$ and $d_v = 1$) or $State = Up$ or $v.parentSet = True$) and
        $v.root = False$ and $v.last = d_v$ and $v.parent \neq NULL$ then
11         State ← Up;                  // exploration of this subtree completed
12         MOVE($v$.parent) ;                             // return to the parent
13     else if $State = Up$ and $v.root = True$ and $v.last = d_v$ then
14         State ← Terminated;       // exploration of the whole tree completed
15     else
16         if $State = Roam$ and $v.visited = True$ and $v.parentSet = False$ then
17             State ← Down;
18         else if $State = Up$ then
19             State ← Roam, $v$.last ← $v$.last + 1;
20         else
21             if  $v.visited = True$ then  $v$.last ← $v$.last + 1;
22             else  $v$.visited ← True;
23         MOVE($v$.last);
```

We will show the following claim using induction over the structure of the tree.

▷ Claim 2. Assume that the agent enters to some previously unvisited subtree rooted at v with n_v vertices for the first time in state Roam in step t_s. Then the agent:

C.1 returns to the parent of v for the first time in state Roam (denote by $t_r > t_s$ the step number of the first return from v to its parent),
C.2 goes back to v in the state Down at step $t_r + 1$,
C.3 returns to the parent of v for the second time in state Up (denote by $t_f > t_r + 1$ the step number of the second return from v to its parent),
C.4 visits all the vertices of this subtree and spends $O(n_v)$ steps within time interval $[t_s, t_f]$.

Proof. We will first show it for all the leaves. Then, assuming that the claim holds for all the subtrees rooted at the children of some node v, we will show it for v. To show this claim for any leaf l, consider the agent entering in state Roam to l. Upon the first visit to l, the agent sets the flag l.visited to True and leaves (without changing the state of the agent) with port 1. This proves C.1. In the next step, at the parent of l, a state changes to Down and the agent uses the same port as during the last time in parent of l. Therefore the agent moves back to l (C.2) and sets l.parent ← 1 (line 6). Then the agent changes its state to Up and moves to the parent (lines 11–12). This shows C.3. Node l was visited twice within the considered time steps, which shows C.4. This completes the proof of the claim for all the leaves.

Now, consider any internal node v of the tree (with $d_v > 1$) and assume that the claim holds for all its children. When the agent enters to v for the first time, it sets the flag $v.\mathsf{visited}$ to `True` and moves to its neighbour w (while being in state Roam) via port 1.

Case 1: w is the parent of v. This immediately shows C.1. If the parent of v is not Root, then it has the degree at least 2, hence the agent will evaluate the if-statement in line 10 to `False`. Thus the agent will execute line 16 and change its state to Down. The agent uses the same port as during the previous visit of w, therefore the agent goes back to v (C.2), hence it will correctly set the pointer $v.\mathsf{parent}$ (it will point to the parent of v), which shows C.3.

Case 2: w is a child of v. In this case we enter to a child of v. We have by the inductive assumption (C.1), that the agent will return from w to v in state Roam. Since the agent enters to v in state Roam, then the value of *parentSet* is `False` and the agent executes line 16, transitions to state Down and then line 23 it takes the same edge as during the last visit to v, hence it moves back to node w (C.2). By C.3 we get that the next time the agent will traverses the edge from w to v, it will be in state Up. Then the agent increments pointer $v.\mathsf{last}$ (line 19) and moves to the next neighbour of v (line 23).

Thus the agent either finds the parent or explores the whole subtree rooted at one of its children w in $O(n_w)$ steps (by the inductive assumption C.4). An analogous analysis holds for ports $2, 3, \ldots, d_v$. When the agent returns from the neighbour connected to v via the last port d_v, it is either in state Up or Down (the second case happens if the edge leading from v to its parent has port number d_v). In both cases it executes lines 11 and 12 and leaves to its parent (the pointer to parent is established since the agent had traversed each outgoing edge, hence it moved to its parent and correctly set the parent pointer).

By this way, the agent visits all the subtrees rooted at v's children $w_1, w_2, \ldots, w_{d_v-1}$ (or up to w_{d_v} if $v = \mathsf{Root}$). Hence, the total number of steps for a node $v \neq \mathsf{Root}$ is $O\left(\sum_{i=1}^{d_v-1} n_{w_i}\right) = O(n_v)$ and $v = \mathsf{Root}$ it is $O\left(\sum_{i=1}^{d_v} n_{w_i}\right) = O(n)$. ◁

To complete the proof of Theorem 1 we need to analyse the actions of the agent at Root. Consider the actions of the agent at Root when the Root.last pointer takes values $i = 1, 2, \ldots, d_{\mathsf{Root}}$. Let v_i be the neighbour of Root pointed by port number i at Root. Observe that the agent at Root behaves similarly as in all the other internal nodes (only exception is that the agent will never enter Root in state Down, because by Claim 2 the agent can enter to it only in states Roam or Up, since the Root has no parent). By Claim 2, the agent visits the whole subtrees rooted at these nodes in time proportional to the number of nodes in these subtrees. When the agent returns from node $v_{d_{\mathsf{Root}}}$ in state Up then the algorithm terminates (line 14). The total runtime is proportional to the total number of nodes in the tree. ◀

4.2 Exploration in Token model

Algorithm's description. In Algorithm 2, the agent has a single token, which can be DROP*ped*, TAKE*n* and MOVE*d* (i.e., carried by the agent across an edge of the graph). Moreover, the agent is always in one state from set $\{\mathsf{Initial}, \mathsf{Roam}, \mathsf{RR}, \mathsf{Down}, \mathsf{Up}, \mathsf{Terminated}\}$. Our algorithm ensures, that in states Roam and RR the agent does not hold the token and in the remaining states, it does. In these four states the agent can eventually DROP it. Each node v has degree denoted by d_v, which is part of the input to the agent, when entering to a node. Moreover, the memory at each node v is organized into three variables: $v.\mathsf{last}$ and $v.\mathsf{parent}$ of size $\lceil \log d_v \rceil$ and a flag $v.\mathsf{root} \in \{\mathtt{True}, \mathtt{False}\}$ to mark the Root. We assume that in all vertices, variables last, parent and root have initially any admissible value (if not, then the agent would easily notice it and change such a value). Moreover, when the agent is entering to a node, it can see whether the node contains the token or not.

We would like to perform a similar exploration as in CleanMem model – we will use pointers last and parent as in Algorithm 1. However, the difficulty in this model is that these pointers may have arbitrary initial values. Especially, if the initial value of parent is incorrect, Algorithm 1 may fall into an infinite loop. Hence in this section we propose a new Algorithm 2 that handles dirty memory using a single token. In this algorithm the agent maintains an invariant that the node with the token, and all the nodes on the path from the token's location to the Root are guaranteed to have correctly set parent pointers. To explore new nodes, the agent performs a Rotor-Router (shortly RR) traversal, starting from node v with the token to its neighbour w (by the invariant, the agent chooses w as one of its children, not its parent). During this traversal, the agent is resetting the parent pointers at each node (and cleaning the root flags). The agent is using last as the pointer for the purpose of RR algorithm. The agent does **not** have to reset last as the RR requires no special initialization. Moreover, by the properties of RR the agent does not traverse the same edge twice in the same direction before returning to the starting node. Since the agent starts in the node with the token, it can notice that it completed a traversal. During this traversal each edge is used at most twice (once in each direction). Moreover, each node visited during this traversal has cleaned memory (pointer parent points to NULL and root is set to False). After returning to the node with the token, pointer v.last points at w and w.last points at v. Hence it is possible to traverse this edge and correctly set pointer w.parent maintaining the invariant. After moving the token down, the agent starts the RR procedure again. Since the agent is not cleaning the pointer last, the RR will use different edges than during the previous traversal. Using this we show in the proof of Theorem 2 that our algorithm traverses every edge at most 6 times.

Preliminaries. Let us introduce some notation: a variable Token associated with a vertex currently occupied by the agent, which is 1, when the agent meets the token and 0 otherwise. If Token $= 1$, then the agent can TAKE the token and the agent with the token can DROP it. Let Path(v) denote a set of all vertices, which are on the shortest path from Root to v, excluding Root. Let T_v denote the subtree of T rooted at v, i.e., $(w \in T_v) \equiv (w = v \vee v \in \mathsf{Path}(w))$. Let $T_{v,p}$ denote a subtree of T_v, rooted at a node connected by edge labelled by outport p at vertex v, i.e. if p leads from v to w (where $v \in \mathsf{Path}(w)$), then $T_{v,p} = T_w$ (we do not define such a tree when p directs towards Root). We say that the action is performed away from Root (downwards), if it starts in v and ends in T_v. Otherwise the action is towards Root (upwards). Let Tok(t) and Ag(t) be the positions of the token and the agent, respectively, at the end of the moment t. Let Act(t) be the action performed during the t-th step. Let the variable v.visited(t) $\in \{0, 1\}$ indicate whether v was visited by Algorithm 2 until the moment t. Note that this variable is not stored at the nodes and this notation is only for the analysis.

Additionally, we consider two substates of RR, depending on the value of the Token variable. Substate RR_0 is state RR, if variable Token $= 0$ at the node to which the agent entered in the considered step. Similarly RR_1 indicates that the agent enters in state RR to a node with Token $= 1$. Note that this distinction is only for the purpose of the analysis, and it does not influence the definition of the algorithm. In our analysis, we will call Initial, Roam, RR_0, RR_1, Down, Up, Terminated *actions* as these states (and substates) correspond to different commands executed by the agent (see the pseudocode of Algorithm 2).

Algorithm 2 Tree exploration in Token model.

```
// Agent is at some node $v$ and the outgoing ports are $\{1, 2, \ldots, d_v\}$
if $State = Initial$ then
    Clean(), $v$.last ← 1, DROP;
    // mark the root for termination
    $v$.root ← True;
    State ← Roam;
else if $State = RR$ and $Token = 1$ then
    // substate RR$_1$
    TAKE, State ← Down;
else if $State = RR$ and $Token = 0$ then
    // substate RR$_0$
    Clean();
    Progress() ;        // increment last
else if $State = Down$ then
    DROP, $v$.parent ← $v$.last;
    Progress();
    State ← Roam;
    IfUp() ;     // check if in a leaf
else if $State = Up$ then
    DROP, Progress();
    State ← Roam;
    if $v.last = 1$ and $v.root = True$ then
        State ← Terminated;
    IfUp();
else    // State = Roam
    Clean(), State ← RR;
if $State \neq Terminated$ then
    MOVE($v$.last);

Procedure Clean()
    $v$.root ← False, $v$.parent ← NULL;

Procedure Progress()
    $v$.last ← ($v$.last mod $d_v$) + 1;

Procedure IfUp()
    if $v.parent = v.last$ then
        TAKE, State ← Up;
```

We assume that Initial action is performed at time step 0. Let Root denote the initial position of the agent. Let v.last(t) and v.parent(t) denote, respectively, the values of v.last and v.parent pointers at the end of the moment t. If v is the starting point of the t-th step, then we say that v.last(t) is the outport related to t-th moment. Moreover, v.last($\cdot$) cannot be changed until the node v will be visited for the next time. Each MOVE(v.last) involves traversing an edge outgoing from the current position of the agent via the port indicated by the current value of variable last at the current position.

Properties of the algorithm. Let us define the following set of properties P(t) (the k-th property at moment t is denoted as $P.k(t)$; in this definition we denote Ag(t) $= w$) that describe the structure of the walk and the interactions with the memory at the nodes by an agent performing Algorithm 2.

P.1 During t-th step:
  a. If Down or Roam is performed, the move of the agent is away from Root (downwards).
  b. If Up or RR$_1$ is performed, the move of the agent is towards Root (upwards).

P.2 $w \in T_{\mathsf{Tok}(t)}$.

P.3 If w is visited for the first time at step t (Act(t) $\in$ {Initial, Roam, RR$_0$}), then it is also cleaned, which means that w.parent is set to NULL and w.root is set to False.

P.4 For every $v \in$ Path(Tok(t)), v.parent(t) $\neq$ NULL.

P.5 If w.visited($t-1$) $= 1$ and w.parent($t-1$) $\neq$ NULL, then Act(t) $\notin$ {Roam, RR$_0$}.

P.6 If w.visited(t) $= 1$ and w.parent(t) $\neq$ NULL, then w.parent(t) points towards Root.

P.7 If the state at the end of t-th moment is Up, then T_w is explored.

P.8 If $t > 0$, then for every $x \leq$ Root.last($t-1$), $T_{\mathsf{Root},x}$ is explored before t-th moment.

P.9 If w.parent(t) = NULL, then there do not exist $s_1 < s_2 < t$ such that w.last(s_1) = w.last$(t) \neq w$.last(s_2), where w.parent(s_1) = NULL (w.last$(\cdot)$ cannot be changed to the same value two times before w.parent$(\cdot)$ was established).

P.10 There do not exist $s_1 < s_2 < t$ such that w.last(s_1) = w.last$(t) \neq w$.last(s_2) and w.parent$(s_1) \neq$ NULL (w.last$(\cdot)$ cannot be changed to the same value two times after w.parent$(\cdot)$ was established).

▶ **Lemma 3.** *At time* 0*, Initial phase of Algorithm 2 guarantees* P(0)*.*

▶ **Lemma 4.** *If Algorithm 2 satisfies* P(s) *for all* $s \leq t$*, then it also fulfills* P$(t+1)$*.*

All the omitted proofs from this section are deferred to the full version of the paper [6].

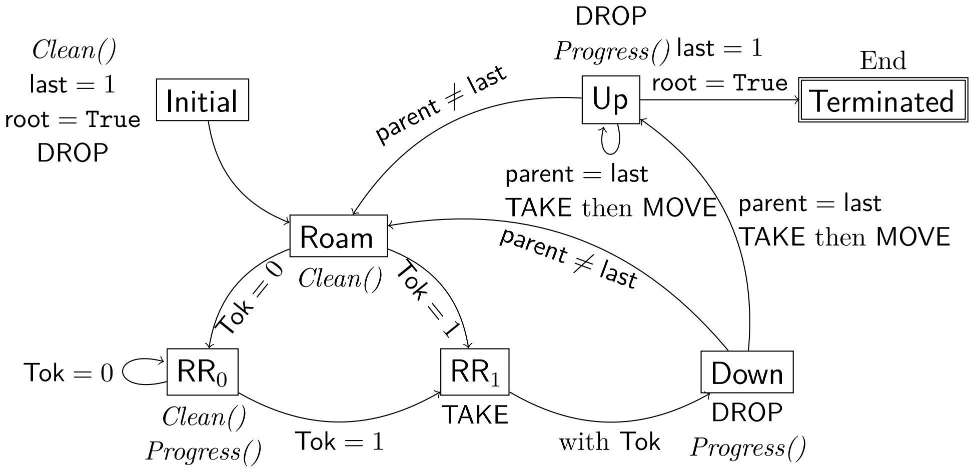


Figure 1 Illustration of actions and state transitions in Algorithm 2.

We will use properties P(t) to show correctness and time complexity of our Algorithm 2.

▶ **Theorem 5.** *Algorithm 2 explores any tree and terminates at the starting node in at most* $6(n-1)$ *steps in the Token model.*

Proof. Using Lemmas 3 and 4, we get that Algorithm 2 satisfies P(t) for each step t. First of all, from P.3, all visited vertices are cleaned upon the first visits.

Consider some vertex v and its arbitrary outport p. If $v \neq$ Root, then from P.9 and P.10, p may be used two times instantly after the incrementation of v.last$(\cdot)$ (once when v.parent$(\cdot)$ = NULL and once when v.parent$(\cdot) \neq$ NULL). Realize that, when v = Root, then from P.2 and the fact, that any vertex with the token cannot be cleaned by RR$_0$ and Roam moves, we claim that Root.parent$(\cdot)$ will always be NULL. By P.9, p may be used once instantly after the incrementation of Root.last$(\cdot)$. Moreover, regardless of the choice of p, it may be used also after Roam or RR$_1$ action without a change of v.last$(\cdot)$.

Case 1. Assume that Act(t) = RR$_1$ is taken via port p. From P.1(t) and P.2(t), p directs upwards towards the token. Then Act$(t+1)$ = Down is performed downwards (from P.1$(t+1)$) via the same edge as p (but with the opposite direction) and changes the position of the token to Tok$(t+1) = v$ and establishes v.parent$(t+1) \leftarrow p$. By P.1, p cannot be used during Roam action. Realize that if $p = v$.last$(s-1)$ for some moment $s > t$ and Act(s) = RR$_1$, then Tok$(s-1)$ is one step towards Root from v (from P.1(s)). Since the token can be moved only during Up and Down actions, then there exists a moment s' such that $t < s' < s$ and $\mathsf{Act}(s') = \mathsf{Up}$ changes the position of the token from v to $\mathsf{Tok}(s)$. However then $\mathsf{Tok}(s).\mathsf{last}(s) \neq \mathsf{Tok}(s).\mathsf{last}(s-1)$ ($\mathsf{Tok}(s)$ is not a leaf) and by P.10, P.6 and P.4, it cannot be changed back since $\mathsf{parent}(\mathsf{Tok}(s)) \neq \mathsf{NULL}$.

Case 2. Assume that $\mathsf{Act}(t) = \mathsf{Roam}$ is taken from node $v = \mathsf{Tok}(t-1)$ via port p (downwards, by P.1(t)). The first action after time t, which leads to the token, is always RR_1. However, as showed before, after RR_1 action there is an instant Down action, which moves the token via p to some vertex w and sets $w.\mathsf{parent} \neq \mathsf{NULL}$. Hence by P.5, Roam action cannot be performed via p once again. By P.1, p cannot be used during RR_1 action.

Our considerations show that each outport can be used at most 3 times. Since each tree of size n has $2(n-1)$ outports, Algorithm 2 terminates after at most $6(n-1)$ moves.

It remains to show that all vertices are then explored. Realize that Algorithm 2 terminates in $(t+1)$-st moment only when the agent is in $\mathsf{Tok}(t)$, $\mathsf{Tok}(t).\mathsf{last}(t) = 1$, $\mathsf{Tok}(t).\mathsf{root} = \mathtt{True}$ and the state is Up at the end of t-th moment. From P.3, we know that the first visit in some v sets root flag to $\mathtt{True}$ in Root and to $\mathtt{False}$ in all other nodes, and this flag does not change (Roam and RR_0 do not clean the vertices with the token, so by P.2, Root cannot be cleaned by these moves). Hence the termination entails $\mathsf{Tok}(t) = \mathsf{Root}$. Since $\mathsf{Root}.\mathsf{parent}(t) = \mathsf{NULL}$, then from P.9 we know that each port p in Root will be set by $\mathsf{Root}.\mathsf{last}(\cdot)$ only once (since the first moment) and P.8 means that each subtree $T_{\mathsf{Root},p}$ for $p \in \{1, \dots, d_{\mathsf{Root}}\}$ were explored before returning to Root, hence the exploration of T_{Root} was completed. ◀

5 Lower bound

In this section we analyse the exploration of paths with one bit of memory at each node and one bit at the agent. We show that in this setting, in the $\mathsf{DirtyMem}$ model, the exploration of the path sometimes requires $\Omega(n^2)$ steps.

Notation. In the following lower bound on the path, we will focus on the actions of the algorithm performed in vertices with degree 2. In such vertices, there are four possible inputs to the algorithm $S = \{(0,0),(0,1),(1,0),(1,1)\}$, where in a pair (a, v), a denotes a bit on the agent and v is a bit saved on the vertex. Let us denote the sets of agent and vertex states by As and Vs, respectively. We also use elements from Ps $= \{0,1\}$ to denote ports in vertices of degree 2. Moreover, a shorthand notation $\overline{b}$ indicates the inverted value of a bit. For example, if $a = 0$, then $\overline{a} = 1$ and vice versa. The goal of this section is to prove:

▶ **Theorem 6.** *Every deterministic algorithm that can explore any path in the* $\mathsf{DirtyMem}$ *model with one bit at the agent and one bit at each node requires time* $\Omega(n^2)$ *to explore some worst-case path with* n *nodes.*

Assume, for a contradiction, that there exists algorithm $\mathcal{A}$, which explores every path in $o(n^2)$ steps. Without loss of generality, we may assume that the adversary always sets the agent in the middle point of the path. For algorithm $\mathcal{A}$, for every $s \in S$, by $A(s)$, $V(s)$, $P(s)$ we denote, respectively, the returned agent state, vertex state and the chosen outport in each vertex with degree 2. Moreover, let $R_3(s) := (A(s), V(s), P(s))$ and $R_2(s) := (A(s), V(s))$. We will show that $\mathcal{A}$ either falls into an infinite loop or performs no faster than Rotor-Router.

The proof of Theorem 6 is divided into several parts. In the first one, we provide several properties of potential algorithms $\mathcal{A}$ that can eventually explore the path in $o(n^2)$ time. These properties are used in the second part, where we prove that the agent does not change its internal state every time, however it has to change the state of the vertex in each step. Further, we check two algorithms (called X and Y), which turn out to explore the path even slower than the Rotor-Router algorithm. The analysis of those two algorithms is quite challenging. Next we show that if three states of $\mathcal{A}$ return the same outport, then either it falls into an infinite loop or explores the path in $\Omega(n^2)$ steps. In the latter part we consider the rest of amenable algorithms, which can either be reduced to previous counterexamples or fall into an infinite loop. A big part of the proof is technical and due to space limitations it is moved to the full version of the paper [6]. Nevertheless, in here we present two parts of the proof to let the reader feel the flavour of high-level ideas and utilized techniques.

▶ **Fact 7.** $(\forall\, a \in \mathrm{As})(\exists\, v \in \mathrm{Vs})\ A(a, v) = \overline{a}$.

Proof. Assume that $(\exists\, a \in \mathrm{As})(\forall\, v \in \mathrm{Vs})\ A(a, v) = a$. If a is the initial state of the agent, then the agent will never change its state before reaching an endpoint of the path. Hence, the time to reach the endpoint cannot be faster than Rotor-Router algorithm, hence the adversary can initialize ports in such a way that the endpoint will be reached after $\Omega(n^2)$ steps. If $\overline{a}$ is the starting state, then either the agent state never changes which reduces to the previous case or $A(\overline{a}, w) = a$ for some $w \in \mathrm{Vs}$. Then, the adversary sets w as the initial state of the starting vertex and after the fist step the state of the agent changes to a and by the same argument as in the first case, the algorithm requires $\Omega(n^2)$ steps. ◀

▶ **Fact 8.** $(\forall\, a \in \mathrm{As})(\forall\, v \in \mathrm{Vs})\ R_2(a, v) \neq (a, v)$.

Proof. We will show that otherwise algorithm $\mathcal{A}$ falls into an infinite loop for some initial state of the path. Assume that $R_3(a, v) = (a, v, p)$ for some $a \in \mathrm{As}$, $v \in \mathrm{Vs}$, $p \in \mathrm{Ps}$. Then if a is the starting agent state, the agent falls into an infinite loop on the left gadget from Figure 2 (double circle denotes the starting position of the agent and a above the node indicates its initial state). If $\overline{a}$ is the starting state, then by Fact 7, $(\exists\, w \in \mathrm{Vs})\ A(\overline{a}, w) = a$. Let $q = P(\overline{a}, w)$ and note that the agent falls into an infinite loop in the right gadget at Figure 2. ◀

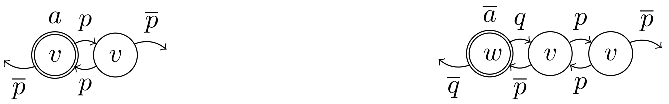


Figure 2 Two looping gadgets for an algorithm with $R_2(a, v) = (a, v)$.

Independently of the above facts, let us consider the foregoing **Algorithm Q** and more complicated gadgets, which force the agent to fall into an infinite loop (see Figure 3).

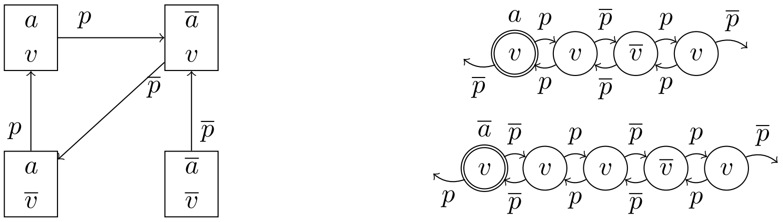


Figure 3 Definition of Algorithm Q (left) and gadgets on which it falls into an infinite loop.

6 Conclusions and open problems

One conclusion from our paper is that certain assumptions of the model of mobile agents can be exchanged. We showed, that in the context of linear time tree exploration, the assumption of clean memory at the nodes can be exchanged for a single token or the knowledge of the incoming port. The paper leaves a number of promising open directions. We showed that token and clean memory allow for linear time exploration of trees, however, we have not ruled out the possibility that linear time exploration of trees is feasible without both these assumptions. Our lower bound suggests that memory $\omega(1)$ at the agent is probably necessary in DirtyMem model. Another open direction would be to consider different graph classes, or perhaps directed graphs. Finally, a very interesting future direction is to study dual-memory exploration with team of multiple mobile agents. Such approach could lead to even smaller exploration time, however, dividing the work between the agents in such models is very challenging since the graph is initially unknown.

References

1 Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász, and Charles Rackoff. Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 218–223. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.34.

2 Evangelos Bampas, Leszek Gąsieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, Adrian Kosowski, and Tomasz Radzik. Robustness of the Rotor-Router Mechanism. *Algorithmica*, 78(3):869–895, 2017. doi:10.1007/s00453-016-0179-y.

3 Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil P. Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. *Inf. Comput.*, 176(1):1–21, 2002. doi:10.1006/inco.2001.3081.

4 Michael A. Bender and Donna K. Slonim. The Power of Team Exploration: Two Robots Can Learn Unlabeled Directed Graphs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 75–85, 1994. doi:10.1109/SFCS.1994.365703.

5 Petra Berenbrink, Colin Cooper, and Tom Friedetzky. Random Walks Which Prefer Unvisited Edges: Exploring High Girth Even Degree Expanders in Linear Time. *Random Struct. Algorithms*, 46(1):36–54, 2015. doi:10.1002/rsa.20504.

6 Dominik Bojko, Karol Gotfryd, Dariusz R. Kowalski, and Dominik Pajak. Tree exploration in dual-memory model, 2022. (Full version). arXiv:2112.13449.

7 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998. Proceedings of the Seventh International World Wide Web Conference. doi:10.1016/S0169-7552(98)00110-X.

8 Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. Label-Guided Graph Exploration by a Finite Automaton. *ACM Trans. Algorithms*, 4(4):42:1–42:18, August 2008. doi:10.1145/1383369.1383373.

9 H. K. Dai and Kevin E. Flannery. Improved Length Lower Bounds for Reflecting Sequences. In Jin-Yi Cai and Chak Kuen Wong, editors, *Computing and Combinatorics*, pages 56–67. Springer Berlin Heidelberg, 1996. doi:10.1007/3-540-61332-3_139.

10 Shantanu Das. Graph Explorations with Mobile Agents. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, Lecture Notes in Computer Science, pages 403–422. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-11072-7_16.

11 Shantanu Das and Nicola Santoro. Moving and Computing Models: Agents. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, Lecture Notes in Computer Science, pages 15–34. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-11072-7_2.

12 Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pająk, and Przemysław Uznański. Fast collaborative graph exploration. *Inf. Comput.*, 243:37–49, 2015. doi:10.1016/j.ic.2014.12.005.

13 Dariusz Dereniowski, Adrian Kosowski, Dominik Pająk, and Przemyslaw Uznanski. Bounds on the cover time of parallel rotor walks. *J. Comput. Syst. Sci.*, 82(5):802–816, 2016. doi:10.1016/j.jcss.2016.01.004.

14 Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *J. Algorithms*, 51(1):38–63, 2004. doi:10.1016/j.jalgor.2003.10.002.

15 Yann Disser, Jan Hackfeld, and Max Klimm. Tight Bounds for Undirected Graph Exploration with Pebbles and Multiple Agents. *J. ACM*, 66(6), October 2019. doi:10.1145/3356883.

16 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Škorić, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theoretical Computer Science*, 811:70–78, 2020. doi:10.1016/j.tcs.2018.03.006.

17 Mirosław Dynia, Jakub Łopuszański, and Christian Schindelhauer. Why Robots Need Maps. In G. Prencipe and S. Zaks, editors, *Structural Information and Communication Complexity*, volume 4474 of *Lecture Notes in Computer Science*, pages 41–50. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-72951-8_5.

18 Klim Efremenko and Omer Reingold. How Well Do Random Walks Parallelize? In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 5687 of *Lecture Notes in Computer Science*, pages 476–489. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-03685-9_36.

19 Robert Elsässer and Thomas Sauerwald. Tight bounds for the cover time of multiple random walks. *Theor. Comput. Sci.*, 412(24):2623–2641, 2011. doi:10.1016/j.tcs.2010.08.010.

20 Uriel Feige. A Tight Lower Bound on the Cover Time for Random Walks on Graphs. *Random Struct. Algorithms*, 6(4):433–438, 1995. doi:10.1002/rsa.3240060406.

21 Uriel Feige. A Tight Upper Bound on the Cover Time for Random Walks on Graphs. *Random Struct. Algorithms*, 6(1):51–54, 1995. doi:10.1002/rsa.3240060106.

22 Pierre Fraigniaud, Leszek Gąsieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective Tree Exploration. *Networks*, 48(3):166–177, 2006. doi:10.1002/net.20127.

23 Pierre Fraigniaud and David Ilcinkas. Digraphs Exploration with Little Memory. In V. Diekert and M. Habib, editors, *STACS 2004*, volume 2996 of *Lecture Notes in Computer Science*, pages 246–257. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24749-4_22.

24 Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, 2005. doi:10.1016/j.tcs.2005.07.014.

25 Pierre Fraigniaud, David Ilcinkas, Sergio Rajsbaum, and Sébastien Tixeuil. Space Lower Bounds for Graph Exploration via Reduced Automata. In A. Pelc and M. Raynal, editors, *Structural Information and Communication Complexity*, volume 3499 of *Lecture Notes in Computer Science*, pages 140–154. Springer Berlin Heidelberg, 2005. doi:10.1007/11429647_13.

26 Leszek Gąsieniec and Tomasz Radzik. Memory Efficient Anonymous Graph Exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *Lecture Notes in Computer Science*, pages 14–29. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-92248-3_2.

27 Ralf Klasing, Adrian Kosowski, Dominik Pająk, and Thomas Sauerwald. The multi-agent rotor-router on the ring: a deterministic alternative to parallel random walks. *Distributed Comput.*, 30(2):127–148, 2017. doi:10.1007/s00446-016-0282-y.

28 Adrian Kosowski. Faster Walks in Graphs: A $\widetilde{O}(n^2)$ Time-Space Trade-off for Undirected s-t Connectivity. In *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1873–1883. SIAM, 2013. doi:10.1137/1.9781611973105.133.

29 Adrian Kosowski and Dominik Pająk. Does adding more agents make a difference? A case study of cover time for the rotor-router. *J. Comput. Syst. Sci.*, 106:80–93, 2019. doi:10.1016/j.jcss.2019.07.001.

30 Michal Koucký. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002. doi:10.1016/S0022-0000(02)00023-5.

31 Michal Koucký. Log-space constructible universal traversal sequences for cycles of length $O(n^{4.03})$. *Theor. Comput. Sci.*, 296(1):117–144, 2003. doi:10.1016/S0304-3975(02)00436-X.

32 László Lovász. Random Walks on Graphs: A Survey. Combinatorics, Paul Erdos is Eighty. *Bolyai Soc. Math. Stud.*, 2:1–46, 1993.

33 Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theoretical Computer Science*, 463:62–72, 2012. Special Issue on Theory and Applications of Graph Searching Problems. doi:10.1016/j.tcs.2012.06.034.

34 Artur Menc, Dominik Pająk, and Przemysław Uznański. Time and space optimality of rotor-router graph exploration. *Inf. Process. Lett.*, 127:17–20, 2017. doi:10.1016/j.ipl.2017.06.010.

35 Nathan Michael et al. Collaborative Mapping of an Earthquake-Damaged Building via Ground and Aerial Robots. *Journal of Field Robotics*, 29(5):832–841, 2012. doi:10.1002/rob.21436.

36 Yoshiaki Nonaka, Hirotaka Ono, Kunihiko Sadakane, and Masafumi Yamashita. The hitting and cover times of Metropolis walks. *Theor. Comput. Sci.*, 411(16-18):1889–1894, 2010. doi:10.1016/j.tcs.2010.01.032.

37 Christian Ortolf and Christian Schindelhauer. A Recursive Approach to Multi-robot Exploration of Trees. In Magnús M. Halldórsson, editor, *Structural Information and Communication Complexity*, volume 8576 of *Lecture Notes in Computer Science*, pages 343–354. Springer International Publishing, 2014. doi:10.1007/978-3-319-09620-9_26.

38 Omer Reingold. Undirected Connectivity in Log-Space. *J. ACM*, 55(4), September 2008. doi:10.1145/1391289.1391291.

39 H. A. Rollik. Automaten in planaren Graphen. *Acta Inf.*, 13(3):287–298, March 1980. doi:10.1007/BF00288647.

40 Yuichi Sudo, Fukuhito Ooshita, and Sayaka Kamei. Self-stabilizing Graph Exploration by a Single Agent. *CoRR*, abs/2010.08929, 2020. arXiv:2010.08929.

41 Vladimir Yanovski, Israel A. Wagner, and Alfred M. Bruckstein. A Distributed Ant Algorithm for Efficiently Patrolling a Network. *Algorithmica*, 37(3):165–186, 2003. doi:10.1007/s00453-003-1030-9.

On Vanishing Sums of Roots of Unity in Polynomial Calculus and Sum-Of-Squares

Ilario Bonacina ✉
Universitat Politècnica de Catalunya, Barcelona, Spain

Nicola Galesi ✉
Sapienza Università di Roma, Italy

Massimo Lauria ✉ 🏠
Sapienza Università di Roma, Italy

Abstract

Vanishing sums of roots of unity can be seen as a natural generalization of knapsack from Boolean variables to variables taking values over the roots of unity. We show that these sums are hard to prove for polynomial calculus and for sum-of-squares, both in terms of degree and size.

2012 ACM Subject Classification Theory of computation → Proof complexity; Computing methodologies → Representation of polynomials

Keywords and phrases polynomial calculus, sum-of-squares, roots of unity, knapsack

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.23

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2022/105/`

Funding The first author was supported by the MICIN grants PID2019-109137GB-C22 and IJC2018-035334-I, and partially by the grant PID2019-109137GB-C21.

Acknowledgements The authors would like to thank Albert Atserias for fruitful discussions.

1 Introduction

Statements from combinatorics, constraint satisfaction problems (CSP), arithmetic circuit design, and algebra itself can be formalized either as statements about polynomial equalities (and inequalities), or via propositional logic. The approach based on propositional logic is amenable to *state-of-the-art* algorithms for satisfiability (SAT), usually variations of *Conflict-Driven-Clause-Learning* SAT solvers (CDCL), see for instance [28, 29, 3]. These solvers are surprisingly efficient, but their reasoning is ultimately based on the *resolution* proof system. On problems coming from algebra, CDCL solvers do not exploit the algebraic aspects of the problem, and therefore are typically unable to solve them. Switching to algebra allows to leverage on tools as Hilbert's Nullstellensatz and Gröbner basis computation in order to solve systems of polynomial equations [10], or semidefinite programming to solve systems of polynomial inequalities [30, 25]. These algebraic tools have been successful in practice for instance to solve κ-COLORING [11, 12, 13] and the verification of arithmetic multiplier circuits [22, 21, 23]. κ-COLORING, and in general CSP problems over finite domains of size κ, are naturally encoded using κ-valued variables. In particular, the algebraic tools for κ-COLORING use the *Fourier encoding*, which represents values via complex variables z subjected to the constraint $z^\kappa = 1$ and hence such that

$$z \in \{1, \zeta, \zeta^2, \ldots, \zeta^{\kappa-1}\} ,$$

where ζ is a primitive κth root of unity. A κ-valued variable z can be alternatively represented as a collection of indicator Boolean variables $x_1, \ldots, x_\kappa$ equipped with the additional constraint $x_1 + \cdots + x_\kappa = 1$.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Picking the right encoding is essential to leverage the algebraic structure of the problem. Even simple changes, for instance adding new variables to represent Boolean negations may already give significant speedups both in theory and in practice [14, 20].

In this paper, following a general approach from proof complexity, we show that algorithms leveraging Hilbert's Nullstellensatz or Gröbner basis computations cannot prove efficiently the unsatisfiability of some natural sets of polynomials equations over the Fourier variables.

The proof systems we consider are *polynomial calculus* and *sum-of-squares*. Polynomial calculus is a well studied proof system that captures Hilbert's Nullstellensatz and Gröbner basis computations. It is a system that certifies the unsatisfiability of sets of polynomial equations. It has been studied for polynomials over different fields or rings and, in particular, also for polynomials over the complex numbers $\mathbb{C}$, see for instance [7]. Given polynomials $p_1, \ldots, p_m$ with coefficients in a field $\mathbb{F}$, a refutation of $\{p_1 = 0, \ldots, p_m = 0\}$ in *polynomial calculus over* $\mathbb{F}$, denoted as $\mathsf{PC}_{\mathbb{F}}$, is a sequence of polynomials $p_1, \ldots, p_s$ over $\mathbb{F}$ such that $p_s = 1$ and each $p_{m+1}, \ldots, p_s$ is either (1) $r \cdot p_k$ for some polynomial r with coeffcients in $\mathbb{F}$ and some $k < i$; or (2) a linear combination $\alpha p_j + \beta p_k$ for $j, k < i$ and $\alpha, \beta \in \mathbb{F}$.

Regarding sum-of-squares, it is a systems to certify the unsatisfiability of sets of polynomial equations *and inequalities* over $\mathbb{R}$. A sum-of-squares $\mathsf{SoS}_{\mathbb{R}}$ refutation of the set of contraints $\{p = 0 \ : \ p \in P\} \cup \{h \geq 0 \ : \ h \in H\}$ is an identity of the form

$$-1 = \sum_{p \in P} q_p \cdot p + \sum_{h \in H} q_h \cdot h + \sum_{s \in S} s^2 \ ,$$

where the s, q_p, q_h are polynomials over $\mathbb{R}$ and moreover the q_hs are sums of squared polynomials. In presence of Boolean or $\{\pm 1\}$-valued variables, $\mathsf{SoS}_{\mathbb{R}}$ p-simulates $\mathsf{PC}_{\mathbb{R}}$ [4, 34].

In this paper, we introduce a generalization of sum-of-squares with polynomials over $\mathbb{C}$, $\mathsf{SoS}_{\mathbb{C}}$ (see Section 2 for the formal definition). Since $\mathbb{C}$ is not an ordered field, this generalization of sum-of-squares to $\mathbb{C}$ can only be used to certify the unsatisfiability of sets of polynomial *equations*. For sets of polynomial equations over $\mathbb{R}$ and in the presence of Boolean variables, $\mathsf{SoS}_{\mathbb{C}}$ coincides with the usual notion of sum-of-squares over $\mathbb{R}$, but the generalization is necessary to deal with Fourier variables or to reason about polynomials over $\mathbb{C}$. In presence of Fourier variables, $\mathsf{SoS}_{\mathbb{C}}$ p-simulates $\mathsf{PC}_{\mathbb{C}}$, see Section 2 for more details.

PC and SoS can be used to solve computational problems once they are encoded as sets of polynomials equations. It is customary to discuss sets of polynomial equations simply as sets of polynomials. We adopt this custom and we say that a set of polynomials over $\mathbb{C}$ is *satisfiable* when it has a common zero $\boldsymbol{\alpha} \in \mathbb{C}^n$. The most naïve algebraic encoding is to use variables ranging over $\{0, 1\}$ to represent the truth values of variables. This Boolean nature of a variable x is enforced via the polynomial $x^2 - x$. With this encoding then, for example, the satisfiability of a propositional clause $x \vee \neg y \vee z$ can be encoded as the satisfiability of the set of polynomials $\{(1 - x)y(1 - z), \ x^2 - x, \ y^2 - y, \ z^2 - z\}$. Truth values of variables are sometimes also encoded in the Fourier basis $\{\pm 1\}$ and, as we already mentioned, for some CSPs it is convenient to use κ-valued variables using the κth roots of unity.

Finding deductions in PC/SoS may be hard, and in general there are important proxy measures to estimate such hardness: the maximum *degree* of the polynomials involved in the deductions, and the number of monomials involved in the whole proof when polynomials are written explicitly as sums of monomials (*size*). The degree is a very rough measure of the proof search space, the size is a lower bound on the time required to produce the proof.

Studying size and degree complexity in algebraic systems over Fourier encodings is particularly relevant to understand how to leverage to proof complexity techniques like the *Smolensky's method* in circuit complexity [33]. He proved exponential lower bounds to compute the MOD_p function by bounded-depth circuits using the unbounded gates in $\{\wedge, \vee, \mathrm{MOD}_q\}$, for p and q relatively prime, employing a reduction to low-degree polynomials over $\mathrm{GF}(q)$ approximating such circuits. In proof complexity, it is a long-standing problem to obtain lower bounds for proof systems over bounded-depth formulas with modular gates.

Non-trivial degree lower bounds for Fourier encodings were first obtained for the Nullstellensatz proof system and PC by Grigoriev in [18] and Buss *et al.* in [7] for the Tseitin principle over p-valued variables (instead of the usual $\{0,1\}$) and the so-called MOD_p principles [7].

For $\mathsf{PC}/\mathsf{SoS}_{\mathbb{R}}$ over Boolean variables we know degree and size lower bounds for the encodings of several computational problems, see for instance [2, 17, 31, 32, 35]. For the size lower bounds in PC and $\mathsf{SoS}_{\mathbb{R}}$ this is essentially due to degree-size tradeoffs: if a set of polynomials *over Boolean variables* has no refutation in $\mathsf{PC}/\mathsf{SoS}_{\mathbb{R}}$ of degree at most D, then it has no refutation containing less than $2^{\Omega\left(\frac{(D-d)^2}{n}\right)}$ monomials, see [1, 19].

No such degree-size relation holds for polynomials over the Fourier variables. For instance, it is well-known that Tseitin contradictions over the Boolean variables $\{0,1\}$ require an exponential number of monomials to be refuted in PC, while PC can refute them with a linear number of monomials if the encoding uses the variables $\{\pm 1\}$, see [7].

To the best of our knowledge, the first size lower bounds in $\mathsf{PC}/\mathsf{SoS}_{\mathbb{R}}$ for polynomials with $\{\pm 1\}$ variables are proved by [34] for the pigeonhole principle and random 11-CNFs. Moreover that work provides a technique to turn strong degree lower bounds in that framework into strong size lower bounds for the same polynomials composed with some carefully constructed gadgets. We extend this latter approach to get size lower bound under the Fourier encoding of κ-valued variables, and we apply it to a generalization of KNAPSACK for these variables.

The classical KNAPSACK problem corresponds to the set of polynomials

$$\left\{ \sum_{i=1}^{n} c_i x_i - r \ , \quad x_1^2 - x_1, \ldots, x_n^2 - x_n \right\} , \tag{1}$$

where $r, c_1, \ldots, c_n \in \mathbb{C}$. For KNAPSACK are known linear degree lower bounds in PC, see [19, Theorem 5.1], and, when all the c_is are 1 and $r \in \mathbb{R}$, degree lower bounds in $\mathsf{SoS}_{\mathbb{R}}$ of the form $\min\{2\lfloor \min\{r, n-r\}\rfloor + 3, n\}$, see [17]. Size lower bounds are also implied by the respective size-degree tradeoffs [19, 1].

Sums of roots of unity. We consider the problem of when a sum of n variables with values in the κth roots of unity can be equal to some value $r \in \mathbb{C}$, that is the satisfiability of

$$\mathsf{SRU}_n^{\kappa,r} := \left\{ \sum_{i\in[n]} z_i - r, \ z_1^{\kappa} - 1, \ldots, z_n^{\kappa} - 1 \right\} . \tag{2}$$

Linear relations of the form $\sum_{i=1}^{n} c_i \zeta_i = 0$, where c_i are complex numbers and ζ_i are roots of unity, arise naturally in several contexts [9], and have been extensively studied in the literature, see for instance [16, 15]. When κ divides n, $\kappa \mid n$, it is easy to see that $\mathsf{SRU}_n^{\kappa,0}$ is satisfiable, because the κth roots of unity sum to zero.

When κ is a power of a prime number p, this is indeed the only possibility, that is $\mathsf{SRU}_n^{\kappa,0}$ is satisfiable over $\mathbb{C}$ if and only if $p \mid n$. (For the simple proof of this fact see the full version.) For the general case of $\kappa \in \mathbb{N}$, Lam and Leung [24] characterize exactly the set of natural numbers n such that $\mathsf{SRU}_n^{\kappa,0}$ is satisfiable. As a corollary of their results, if κ is not a power of a prime then, there exists a $n_0(\kappa)$ s.t. for every $n \geq n_0(\kappa)$ the set of polynomials $\mathsf{SRU}_n^{\kappa,0}$ is satisfiable.

Our results. In this paper we show the hardness to certify in PC and $\mathsf{SoS}_{\mathbb{C}}$ the unsatisfiability of $\mathsf{SRU}_n^{\kappa,0}$ when κ is a prime and does not divide n. For simplicity, we leave the discussion for the case when κ is a power of a prime for the journal version. Our main results regarding $\mathsf{PC}/\mathsf{SoS}_{\mathbb{C}}$ informally say that $\mathsf{SoS}_{\mathbb{C}}$ and $\mathsf{PC}_{\mathbb{C}}$ cannot capture divisibility arguments.

A linear degree lower bound for $\mathsf{SRU}_n^{2,0}$ follows immediately, via a linear transformation, from the known degree lower bound for KNAPSACK in SoS, since the Grigoriev's lower bound in [17] can easily extended to $\mathsf{SoS}_{\mathbb{C}}$. In this paper we generalize this result proving degree and size lower bounds in $\mathsf{SoS}_{\mathbb{C}}$ for $\mathsf{SRU}_n^{\kappa,r}$ for κ an odd prime.

▶ **Theorem 1** (Degree lower bound for $\mathsf{SRU}_n^{\kappa,r}$). *Let $n, d \in \mathbb{N}$, κ be a prime, $r \in \mathbb{C}$. Let r be written as $r_1 + \zeta r_2$, where $r_1, r_2 \in \mathbb{R}$ and ζ is some κth primitive root of unity. If*

$$\kappa d \leq \min\{r_1 + r_2 + (\kappa - 1)n + \kappa,\ n - r_1 - r_2 + \kappa\}\ ,$$

then there are no $\mathsf{SoS}_{\mathbb{C}}$-refutations of $\mathsf{SRU}_n^{\kappa,r}$ of degree at most d. In particular, $\mathsf{SRU}_n^{\kappa,0}$ requires refutations of degree $\Omega\left(\frac{n}{\kappa}\right)$ in $\mathsf{SoS}_{\mathbb{C}}$.

From the set of polynomials in $\mathsf{SRU}_n^{2,r}$ we can easily infer the polynomials in $\mathsf{SRU}_n^{\kappa,0}$, via a linear transformation and a weakening. This is enough to prove degree lower bounds for $\mathsf{SRU}_n^{\kappa,0}$ in $\mathsf{PC}_{\mathbb{C}}$ since, Impagliazzo, Pudlák, and Sgall [19, Theorem 5.1] proved a linear degree lower bound for KNAPSACK and therefore $\mathsf{SRU}_n^{2,r}$ for any r. This is not the case for $\mathsf{SoS}_{\mathbb{C}}$: $\mathsf{SRU}_n^{2,r}$ is refutable in small degree and size in $\mathsf{SoS}_{\mathbb{C}}$ if $r \in \mathbb{C} \setminus \mathbb{R}$, see Example 4. In other words, in $\mathsf{SoS}_{\mathbb{C}}$, unlike the case of PC, it is not possible to reduce the hardness of $\mathsf{SRU}_n^{\kappa,0}$, for $\kappa > 2$ to KNAPSACK.

To prove the degree lower bound in $\mathsf{SoS}_{\mathbb{C}}$ for $\mathsf{SRU}_n^{\kappa,r}$ (Theorem 1) first we construct a candidate pseudo-expectation for $\mathsf{SRU}_n^{\kappa,r}$ based on the symmetries of the set of polynomials. Then we prove its correctness, following the approach by Blekherman [5, 6] as presented in [27, Theorem B.11] but generalized to $\mathsf{SoS}_{\mathbb{C}}$. Due to page limitations we only show in Section 4 how to use the generalization of Blekherman's theorem (Theorem 13) to prove Theorem 1.

We also prove a size lower bound for $\mathsf{SRU}_n^{\kappa,0}$ in $\mathsf{SoS}_{\mathbb{C}}$. We lift degree lower bounds to size lower bounds generalizing to κ-valued Fourier variables the lifting approach due to Sokolov [34], originally designed for real valued polynomials and $\{\pm 1\}$-variables.

▶ **Theorem 2** (Size lower bound for $\mathsf{SRU}_n^{\kappa,0}$). *Let κ be a prime and $n \in \mathbb{N}$, if $n \gg \kappa$ then the set of polynomials $\mathsf{SRU}_n^{\kappa,0}$ has no refutation in $\mathsf{SoS}_{\mathbb{C}}$ within monomial size $2^{o(n)}$.*

Theorem 2, for $\kappa = 2$, follows easily from the techniques of Sokolov [34] and Grigoriev's degree lower bound for KNAPSACK [17]. For $\kappa > 2$ it requires some non-trivial extension of the lifting technique from [34]. That is, the composition of polynomials with appropriate gadgets (see Definition 6). Our generalization of the lifting from [34] is Theorem 7 in Section 3.

Theorem 1 and Theorem 2 also hold for $\mathsf{PC}_{\mathbb{C}}$, since $\mathsf{SoS}_{\mathbb{C}}$ simulates $\mathsf{PC}_{\mathbb{C}}$.

Structure of the paper. In the next section, we give the necessary preliminaries on roots of unity and the formal definition of $\mathsf{SoS}_{\mathbb{C}}$. In Section 3 we layout the proof of a way to lift degree lower bounds to size lower bounds in $\mathsf{SoS}_{\mathbb{C}}$ for sets of polynomials over the Fourier variables (Theorem 7) and we show how to prove Theorem 2 from Theorem 1 and Theorem 7. The proof of Theorem 1 is in Section 4.

2 Preliminaries

Given $n, k \in \mathbb{N}$, let $[n] := \{1, \dots, n\}$, and if k divides n we write $k \mid n$. For $a \in \mathbb{R}$ and $b \in \mathbb{N}$, let $\binom{a}{0} := 1$ and $\binom{a}{b} := \frac{a(a-1)\dots(a-b+1)}{b!}$ for $b \geq 1$. **Boldface** symbols indicate vectors, and $\boldsymbol{x}$ denotes a vector with n elements $(x_1, \dots, x_n)$. We denote with $\boldsymbol{x}$ Boolean variables, with $\boldsymbol{z}$ κ-valued variables and with $\boldsymbol{y}$ generic variables or auxiliary variables. Given a set of polynomials $P \subseteq \mathbb{C}[\boldsymbol{y}]$, $\langle P \rangle$ denotes the ideal generated by P in $\mathbb{C}[\boldsymbol{y}]$. Let $\underline{i}$ be the imaginary unit in $\mathbb{C}$, i.e. $\underline{i}^2 = -1$.

Roots of unity. For a positive integer κ, a *κth root of unity* is a root of the polynomial $z^\kappa - 1$. All the roots of unity except 1 are also roots of the polynomial $1 + z + \cdots + z^{\kappa-1}$, indeed $z^\kappa - 1 = (z-1) \cdot (1 + z + \cdots + z^{\kappa-1})$. A κth root of unity ζ is called *primitive* if $\zeta^t \neq 1$ for all $1 \leq t < \kappa$. If this is the case the κth roots of unity are indeed $1, \zeta, \zeta^2, \dots, \zeta^{\kappa-1}$. Some of the results of this paper hold for roots of unity in generic fields but, for sake of clarity, we only consider roots of unity in $\mathbb{C}$. Notice that the complex conjugate of ζ^t is $\zeta^{\kappa-t}$. For concreteness, we denote as ζ a specific primitive κth root of unity, for instance $\mathrm{e}^{2\pi\underline{i}/\kappa}$, and as Ω_κ the set $\{1, \zeta, \zeta^2, \dots, \zeta^{\kappa-1}\}$. We often denote as ω a generic element in Ω_κ.

SoS over the complex numbers. The key concept at the core of the sum-of-squares proof system is that squares of real valued polynomials are always positive. For a complex valued polynomial $p \in \mathbb{C}[\boldsymbol{y}]$ we use that $p \cdot p^* \geq 0$, where p^* is the function that maps the assignment $\boldsymbol{\alpha}$ to the complex conjugate of the value $p(\boldsymbol{\alpha})$. We need a polynomial representation of function p^* that we call *formal conjugate* of p. To have such polynomial, in general, we would need to use a twin formal variable to represent x^* for any original variable x. Furthermore we would need to add to the proof system various axioms to relate x and x^*. In this work we focus on $\mathsf{SoS}_\mathbb{C}$ under the Boolean and Fourier encodings, hence we can represent formal conjugates as polynomials without any additional axiom or variable. For a Boolean variable $x \in \{0, 1\}$ we have that x^* is x itself. For a Fourier variable z raised to an integer power $0 \leq t < \kappa$, the function $(z^t)^*$ is $z^{\kappa-t}$. Then the operator * extends homomorphically on sums and products, and it is equal to the usual complex conjugate on complex number. We are now ready to define the sum-of-squares proof system over complex number.

▶ **Definition 3** (Sum-of-Squares over $\mathbb{C}$, $\mathsf{SoS}_\mathbb{C}$)**.** *Fix an integer $\kappa \geq 2$. Consider a set of polynomials $P \subseteq \mathbb{C}[\boldsymbol{x}, \boldsymbol{z}]$ where P contains $z^\kappa - 1$ and for each variable z, and contains $x^2 - x$ for each variable x. A* refutation of P in $\mathsf{SoS}_\mathbb{C}$ *is an equality of the form*

$$-1 = \sum_{p \in P} q_p \cdot p + \sum_{s \in S} s \cdot s^* \ ,$$

where the $s \in S$ and q_p for $p \in P$ are in $\mathbb{C}[\boldsymbol{x}, \boldsymbol{z}]$ and each s^* is the formal conjugate of s.

The degree *of the refutation is* $\max\{\deg(q_p) + \deg(p), \deg(s \cdot s^*) \ : \ p \in P, \ s \in S\}$*. The* size *of the refutation is the total number of monomials occurring with non-zero coefficients among polynomials $\{q_p, p \ : \ p \in P\} \cup \{s, s^* \ : \ s \in S\}$.*

Notice that, for polynomials $p, q \in \mathbb{R}[\boldsymbol{x}, \boldsymbol{z}]$, $(p + \underline{i}q)(p - \underline{i}q) = p^2 + q^2$. Therefore for $P \subseteq \mathbb{R}[\boldsymbol{x}]$ and containing $x_i^2 - x_i$ for every $i \in [n]$, the notion of $\mathsf{SoS}_\mathbb{C}$ and $\mathsf{SoS}_\mathbb{R}$ coincide.

By Hilbert's Nullstellensatz, $\mathsf{SoS}_\mathbb{C}$ is complete: for every unsatisfiable set of polynomials P there is a $\mathsf{SoS}_\mathbb{C}$-refutation. Conversely, only unsatisfiable sets of polynomials have $\mathsf{SoS}_\mathbb{C}$ refutations: for any assignment $\boldsymbol{\alpha}$ of a polynomial s, polynomial $s \cdot s^*$ evaluates to $|s(\boldsymbol{\alpha})|^2$ which is a non-negative real number.

▶ **Example 4.** The set of polynomials $\{\sum_{j\in[n]} x_j - \underline{i},\ x_1^2 - x_1, \dots,\ x_n^2 - x_n\}$ has a simple $\mathsf{SoS}_{\mathbb{C}}$ refutation:

$$-1 = -(\sum_{j\in[n]} x_j - \underline{i})(\sum_{j\in[n]} x_j + \underline{i}) + (\sum_{j\in[n]} x_j)^2 \ .$$

Via similar algebraic equalities it is not hard to see that $\mathsf{SoS}_{\mathbb{C}}$ can refute easily the set of polynomials corresponding to KNAPSACK in eq. (1) when $r \in \mathbb{C} \setminus \mathbb{R}$ and all c_is are real. By a simple modification of [4, Lemma 3.1] and [34], we also have that, in presence of the axioms $y_i^\kappa - 1$, $\mathsf{SoS}_{\mathbb{C}}$ simulates $\mathsf{PC}_{\mathbb{C}}$, that is $\mathsf{PC}_{\mathbb{C}}$ refutations can be converted to $\mathsf{SoS}_{\mathbb{C}}$ refutations with just a polynomial increase in size.[1] Impagliazzo, Pudlák, and Sgall in [19, Theorem 5.1] prove that the set of polynomials in eq. (1) is hard for $\mathsf{PC}_{\mathbb{C}}$, hence $\mathsf{SoS}_{\mathbb{C}}$ is strictly stronger than $\mathsf{PC}_{\mathbb{C}}$.

3 Size lower bounds in Sum-of-Squares

In this section we prove the size lower bound for $\mathsf{SRU}_n^{\kappa,0}$ in $\mathsf{SoS}_{\mathbb{C}}$ from the the corresponding degree lower bound. That is we show how to prove Theorem 2 from Theorem 1. On a very high level, this is done *composing* the polynomials in $\mathsf{SRU}_n^{\kappa,r}$ with some polynomials $\boldsymbol{g}$, obtaining then some new set of polynomials $\mathsf{SRU}_n^{\kappa,r} \circ \boldsymbol{g}$, and then via a lifting theorem showing how degree lower bounds on $\mathsf{SRU}_n^{\kappa,r}$ imply size lower bounds on $\mathsf{SRU}_n^{\kappa,r} \circ \boldsymbol{g}$.

▶ **Definition 5** (composition of polynomials). *Let $\boldsymbol{x}, \boldsymbol{y}_1, \dots \boldsymbol{y}_n$ be tuples of distinct variables where $\boldsymbol{y}_j = (y_{j1}, \dots, y_{j\ell_j})$. Given a polynomial $p \in \mathbb{C}[\boldsymbol{x}]$ and $\boldsymbol{g} = (g_1 \dots, g_n)$ with $g_j \in \mathbb{C}[\boldsymbol{y}_j]$ we denote by $p \circ \boldsymbol{g}$ the polynomial obtained substituting each instance of the variable x_j in p with the polynomial $g_j(\boldsymbol{y}_j)$ and then expanding the obtained algebraic expression as a sum of monomials in the new variables. The polynomial $p \circ \boldsymbol{g}$ then belongs to the ring $\mathbb{C}[\boldsymbol{y}_1, \dots, \boldsymbol{y}_n]$.*

Similarly, for a set of polynomials $P \subset \mathbb{C}[\boldsymbol{x}]$ we denote as $P \circ \boldsymbol{g}$ the set of polynomials $\{p \circ \boldsymbol{g} \ : \ p \in P\}$.

We are only interested in composing polynomials with $\boldsymbol{g}$ when $\boldsymbol{g}$ has some good properties. Those are a generalization of the notion of *compliant gadgets* from [34, Definition 2.1].

▶ **Definition 6** (compliant polynomial). *A polynomial $g \in \mathbb{C}[y_1, \dots, y_\ell]$ is* compliant *if it is symmetric and there exists a function $h : \Omega_\kappa \to \Omega_\kappa^\ell$ such that*

1. *$g \circ h = \mathbf{id}$, i.e. for all $b \in \Omega_\kappa$, $g(h(b)) = b$;*
2. *for each $b \in \Omega_\kappa$, the first κ coordinates of $h(b)$ list all the elements of Ω_κ; and*
3. *$\prod_{\omega\in\Omega_\kappa} h(\omega)$ is a constant function.*

We say that $\boldsymbol{g} = (g_1 \dots, g_n)$ with $g_j \in \mathbb{C}[\boldsymbol{y}_j]$ is compliant *when each g_j is compliant.*

The original definition of [34, Definition 2.1] focuses on real polynomials and sets of values $\{0, 1\}$ and $\{\pm 1\}$, while ours focuses on complex polynomials and the set of κth roots of unity.

The size lower bound on $\mathsf{SRU}_n^{\kappa,0}$ in $\mathsf{SoS}_{\mathbb{C}}$ follows from the following general result.

▶ **Theorem 7.** *Let P a finite set of polynomials of degree at most d_0 in $\mathbb{C}[\boldsymbol{x}]$ containing the polynomials $x_i^\kappa - 1$ for each $i \in [n]$. Let $\boldsymbol{g}$ be a tuple of compliant polynomials with $g_i \in \mathbb{C}[y_{i1}, \dots, y_{i\ell_i}]$. If P requires degree D to be refuted in $\mathsf{SoS}_{\mathbb{C}}$, then*

$$P \circ \boldsymbol{g} \cup \{y_{ij}^\kappa - 1 \ : \ i \in [n],\ j \in [\ell_i]\}$$

requires monomial size at least $\exp(\frac{(D-d_0)^2}{8\ell^\kappa(\kappa-1)n})$ to be refuted in $\mathsf{SoS}_{\mathbb{C}}$, where $\ell = \max_{i\in[n]} \ell_i$.

[1] The main difference with [4, Lemma 3.1] and [34] is to consider polynomials $s \cdot s^*$ instead of squares s^2 and then to use the algebraic equality $(p+q)(p+q)^* + (p-q)(p-q)^* = 2pp^* + 2qq^*$ instead of the one for the reals $(p+q)^2 + (p-q)^2 = 2p^2 + 2q^2$.

This result is a generalization of [34, Theorem 4.2]. Before seeing how to prove this result let us see how to apply it to prove a size lower bound for $\mathsf{SRU}_n^{\kappa,0}$, that is Theorem 2, restated below for convenience of the reader.

▶ **Theorem 2** (Size lower bound for $\mathsf{SRU}_n^{\kappa,0}$). *Let κ be a prime and $n \in \mathbb{N}$, if $n \gg \kappa$ then the set of polynomials $\mathsf{SRU}_n^{\kappa,0}$ has no refutation in $\mathsf{SoS}_{\mathbb{C}}$ within monomial size $2^{o(n)}$.*

Proof. Let $n = (2\kappa + 1)n' + b$ with $b \in \{0, \dots, 2\kappa\}$. Let $\ell_1 = \cdots = \ell_b = 2\kappa + 2$ and $\ell_{b+1} = \cdots = \ell_{n'} = 2\kappa + 1$. Consider the tuple $\boldsymbol{g} = (g_1, \dots, g_{n'})$ where $g_i \in \mathbb{C}[y_{i1}, \dots, y_{i\ell_i}]$ is the polynomial

$$g_i(y_{i1}, \dots, y_{i\ell_i}) := \frac{1}{\kappa}\Big(\sum_{j \in [\ell_i]} y_{ij} - (\ell_i - 2\kappa)\Big) .$$

We have that $\mathsf{SRU}_n^{\kappa,0}$ after renaming of variables is a subset of

$$\mathsf{SRU}_{n'}^{\kappa,r} \circ \boldsymbol{g} \cup \{y_{ij}^{\kappa} - 1 \ :\ i \in [n'],\ j \in [\ell_i]\} \tag{3}$$

with $r = -\frac{n'+b}{\kappa}$. By Theorem 1, there are no $\mathsf{SoS}_{\mathbb{C}}$ refutations of $\mathsf{SRU}_{n'}^{\kappa,r}$ in degree $\frac{n'}{\kappa}$. Each g_i is compliant. Indeed, the polynomial g_i is symmetric and we can take as $h_i : \Omega_\kappa \to \Omega_\kappa^{\ell_i}$ the function mapping

$$h_i : \omega \mapsto (1, \zeta, \zeta^2, \dots, \zeta^{\kappa-1}, \underbrace{1, 1, \dots, 1}_{\ell_i - 2\kappa}, \underbrace{\omega, \omega, \dots, \omega}_{\kappa}) ,$$

where ζ is a primitive κth root of unity in $\mathbb{C}$. Clearly, $g \circ h$ is the identity and

$$\prod_{\omega \in \Omega_\kappa} h_i(\omega) = \zeta^{\kappa(\kappa-1)/2}\omega^{\kappa} = \zeta^{\kappa(\kappa-1)/2}$$

since ω is a κth root of unity. By Theorem 7, the set of polynomials (3) requires $\mathsf{SoS}_{\mathbb{C}}$ refutations of monomial size at least $\exp\big(\frac{(\frac{n'}{\kappa} - \kappa)^2}{8\ell^{\kappa}(\kappa-1)n'}\big) = 2^{\Omega(n)}$ if $n \gg \kappa$. Therefore $\mathsf{SRU}_n^{\kappa,0}$ requires refutations size $2^{\Omega(n)}$, too. ◀

We conclude this section with a proof sketch of Theorem 7. The overall structure of the argument is that typical for size-degree trade-offs and can be found for instance in [8, 34, 1]. The idea is to show, on one side, that there exists a relatively long sequence of restrictions such that the restricted polynomials have small degree refutations (Theorem 8) and that each individual restriction can only make the degree decrease a little (Lemma 9). Those two facts will imply that the sequence of restrictions must be very long and this will imply the size-degree trade-off.

The *reduced degree* of a refutation in $\mathsf{SoS}_{\mathbb{C}}$ of a set of polynomials P containing the polynomials $x_j^{\kappa} - 1$ is the degree of the refutation where we do not take in account the degrees of the polynomials q_p where p is $x_j^{\kappa} - 1$ (see Definition 3).

Next theorem is the first ingredient for the proof of Theorem 7. It is a generalization of [34, Theorem 4.1] and its proof, an adaptation of the argument given in [34], is in the full version.

▶ **Theorem 8.** *Let P be finite a set of polynomials of degree d_0 in $\mathbb{C}[\boldsymbol{x}]$ containing the polynomials $x_j^{\kappa} - 1$ for each $j \in [n]$. Let $\boldsymbol{g}$ be a tuple of compliant polynomials with $g_i \in \mathbb{C}[y_{i1}, \dots, y_{i\ell_i}]$ and $\omega_1, \omega_2, \dots, \omega_m \in \Omega_\kappa$. If there is a $\mathsf{SoS}_{\mathbb{C}}$ refutation of $P \circ \boldsymbol{g} \cup \{y_{ij}^{\kappa} - 1 \ :\ i \in [n],\ j \in [\ell_i]\}$ of size s then there exists a sequence of variables $x_{i_1}, \dots, x_{i_m}$ with $m \geq \ell^{\kappa} n \ln(s)/D$ such that*

1. *$\ell = \max_i \ell_i$;*
2. *the choice of x_{i_t} only depends on $\omega_1, \dots, \omega_{t-1}$;*
3. *there is a $\mathsf{SoS}_{\mathbb{C}}$ refutation of $P\restriction_{x_{i_1}=\omega_1,\dots,x_{i_m}=\omega_m}$ of reduced degree at most $D + d_0$.*

The second ingredient for the proof of Theorem 7 is the following lemma.

▶ **Lemma 9.** *Let P be a finite set of polynomials in $\mathbb{C}[\boldsymbol{x}]$ containing the polynomials $x_j^\kappa - 1$ for each $j \in [n]$. Suppose any $\mathsf{SoS}_{\mathbb{C}}$ refutation of P has reduced degree at least D. Then, for any variable x_j there is $\omega \in \Omega_\kappa$ such that $\mathsf{SoS}_{\mathbb{C}}$ refutations of $P\restriction_{x_j=\omega}$ must have reduced degree at least $D - 2\kappa + 2$.*

Proof. (sketch) For sake of contradiction, suppose there exists some variable x such that for every $\omega \in \Omega_\kappa$, $P\restriction_{x=\omega}$ has a refutation of reduced degree $D - 2\kappa + 1$. For every $\ell \in \mathbb{N}$, $x^\ell - \omega^\ell$ is a multiple of $x - \omega$. Therefore, for every $p \in P$, the polynomial $p - p\restriction_{x=\omega}$ belongs to the ideal generated by $x - \omega$. This means that we can transform refutations of $P\restriction_{x=\omega}$ into refutations of $P \cup \{x - \omega\}$ without increasing the degree. Hence, there are refutations of $P \cup \{x - \omega\}$ of reduced degree $D - 2\kappa + 1$ for every $\omega \in \Omega_\kappa$.

Let π_ω be a refutation of $P \cup \{x - \omega\}$ of reduced degree $D - 2\kappa + 1$. Let $q_\omega(x) = \prod_{\omega' \neq \omega}(x - \omega')$.

It is easy to see that multiplying π_ω by the polynomial $q_\omega q_\omega^*$ we get a derivation of $-q_\omega q_\omega^*$ from P. This new derivation has reduced degree $D-2\kappa+1+2(\kappa-1) = D-1$. Now we can take a linear combination (with *non-negative real* coefficients) of the previous derivations to get the derivation of -1. More precisely we need numbers $\alpha_\omega \geq 0$ such that $\sum_{\omega\in\Omega_\kappa} \alpha_\omega q_\omega q_\omega^* - 1 \in \langle x^\kappa - 1\rangle$. Setting $\alpha_\omega = 1/q_\omega(\omega)q_\omega(\omega)^*$ we get that that $\sum_{\omega\in\Omega_\kappa} \alpha_\omega q_\omega q_\omega^* - 1$ is zero for all $\omega \in \Omega_\kappa$ and therefore in the ideal $\langle x^\kappa - 1\rangle$. This finally gives a $\mathsf{SoS}_{\mathbb{C}}$ refutation of P in degree $D - 1$, contradicting the assumption on P. ◀

Proof of Theorem 7. Let s be the smallest size of a $\mathsf{SoS}_{\mathbb{C}}$ refutation of the set of polynomials $P \circ \boldsymbol{g} \cup \{y_{ij}^\kappa - 1 : \ i \in [n],\ j \in [\ell_i]\}$. We alternate applications of Theorem 8 to pick x_{i_t} with applications of Lemma 9 to pick ω_t, and in the end we have a sequence of variables/values $x_{i_1} = \omega_1, \dots, x_{i_m} = \omega_m$. By these choices, the restricted set of polynomials $P\restriction_{x_{i_1}=\omega_1,\dots,x_{i_m}=\omega_m}$ requires refutations of reduced degree at least $D - 2\kappa m + 2m$. By Theorem 8, we can set $m = \ell^k n \ln(s)/D'$ for some $D' > 0$ and get a refutation of reduced degree at most $D' + d_0$. Hence, $D' + d_0 \geq D - 2m(\kappa - 1)$ and we get that $\ln(s) \geq \frac{D'(D-D'-d_0)}{2\ell^k n(\kappa-1)}$. The largest value is attained for $D' = (D - d_0)/2$ and we get $\ln(s) \geq \frac{(D-d_0)^2}{8\ell^k n(\kappa-1)}$. ◀

4 Degree lower bounds in $\mathsf{SoS}_{\mathbb{C}}$

In this section we prove Theorem 1, restated here for convenience of the reader.

▶ **Theorem 1** (Degree lower bound for $\mathsf{SRU}_n^{\kappa,r}$). *Let $n, d \in \mathbb{N}$, κ be a prime, $r \in \mathbb{C}$. Let r be written as $r_1 + \zeta r_2$, where $r_1, r_2 \in \mathbb{R}$ and ζ is some κth primitive root of unity. If*

$$\kappa d \leq \min\{r_1 + r_2 + (\kappa - 1)n + \kappa,\ n - r_1 - r_2 + \kappa\}\ ,$$

then there are no $\mathsf{SoS}_{\mathbb{C}}$-refutations of $\mathsf{SRU}_n^{\kappa,r}$ of degree at most d. In particular, $\mathsf{SRU}_n^{\kappa,0}$ requires refutations of degree $\Omega\left(\frac{n}{\kappa}\right)$ in $\mathsf{SoS}_{\mathbb{C}}$.

It is convenient to consider the following Boolean encoding of the sums of roots of unity,

$$\mathsf{bool\text{-}SRU}_n^{\kappa,r} := \left\{ \sum_{i\in[n]} \Big(\sum_{j\in[\kappa]} \zeta^{j-1} x_{ij} \Big) - r,\ x_{ij}^2 - x_{ij},\ \sum_{j\in[\kappa]} x_{ij} - 1\ :\ i \in [n], j \in [\kappa] \right\}\ . \quad (4)$$

The set of equations $\mathsf{SRU}_n^{\kappa,r}$ uses variables taking values in $\{1,\zeta,\zeta^2,\dots,\zeta^{\kappa-1}\}$, the encoding in eq. (4) uses indicator variables to select the appropriate power of ζ. It is easy to see that the degree needed to refute $\mathsf{SRU}_n^{\kappa,r}$ in $\mathsf{PC}_{\mathbb{C}}/\mathsf{SoS}_{\mathbb{C}}$ is at least the degree needed to refute $\mathsf{bool\text{-}SRU}_n^{\kappa,r}$ in $\mathsf{PC}_{\mathbb{C}}/\mathsf{SoS}_{\mathbb{C}}$. Hence, it is enough to show the degree lower bound for $\mathsf{bool\text{-}SRU}_n^{\kappa,r}$. To show this we construct a degree-d *pseudo-expectation* for $\mathsf{bool\text{-}SRU}_n^{\kappa,r}$, i.e., a linear operator $\tilde{\mathbb{E}}:\mathbb{C}[\boldsymbol{x}]\to\mathbb{C}$ such that

- $\tilde{\mathbb{E}}(1)=1$,
- $\tilde{\mathbb{E}}(mp)=0$, for every $p\in\mathsf{bool\text{-}SRU}_n^{\kappa,r}$ and m monomial such that $\deg(p)+\deg(m)\le d$,
- $\tilde{\mathbb{E}}(s\cdot s^*)\in\mathbb{R}_{\ge 0}$, for every polynomial s s.t. $\deg(s\cdot s^*)\le d$.

It is easy to see that the existence of a degree-d pseudo-expectation for a set of polynomials P implies that P cannot be refuted in degree-d $\mathsf{SoS}_{\mathbb{C}}$. The construction of an appropriate pseudo-expectation $\tilde{\mathbb{E}}$ for $\mathsf{bool\text{-}SRU}_n^{\kappa,r}$ is the goal of this section.

Some notation. In this section we consider fixed $r\in\mathbb{C}$ and $r_1,r_2\in\mathbb{R}$ such that $r=r_1+\zeta r_2$. Let $\boldsymbol{e}_j$ be the vector of dimension κ with the jth entry 1 and all other entries 0. For $j\in[\kappa]$, let $\boldsymbol{x}^{(j)}:=(x_{1j},\dots,x_{nj})$. That is, $\mathsf{bool\text{-}SRU}_n^{\kappa,r}$ is a set of polynomials in $\mathbb{C}[\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(\kappa)}]$. Given a tuple of sets $\boldsymbol{I}=(I_1,\dots,I_\kappa)$, where $I_j\subseteq[n]$, let $X_{\boldsymbol{I}}:=\prod_{j\in[\kappa]}\prod_{i\in I_j}x_{ij}$. With $\|\cdot\|$ we always denote the 1-norm. So $\|\boldsymbol{x}^{(j)}\|$ denotes the polynomial $\sum_{i\in[n]}x_{ij}$.

A potential satisfying assignment of $\mathsf{bool\text{-}SRU}_n^{\kappa,r}$ consists of $\boldsymbol{\gamma}=(\gamma_1,\dots,\gamma_\kappa)$, the allocation of the n roots of unity in the directions $\zeta^0,\dots,\zeta^{\kappa-1}$. The sum $\sum_{j\in[\kappa]}\zeta^{j-1}\gamma_j$ must be equal to the target value $r=r_1+\zeta r_2$, so we spread uniformly $n-r_1-r_2$ among the γ_js, and then add r_1 and r_2 to γ_1 and γ_2 respectively. This leads to the definitions

$$\begin{cases}\gamma_1=\frac{n-r_1-r_2}{\kappa}+r_1\ ,\\ \gamma_2=\frac{n-r_1-r_2}{\kappa}+r_2\ ,\\ \gamma_j=\frac{n-r_1-r_2}{\kappa}\quad\text{for } j\ge 3\ .\end{cases}\tag{5}$$

Observe that $\|\boldsymbol{\gamma}\|=n$. For ease of notation let $\hat{\gamma}=\frac{n-r_1-r_2}{\kappa}$ and $r_3=\dots=r_\kappa=0$. Therefore, we can write $\gamma_j=\hat{\gamma}+r_j$ for each $j\in[\kappa]$.

Given $\boldsymbol{I}=(I_1,\dots,I_\kappa)$ with $I_j\subseteq[n]$, and variables $\boldsymbol{v}=(v_1,\dots,v_\kappa)$, let $S(X_{\boldsymbol{I}})$ be the polynomial in the variables $\boldsymbol{v}$ defined by

$$S(X_{\boldsymbol{I}}):=\begin{cases}\dfrac{(n-|\bigcup_{j\in[\kappa]}I_j|)!}{n!}\displaystyle\prod_{j\in[\kappa]}\prod_{\ell=0}^{|I_j|-1}(v_j-\ell) & \text{if the sets in } \boldsymbol{I} \text{ are pair-wise disjoint}\ ,\\ 0 & \text{otherwise}\ .\end{cases}\tag{6}$$

By linearity, extend $S(\cdot)$ to all polynomials. That is, given $p=\sum_{\boldsymbol{I}}\alpha_{\boldsymbol{I}}X_{\boldsymbol{I}}$ with $\alpha_{\boldsymbol{I}}\in\mathbb{C}$, let $S(p):=\sum_{\boldsymbol{I}}\alpha_{\boldsymbol{I}}S(X_{\boldsymbol{I}})$. We define

$$\tilde{\mathbb{E}}(p):=S(p)(\boldsymbol{\gamma})$$

and we show that $\tilde{\mathbb{E}}$ is a pseudo-expectation for $\mathsf{bool\text{-}Kn}_n^{\kappa,r}$.

Let $\mathbb{B}$ be the ideal $\langle x_{ij}^2-x_{ij},\ x_{ij}x_{ij'}:i\in[n],\ j,j'\in[\kappa],\ j\ne j'\rangle$. Given polynomials $p,q\in\mathbb{C}[\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(\kappa)}]$, we use the notation $p\equiv q$ to denote that $p-q\in\mathbb{B}$.

▶ **Lemma 10.** *If $p\equiv q$ then $\tilde{\mathbb{E}}(p)=\tilde{\mathbb{E}}(q)$.*

Proof. By definition $p\equiv q$ means there exists a polynomial $s\in\mathbb{B}$ such that $p=q+s$. By construction, $\tilde{\mathbb{E}}$ maps to 0 every polynomial in $\mathbb{B}$, in particular $\tilde{\mathbb{E}}(s)=0$. By the linearity of $\tilde{\mathbb{E}}$, then $\tilde{\mathbb{E}}(p)=\tilde{\mathbb{E}}(q)$. ◀

From the definition of $\tilde{\mathbb{E}}$, it follows easily that the lifts of the polynomials in $\textsf{bool-SRU}_n^{\kappa,r}$ are mapped to 0 by $\tilde{\mathbb{E}}$.

▶ **Theorem 11.** *For every $\boldsymbol{I} = (I_1, \dots, I_\kappa)$ with $I_j \subseteq [n]$ and $i \in [n]$, and every $p \in \textsf{bool-SRU}_n^{\kappa,r}$, $\tilde{\mathbb{E}}(X_{\boldsymbol{I}} p) = 0$.*

Proof. The fact that $\tilde{\mathbb{E}}(X_{\boldsymbol{I}}(x_{ij}^2 - x_{ij})) = 0$ is immediate by the definition of $\tilde{\mathbb{E}}$.

Given $\boldsymbol{a} = (a_1, \dots, a_\kappa) \in [n]^\kappa$, let $E_{\boldsymbol{a}} := \frac{(n-\|\boldsymbol{a}\|)!}{n!} \prod_{j\in[\kappa]} \prod_{\ell=0}^{a_j-1} (\gamma_j - \ell)$. Notice that for every $j \in [\kappa]$, $E_{\boldsymbol{a}+\boldsymbol{e}_j} = E_{\boldsymbol{a}} \frac{\gamma_j - a_j}{n - \|\boldsymbol{a}\|}$. If the sets I_j are not pair-wise disjoint then, by definition, the pseudo-expectation is already 0, so it is enough to consider the case when the I_js are pair-wise disjoint.

Let $\boldsymbol{t} = (t_1, \dots, t_\kappa)$ where $t_j = |I_j|$. To show that $\tilde{\mathbb{E}}(X_{\boldsymbol{I}}(\sum_{j\in[\kappa]} x_{ij} - 1)) = 0$ we have two cases. If $i \in \bigcup_{j\in[\kappa]} I_j$, then

$$\tilde{\mathbb{E}}(X_{\boldsymbol{I}}(\sum_{j\in[\kappa]} x_{ij} - 1)) = E_{\boldsymbol{t}} - E_{\boldsymbol{t}} = 0 \ .$$

If $i \notin \bigcup_{j\in[\kappa]} I_j$, then

$$\tilde{\mathbb{E}}(X_{\boldsymbol{I}}(\sum_{j\in[\kappa]} x_{ij} - 1)) = \sum_{j\in[\kappa]} E_{\boldsymbol{t}+\boldsymbol{e}_j} - E_{\boldsymbol{t}} = E_{\boldsymbol{t}} \cdot \left(\sum_{j\in[\kappa]} \frac{\gamma_j - t_j}{n - \|\boldsymbol{t}\|} - 1 \right) = E_{\boldsymbol{t}} \cdot \left(\frac{\|\boldsymbol{\gamma}\| - \|\boldsymbol{t}\|}{n - \|\boldsymbol{t}\|} - 1 \right) = 0 \ ,$$

since $\|\boldsymbol{\gamma}\| = n$.
Finally we prove that $\tilde{\mathbb{E}}(X_{\boldsymbol{I}}(\sum_{j\in[\kappa]} \zeta^{j-1} \|\boldsymbol{x}^{(j)}\| - r_1 - \zeta r_2)) = 0$:

$$\begin{aligned}
\tilde{\mathbb{E}}(X_{\boldsymbol{I}}(\sum_{j\in[\kappa]} \zeta^{j-1} \|\boldsymbol{x}^{(j)}\| - r_1 - \zeta r_2)) &= E_{\boldsymbol{t}} \sum_{j\in[\kappa]} \zeta^{j-1} t_j + \sum_{i \notin \bigcup_{j\in[\kappa]} I_j} (\sum_{j\in[\kappa]} \zeta^{j-1} E_{\boldsymbol{t}+\boldsymbol{e}_j}) - (r_1 + \zeta r_2) E_{\boldsymbol{t}} \\
&= E_{\boldsymbol{t}} \sum_{j\in[\kappa]} \zeta^{j-1} t_j + (n - \|\boldsymbol{t}\|) \sum_{j\in[\kappa]} \zeta^{j-1} E_{\boldsymbol{t}+\boldsymbol{e}_j} - (r_1 + \zeta r_2) E_{\boldsymbol{t}} \\
&= E_{\boldsymbol{t}} \sum_{j\in[\kappa]} \zeta^{j-1} t_j + E_{\boldsymbol{t}} \sum_{j\in[\kappa]} \zeta^{j-1} (\gamma_j - t_j) - (r_1 + \zeta r_2) E_{\boldsymbol{t}} \\
&= E_{\boldsymbol{t}} \cdot \left(\sum_{j\in[\kappa]} \zeta^{j-1} t_j + \sum_{j\in[\kappa]} \zeta^{j-1} (\gamma_j - t_j) - (r_1 + \zeta r_2) \right) \\
&= E_{\boldsymbol{t}} \cdot \left(\sum_{j\in[\kappa]} \zeta^{j-1} \gamma_j - (r_1 + \zeta r_2) \right) \\
&= E_{\boldsymbol{t}} \cdot \left(\sum_{j\in[\kappa]} \zeta^{j-1} \hat{\gamma} + \sum_{j\in[\kappa]} \zeta^{j-1} r_j - (r_1 + \zeta r_2) \right) \\
&= 0 \ ,
\end{aligned}$$

since $\gamma_j = \hat{\gamma} + r_j$, $r_j = 0$ for $j > 2$, and $\sum_{j\in[k]} \zeta^{j-1} = 0$. ◀

This result, together with Theorem 12 below, implies that $\tilde{\mathbb{E}}$ is a degree-d pseudo-expectation for $\textsf{bool-SRU}_n^{\kappa,r}$, and therefore a degree-d lower bound for the refutations of $\textsf{bool-SRU}_n^{\kappa,r}$ and $\textsf{SRU}_n^{\kappa,r}$ in $\textsf{SoS}_{\mathbb{C}}$, i.e. Theorem 1. The idea is to use to Blekherman's approach in [27, Appendix B,C]. Let us recall first some useful notation.

Let $\mathfrak{S}_n$ be the symmetric group of n elements. For a set $J \subseteq [n]$ and a permutation $\sigma \in \mathfrak{S}_n$, let $\sigma J := \{\sigma(j) \,:\, j \in J\}$. Consider variables $\boldsymbol{y} = (y_1, \dots, y_n)$. For a set $J \subseteq [n]$ let $Y_J := \prod_{j \in J} y_j$. Given a polynomial $p \in \mathbb{C}[\boldsymbol{y}]$, that is $p(\boldsymbol{y}) = \sum_{J \subseteq [n]} p_J Y_J$, with $p_J \in \mathbb{C}$, let

$$\sigma p(\boldsymbol{y}) := \sum_J p_J Y_{\sigma J} \ .$$

Then define the *symmetrization* of p as the polynomial $\mathrm{Sym}(p) \in \mathbb{C}[\boldsymbol{y}]$ given by

$$\mathrm{Sym}(p)(\boldsymbol{y}) := \frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} \sigma p(\boldsymbol{y}) \ .$$

▶ **Theorem 12.** *For every polynomial $p \in \mathbb{C}[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\kappa)}]$ of degree at most d, if*

$$-(\kappa - 1)n + \kappa d - \kappa \le r_1 + r_2 \le n - \kappa d + \kappa \ ,$$

then $\tilde{\mathbb{E}}(p \cdot p^*) \ge 0$ where p^* is the formal conjugate of p.

Proof. Let $\boldsymbol{\gamma}$ be defined as in eq. (5), and recall $\hat{\gamma} = \frac{n - r_1 - r_2}{\kappa}$. Recall that the polynomial $S(X_{\boldsymbol{I}})$ when evaluated on $\boldsymbol{\gamma}$ is exactly $\tilde{\mathbb{E}}(X_{\boldsymbol{I}})$, see the comment after eq. (6). We have that

$$\begin{aligned}
\tilde{\mathbb{E}}(p \cdot p^*) &= S(p \cdot p^*)(\boldsymbol{\gamma}) && \text{[by the definition of } \tilde{\mathbb{E}}] \\
&= S(p \cdot p^*)(r_1 + \hat{\gamma}, r_2 + \hat{\gamma}, \dots, r_\kappa + \hat{\gamma}) && \text{[by the definition of } \boldsymbol{\gamma}] \\
&= \mathrm{Sym}(p\restriction_\rho \cdot p\restriction^*_\rho)(\hat{\gamma} \boldsymbol{e}_1) && \text{[by Theorem 14 below]} \\
&= \sum_{j=0}^{d} p_{d-j}(\hat{\gamma}) \cdot p^*_{d-j}(\hat{\gamma}) \prod_{i=0}^{j-1} (\hat{\gamma} - i)(n - \hat{\gamma} - i) \ , && \text{[by Theorem 13 below]}
\end{aligned}$$

where ρ is the substitution given by $\rho(x_{ij}) := y_i + \frac{r_j}{n}$ (recall that $r_3 = \cdots = r_\kappa = 0$). Now, $p_{d-j}(\hat{\gamma}) \cdot p^*_{d-j}(\hat{\gamma})$ is always real and non-negative since it is the module of the complex number $p_{d-j}(\hat{\gamma})$, hence to enforce the non-negativity of $\tilde{\mathbb{E}}(p \cdot p^*)$ it is enough to argue that $\prod_{i=0}^{j-1} (\hat{\gamma} - i)(n - \hat{\gamma} - i) \ge 0$. This is true if $\hat{\gamma} - d + 1 \ge 0$ and $n - \hat{\gamma} - d + 1 \ge 0$. I.e. if

$$-(\kappa - 1)n + \kappa d - \kappa \le r_1 + r_2 \le n - \kappa d + \kappa \ . \qquad ◀$$

▶ **Theorem 13** (adaptation of [27, Theorem B.11])**.** *Given variables $\boldsymbol{y} = (y_1, \dots, y_n)$ and $p, q \in \mathbb{C}[\boldsymbol{y}]$ with degree at most $d \le n/2$,*

$$\mathrm{Sym}(p \cdot p^*)(\boldsymbol{y}) \equiv \sum_{j=0}^{d} p_{d-j}(\|\boldsymbol{y}\|) \cdot p^*_{d-j}(\|\boldsymbol{y}\|) \prod_{i=0}^{j-1} (\|\boldsymbol{y}\| - i)(n - \|\boldsymbol{y}\| - i) \ ,$$

where p_{d-j} is a univariate polynomial with coefficients in $\mathbb{C}$, p^*_{d-j} is the formal conjugate of p_{d-j} and the degree of both polynomials is at most $(d - j)/2$.

This result is provable using exactly the same argument of Blekherman in [27, Theorem B.11], adapted to complex numbers.

▶ **Theorem 14.** *Given $p \in \mathbb{C}[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\kappa)}]$,*

$$S(p)(r_1 + \|\boldsymbol{y}\|, r_2 + \|\boldsymbol{y}\|, r_3 + \|\boldsymbol{y}\|, \dots, r_\kappa + \|\boldsymbol{y}\|) \equiv \mathrm{Sym}(p\restriction_\rho)(\boldsymbol{y}) \ ,$$

where ρ is the substitution given by $\rho(x_{ij}) := y_i + \frac{r_j}{n}$ (recall that $r_3 = \cdots = r_\kappa = 0$).

Proof. Given a vector of variables $\boldsymbol{y} = (y_1, \dots, y_m)$, let $\binom{\|\boldsymbol{y}\|}{t}$ be the polynomial

$$\binom{\|\boldsymbol{y}\|}{t} := \frac{\|\boldsymbol{y}\|(\|\boldsymbol{y}\| - 1) \cdots (\|\boldsymbol{y}\| - t + 1)}{t!} .$$

It holds that $\binom{\|\boldsymbol{y}\|}{t} \equiv \sum_{\substack{I \subseteq [n] \\ |I|=t}} Y_I$. (A proof of this fact is in the full version.) This immediately implies that

$$\prod_{j \in [\kappa]} \binom{\|\boldsymbol{x}^{(j)}\|}{t_j} \equiv \sum_{\substack{\boldsymbol{I}=(I_1,\dots,I_\kappa),\ I_j \subseteq [n] \\ |I_j|=t_j}} X_{\boldsymbol{I}} . \tag{7}$$

For a vector of sets $\boldsymbol{I} = (I_1, \dots, I_\kappa)$ and a permutation $\sigma \in \mathfrak{S}_n$, let $\sigma\boldsymbol{I} := (\sigma I_1, \dots, \sigma I_\kappa)$. Given a polynomial $p = \sum_{\boldsymbol{I}} p_{\boldsymbol{I}} X_{\boldsymbol{I}}$ in $\mathbb{C}[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\kappa)}]$ and a permutation $\sigma \in \mathfrak{S}_n$ let

$$\sigma p := \sum_{\boldsymbol{I}} p_{\boldsymbol{I}} X_{\sigma\boldsymbol{I}} .$$

Now, for any polynomial $p \in \mathbb{C}[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\kappa)}]$

$$\frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} \sigma p \equiv S(p)(\|\boldsymbol{x}^{(1)}\|, \dots, \|\boldsymbol{x}^{(\kappa)}\|) . \tag{8}$$

To see this equivalence, by linearity, it is enough to show that for every $\boldsymbol{I}$ with $I_j \subseteq [n]$

$$\frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} X_{\sigma\boldsymbol{I}} \equiv S(X_{\boldsymbol{I}})(\|\boldsymbol{x}^{(1)}\|, \dots, \|\boldsymbol{x}^{(\kappa)}\|) .$$

If the sets in $\boldsymbol{I}$ are not pair-wise disjoint it is immediate to see that $\frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} X_{\sigma\boldsymbol{I}} \in \mathbb{B}$, and therefore $\frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} X_{\sigma\boldsymbol{I}} \equiv 0$. Suppose then $\boldsymbol{I} = (I_1, \dots, I_\kappa)$ and the sets I_j are pair-wise disjoint. Let $t_j = |I_j|$, then

$$\begin{aligned}
\frac{1}{n!} \sum_{\sigma \in \mathfrak{S}_n} X_{\sigma\boldsymbol{I}} &= \frac{(n - \|\boldsymbol{t}\|)! \prod_{j \in [\kappa]} t_j!}{n!} \cdot \sum_{\substack{\boldsymbol{S}=(S_1,\dots,S_\kappa) \\ \text{pair-wise disj.} \\ |S_j|=t_j}} X_{\boldsymbol{S}} \\
&\equiv \frac{(n - \|\boldsymbol{t}\|)! \prod_{j \in [\kappa]} t_j!}{n!} \cdot \sum_{\substack{\boldsymbol{S}=(S_1,\dots,S_\kappa) \\ |S_j|=t_j}} X_{\boldsymbol{S}} \\
&\equiv \frac{(n - \|\boldsymbol{t}\|)!}{n!} \prod_{j \in [\kappa]} t_j! \cdot \prod_{j \in [\kappa]} \binom{\|\boldsymbol{x}^{(j)}\|}{t_j} \qquad (9) \\
&= S(X_{\boldsymbol{I}})(\|\boldsymbol{x}^{(1)}\|, \dots, \|\boldsymbol{x}^{(\kappa)}\|) ,
\end{aligned}$$

where the equality in eq. (9) follows from eq. (7).

To conclude, it is then enough to observe that the statement we want to prove follows from eq. (8) restricting both sides of the equality by ρ. To prove this we use that $\sigma X_{\boldsymbol{I}}\restriction_\rho = \sigma(X_{\boldsymbol{I}}\restriction_\rho)$. ◀

5 Conclusions

The study of algebraic proof systems under Fourier encoding is still at its infancy. There are many natural questions about its size efficiency. We understand reasonably well the strength relation between resolution and PC in the Boolean encoding. Sokolov [34] stresses that we do not even know yet whether PC with $\{\pm 1\}$ simulates resolution or not.

We mentioned already that the study of κ-COLORING of graphs is a very natural application of PC with Fourier encoding. There are some degree lower bounds in literature [26], but size lower bounds are still unknown. Understanding size would allow to understand larger classes of algebraic algorithms for this problem.

References

1 Albert Atserias and Tuomas Hakoniemi. Size-degree trade-offs for sums-of-squares and positivstellensatz proofs. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.24.

2 Albert Atserias and Joanna Ochremiak. Proof complexity meets algebra. *ACM Trans. Comput. Logic*, 20(1), December 2018.

3 Roberto J Bayardo Jr and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *AAAI/IAAI*, pages 203–208, 1997.

4 Christoph Berkholz. The Relation between Polynomial Calculus, Sherali-Adams, and Sum-of-Squares Proofs. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

5 Grigoriy Blekherman, João Gouveia, and James Pfeiffer. Sums of squares on the hypercube. *Mathematische Zeitschrift*, pages 1–14, 2016. doi:10.1007/s00209-016-1644-7.

6 Grigoriy Blekherman and Cordian Riener. Symmetric non-negative forms and sums of squares. *Discrete and Computational Geometry*, 65(3):764–799, May 2020. doi:10.1007/s00454-020-00208-w.

7 Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *J. Comput. Syst. Sci.*, 62(2):267–289, 2001. doi:10.1006/jcss.2000.1726.

8 Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 174–183. ACM, 1996.

9 John Conway and A. Jones. Trigonometric diophantine equations (on vanishing sums of roots of unity). *Acta Arithmetica*, 30(3):229–240, 1976. doi:10.4064/aa-30-3-229-240.

10 David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3rd edition*. Springer, 2007.

11 Jesús A De Loera, J. Lee, S. Margulies, and S. Onn. Expressing combinatorial problems by systems of polynomial equations and Hilbert's Nullstellensatz. *Comb. Probab. Comput.*, 18(4):551–582, July 2009. doi:10.1017/S0963548309009894.

12 Jesús A De Loera, Jon Lee, Peter N Malkin, and Susan Margulies. Computing infeasibility certificates for combinatorial problems through Hilbert's Nullstellensatz. *Journal of Symbolic Computation*, 46(11):1260–1283, 2011.

13 Jesús A De Loera, Susan Margulies, Michael Pernpeintner, Eric Riedl, David Rolnick, Gwen Spencer, Despina Stasi, and Jon Swenson. Graph-coloring ideals: Nullstellensatz certificates, Gröbner bases for chordal graphs, and hardness of Gröbner bases. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 133–140. ACM, 2015.

14 Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Dmitry Sokolov. The Power of Negative Reasoning. In *36th Computational Complexity Conference (CCC 2021)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:24, 2021. doi:10.4230/LIPIcs.CCC.2021.40.

15 R. Dvornicich and U. Zannier. Sums of roots of unity vanishing modulo a prime. *Archiv der Mathematik*, 79(2):104–108, August 2002. doi:10.1007/s00013-002-8291-4.

16 Roberto Dvornicich and Umberto Zannier. On sums of roots of unity. *Monatshefte für Mathematik*, 129(2):97–108, February 2000. doi:10.1007/s006050050009.

17 D. Grigoriev. Complexity of positivstellensatz proofs for the knapsack. *Computational Complexity*, 10(2):139–154, December 2001.

18 Dima Grigoriev. Tseitin's tautologies and lower bounds for Nullstellensatz proofs. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 648–652. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743515.

19 R. Impagliazzo, P. Pudlák, and J. Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, November 1999.

20 Daniela Kaufmann, Paul Beame, Armin Biere, and Jakob Nordström. Adding dual variables to algebraic reasoning for gate-level multiplier verification. In *Proceedings of the 25th Design, Automation and Test in Europe Conference (DATE'22)*, 2022.

21 Daniela Kaufmann and Armin Biere. Nullstellensatz-proofs for multiplier verification. In *Computer Algebra in Scientific Computing - 22nd International Workshop, CASC 2020, Linz, Austria, September 14-18, 2020, Proceedings*, pages 368–389, 2020. doi:10.1007/978-3-030-60026-6_21.

22 Daniela Kaufmann, Armin Biere, and Manuel Kauers. Verifying large multipliers by combining SAT and computer algebra. In *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 28–36, 2019. doi:10.23919/FMCAD.2019.8894250.

23 Daniela Kaufmann, Armin Biere, and Manuel Kauers. From DRUP to PAC and back. In *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*, pages 654–657, 2020. doi:10.23919/DATE48585.2020.9116276.

24 T.Y Lam and K.H Leung. On vanishing sums of roots of unity. *Journal of Algebra*, 224(1):91–109, 2000.

25 J. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. *Integer Programming and Combinatorial Optimization*, pages 293–303, 2001.

26 Massimo Lauria and Jakob Nordström. Graph Colouring is Hard for Algorithms Based on Hilbert's Nullstellensatz and Gröbner Bases. In *32nd Computational Complexity Conference (CCC 2017)*, volume 79, pages 2:1–2:20, 2017. doi:10.4230/LIPIcs.CCC.2017.2.

27 Troy Lee, Anupam Prakash, Ronald de Wolf, and Henry Yuen. On the sum-of-squares degree of symmetric quadratic functions. In *31st Conference on Computational Complexity*, volume 50 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 17, 31. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016.

28 João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 1999.

29 M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

30 Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96(2):293–320, 2003.

31 Aaron Potechin. Sum of Squares Bounds for the Ordering Principle. In Shubhangi Saraf, editor, *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:37, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2020.38.

32 Grant Schoenebeck. Linear level Lasserre lower bounds for certain k-CSPs. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 593–602. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.74.

33 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987. doi:10.1145/28395.28404.

34 Dmitry Sokolov. (Semi)Algebraic proofs over {±1} variables. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. ACM, June 2020.

35 Madhur Tulsiani. CSP gaps and reductions in the Lasserre hierarchy. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 303–312. ACM, 2009. doi:10.1145/1536414.1536457.

Complete ZX-Calculi for the Stabiliser Fragment in Odd Prime Dimensions

Robert I. Booth
University of Edinburgh, Edinburgh, United Kingdom
Sorbonne Université, CNRS, LIP6, 4 place Jussieu, F-75005 Paris, France
LORIA CNRS, Inria Mocqua, Université de Lorraine, F-54000 Nancy, France

Titouan Carette
Université Paris-Saclay, Inria, CNRS, LMF, 91190, Gif-sur-Yvette, France

Abstract

We introduce a family of ZX-calculi which axiomatise the stabiliser fragment of quantum theory in odd prime dimensions. These calculi recover many of the nice features of the qubit ZX-calculus which were lost in previous proposals for higher-dimensional systems. We then prove that these calculi are complete, i.e. provide a set of rewrite rules which can be used to prove any equality of stabiliser quantum operations. Adding a discard construction, we obtain a calculus complete for mixed state stabiliser quantum mechanics in odd prime dimensions, and this furthermore gives a complete axiomatisation for the related diagrammatic language for affine co-isotropic relations.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases ZX-calculus, completeness, quantum, stabiliser, qudits

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.24

Related Version *Full Version*: `https://arxiv.org/abs/2204.12531` [14]

Funding *Robert I. Booth*: ANR VanQuTe project (ANR-17-CE24-0035) and Cisco University Research Program Fund.

Acknowledgements We thank Cole Comfort and Simon Perdrix for enlightening discussions.

1 Introduction

The ZX-calculus is a powerful yet intuitive graphical language for reasoning about quantum computing, or, more generally, about operations on quantum systems [23, 46]. It allows one to represent such quantum operations pictorially, and comes equipped with a set of rules which, in principle, make it possible to derive any equality between those pictures [3, 38, 42]. It has now had several applications in quantum information processing, from MBQC [32, 5] through quantum error correction codes [31, 29, 20, 33]. More recently, it has been used to obtain state-of-the-art optimisation techniques for quantum circuits [39, 28, 30] and faster classical simulation algorithms for general quantum computations [40].

Despite its origins in categorical quantum mechanics and the diagrammatic language for finite-dimensional linear spaces [1, 22, 23] the literature on the ZX-calculus has been concerned almost exclusively with small-dimensional quantum systems, and even then mostly with the case of two-dimensional quantum systems, or qubits. The qubit ZX-calculus is remarkable in its simple treatment of stabiliser quantum mechanics, along with the fact that any diagram can be treated purely graph-theoretically, without concern to its overall layout, and without losing its quantum-mechanical interpretation. Those proposals that go beyond qubits lose many of these nice features, and are significantly more complicated than the qubit case [43, 51, 9, 50, 49]. In particular, they eschew the prised “Only Connectivity Matters” (OCM) meta-rule, often cited as one of the key features in the qubit case. In these calculi, which can represent any linear map between the corresponding Hilbert spaces, it is also not necessarily obvious (at least, to us) how to pick out and work with the stabiliser fragment.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Stabiliser quantum mechanics is a simple yet particularly important fragment of quantum theory. While much less powerful than the full fragment – it can be efficiently classically simulated, even in odd prime dimensions [27] – it has seen significant study [34, 36] and forms the basis for a number of key methods in quantum information theory [35]. Operationally, it can be described as the fragment of quantum mechanics which is obtained if one allows only state preparation and measurement in the computational basis, and unitary operations from the *Clifford groups* [35]. In the qubit case, the stabiliser fragment of the ZX-calculus was proved complete in [3] while ignoring global scalars, and extended to include scalars in [4]. A simplified calculus further reducing the set of axioms of the calculus was presented in [6].

In this article, we present a simple family of ZX-calculi which are complete for stabiliser quantum mechanics in odd prime dimensions, and which recover as many of the nice features of the qubit calculus as possible. In odd prime dimensions, stabiliser quantum mechanics can be given a particularly nice graphical presentation, owing to the group-theory underlying the corresponding Clifford groups [41, 2, 27]. We then give this calculus a set of rewrite rules that is complete, i.e. rich enough to derive any equality of stabiliser quantum operations. In particular, it is a design priority to recover OCM, and to make explicit the stabiliser fragment and its group-theoretical underpinnings. Adding a discard construction [18], we obtain a universal and complete calculus for mixed state stabiliser quantum mechanics in odd prime dimensions. By previous work [26], this gives a complete axiomatisation for the related diagrammatic language for affine co-isotropic relations, while still maintaining OCM.

Although we do not do so here, these calculi can naturally be extended to represent much larger fragments of quantum theory, up to the entire theory in odd prime dimensions [51]. However, finding a complete axiomatisation for such calculi will presumably be a much more complicated task, and we leave this for future work.

All proofs and additional appendices are available in the full version of the paper [14].

2 Stabiliser quantum mechanics in odd prime dimensions

Throughout this paper, p denotes an arbitrary odd prime, and $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ the ring of integers with arithmetic modulo p. We also put $\omega := e^{i\frac{2\pi}{p}}$, and let $\mathbb{Z}_p^*$ be the group of units of $\mathbb{Z}_p$. Since p is prime, $\mathbb{Z}_p$ is a field and $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ as a set. We also have need of the following definition:

$$\chi_p(x) = \begin{cases} 1 & \text{if} \quad \text{there is no } y \in \mathbb{Z}_p \text{ s.t. } x = y^2; \\ 0 & \text{otherwise;} \end{cases} \tag{1}$$

which is just the characteristic function of the complement of the set of squares in $\mathbb{Z}_p$.

The Hilbert space of a qupit [35, 52] is $\mathcal{H} = \text{span}\{|m\rangle \mid m \in \mathbb{Z}_p\} \cong \mathbb{C}^p$, and we write $U(\mathcal{H})$ the group of unitary operators acting on $\mathcal{H}$. We have the following standard operators on $\mathcal{H}$, also known as the *clock* and *shift* operators: $Z\,|m\rangle := \omega^m\,|m\rangle$ and $X\,|m\rangle := |m+1\rangle$ for any $m \in \mathbb{Z}_p$. In particular, note that $ZX = \omega XZ$.

We call any operator of the form $\omega^k X^a Z^b$ for $k, a, b \in \mathbb{Z}_p$ a *Pauli operator*. We say a Pauli operator is *trivial* if it is proportional to the identity. The collection of all Pauli operators is denoted $\mathscr{P}_1$ and called the *Pauli group*. For $n \in \mathbb{N}^*$, the *generalised Pauli group* is $\mathscr{P}_n := \bigotimes_{k=1}^n \mathscr{P}_1$.

Of particular importance to us are the *(generalised) Clifford groups*. These are defined, for each $n \in \mathbb{N}^*$, as the normaliser of $\mathscr{P}_n$ in $U(\mathcal{H}^{\otimes n})$: C is a Clifford operator if for any $P \in \mathscr{P}_n$, $CPC^\dagger \in \mathscr{P}_n$. It is clear that that every Pauli operator is Clifford, but there are non-Pauli Clifford operators. An important example is the *Fourier* gate: $F\,|m\rangle = \frac{1}{\sqrt{d}} \sum_{n\in\mathbb{Z}_p} \omega^{mn}\,|n\rangle$

which is such that $FXF^\dagger = Z$ and $FZF^\dagger = X^{-1}$. We also need the *phase* gate: $S\,|m\rangle = \omega^{2^{-1}m(m+1)}\,|m\rangle$ such that $SXS^\dagger = \omega XZ$ and $SZS^\dagger = Z$. Yet another important example is the *controlled-phase* gate, which acts on $\mathcal{H}\otimes\mathcal{H}$, $E\,|m\rangle\,|n\rangle \coloneqq \omega^{mn}\,|m\rangle\,|n\rangle$. It is important to emphasise a key difference between the qupit and the qubit case: when $p \neq 2$, none of these operators are self-inverse. In fact, if Q is a Pauli and I the identity operator on $\mathcal{H}$, we have $Q^p = I$, $E^p = I \otimes I$ and $F^4 = I$.

As a side note, these equations imply that both X and Z, and in fact every Pauli, have spectrum $\{\omega^k \mid k \in \mathbb{Z}_p\}$. As a result, we denote $|k : Q\rangle$ the eigenvector of a given Pauli Q associated with eigenvalue ω^k, and furthermore use the notation $|k,\ldots,k : Q\rangle = \bigotimes_{k=1}^n |k : Q\rangle$. It follows from the definition of Z that we can identify $|k : Z\rangle = |k\rangle$.

Now, for any $\alpha \in [0, 2\pi)$, the operator $e^{i\alpha}I$ is Clifford. However, we want to construct calculi with a finite axiomatisation. As a result, the diagrams in the calculus are countable and this makes it impossible for us to represent all such phases $e^{i\alpha}$. Unfortunately, finding a group of phases that behaves well diagrammatically is somewhat inconvenient, and involves some elementary number theory. This is why, for an odd prime p, we consider the group of phases generated by the composition of the previously defined Clifford gates. Explicitly, it is given by:

if $p \equiv 1 \mod 4$, $\mathbb{P}_p \coloneqq \{(-1)^s\omega^t \mid s,t \in \mathbb{Z}\}$

if $p \equiv 3 \mod 4$, $\mathbb{P}_p \coloneqq \{i^s\omega^t \mid s,t \in \mathbb{Z}\}$

This of course covers all cases since p is odd. Then, we restrict our attention to the *reduced* Clifford group, $\mathscr{C}_n = \{\lambda C \mid \lambda \in \mathbb{P}_p, C \in U(\mathcal{H}^{\otimes n})$ is Clifford and special unitary$\}$. We call $\mathscr{C}_1^{\otimes n}$ the *local Clifford group* on n qupits. It is clear from these examples that $\mathscr{C}_n$ is strictly larger than $\mathscr{C}_1^{\otimes n}$, but it turns out to not be that much larger:

▶ **Proposition 1** ([21, 41]). *The reduced Clifford group $\mathscr{C}_n$ is generated by the gate-set $\{F_j, S_j, E_{j,k} \mid j,k = 1,\ldots,n\}$.*

Stabiliser quantum mechanics can be operationally described as the fragment of quantum mechanics in which the only operations allowed are initialisations and measurements in the eigenbases of Pauli operators, and unitary operations from the generalised Clifford groups. As before, we restrict our attention to the fragment of stabiliser quantum mechanics where only unitary operations from the reduced Clifford group are allowed. Scalars are then taken from the monoid $\mathbb{G}_p \coloneqq \{0, \sqrt{p^r}\lambda \mid r \in \mathbb{Z}, \lambda \in \mathbb{P}_p\}$. Little is lost for the description of quantum algorithms, since we can always simplify by a global phase to make the Clifford generators special unitary. Thus, we can embed any stabiliser circuit into the calculus, and then calculate the relative phases of different branches of a computation without restriction.

▶ **Definition 2.** *The symmetric monoidal category* Stab_p *has as objects $\mathbb{C}^{pn}$ for each $n \in \mathbb{N}$, and morphisms generated by:*

- $\mathbb{C} \to \mathbb{C}^p : \lambda \mapsto \lambda\,|0\rangle$;
- $\mathbb{C}^{pn} \to \mathbb{C}^{pn} : |\psi\rangle \mapsto U\,|\psi\rangle$ *for any* $U \in \mathscr{C}_n$;
- $\mathbb{C}^p \to \mathbb{C} : |\psi\rangle \mapsto \langle 0|\psi\rangle$.

The monoidal product is given by the usual tensor product of linear spaces.

It is clear that Stab_p is a subcategory of the category FLin of finite dimensional $\mathbb{C}$-linear spaces; it is also a PROP.

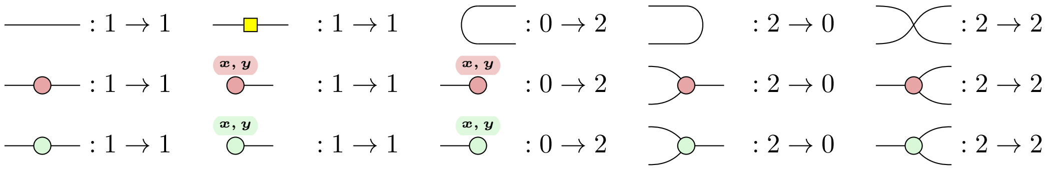
3 A ZX-calculus for odd prime dimensions

In this section, we present our family of ZX-calculi, with one for each odd prime. Relying on some of the group theoretical properties of the qupit Clifford groups, we can give a relatively simple presentation of the calculi, which avoids the need to explicitly consider rotations in p-dimensional space, significantly simplifing the presentation compared to previous work [43]. These calculi are also constructed in order to satisfy the property of *flexsymmetry*, proposed in [15, 16], and which allows one to recover the OCM meta-rule. OCM is an intuitively desirable feature for the design of a graphical language; anecdotally, it greatly simplifies the human manipulation of diagrams, including in the proofs of this paper. More formally, it means that the equational theory can be formalised in terms of double pushout rewriting over graphs rather than over hypergraphs as is necessary in the more general theory [11, 10, 12].

Another key concern is the issue of completeness, which we begin to address in this article. Outside of qubits [38, 42, 47], there has so far been a complete axiomatisation only for the stabiliser fragment in dimension $p = 3$ [48]. We present an axiomatisation which is complete for the stabiliser fragment for any odd prime p, and leave the general case for future work.

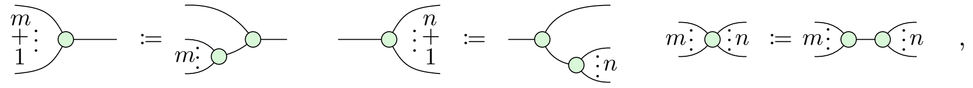
3.1 Generators

For any odd prime p, consider the symmetric monoidal category $\mathsf{ZX}_p^{\mathrm{Stab}}$ with objects $\mathbb{N}$ and morphisms generated by:



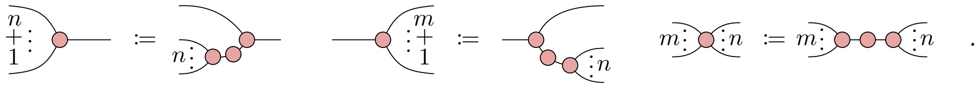
where $x, y \in \mathbb{Z}_p$. We also introduce a generator $\star : 0 \to 0$ to simplify the calculus; it will correspond to a scalar whose representation in terms of the other generators depends non-trivially on the dimension p. Morphisms are composed by connecting output wires to input wires, and the monoidal product is given on objects by $n \otimes m = n + m$ and on morphisms by vertical juxtaposition of diagrams.

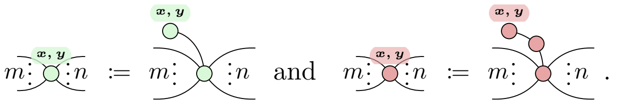
We extend this elementary notation with a first piece of syntactic sugar, which is standard for the ZX-calculus family: *green spiders* are defined inductively, for any $m, n \in \mathbb{N}$, by



and it is clear that these have types $m + 1 \to 1$, $1 \to n + 1$ and $m \to n$ respectively.

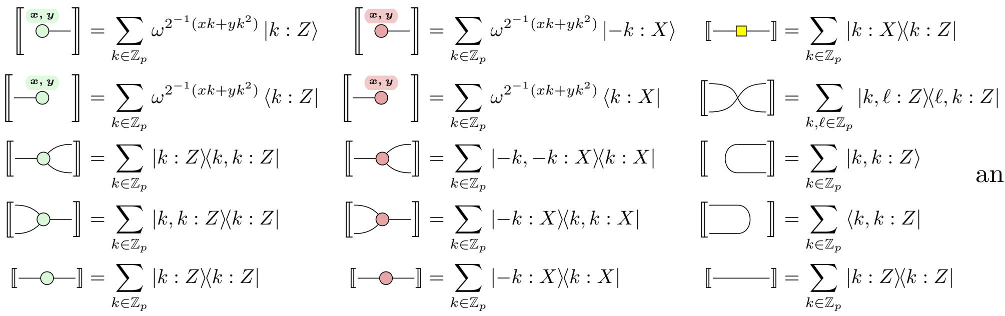
Red spiders are defined analogously to ZH-calculus harvestmen:



Labelled spiders are given by: 

3.2 Standard interpretation and universality

The standard interpretation of a $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram is a symmetric monoidal functor $[\![-]\!]$: $\mathsf{ZX}_p^{\mathrm{Stab}} \to \mathsf{FLin}$ (the category of finite-dimensional complex Hilbert spaces). It is defined on objects as $[\![m]\!] := \mathbb{C}^{p \times m}$, and on the generators of the morphisms as:



$$= \sum_{k\in\mathbb{Z}_p} \omega^{2^{-1}(xk+yk^2)} |k:Z\rangle \qquad = \sum_{k\in\mathbb{Z}_p} \omega^{2^{-1}(xk+yk^2)} |-k:X\rangle \qquad = \sum_{k\in\mathbb{Z}_p} |k:X\rangle\langle k:Z|$$

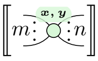
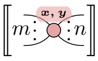
$$= \sum_{k\in\mathbb{Z}_p} \omega^{2^{-1}(xk+yk^2)} \langle k:Z| \qquad = \sum_{k\in\mathbb{Z}_p} \omega^{2^{-1}(xk+yk^2)} \langle k:X| \qquad = \sum_{k,\ell\in\mathbb{Z}_p} |k,\ell:Z\rangle\langle \ell,k:Z|$$

$$= \sum_{k\in\mathbb{Z}_p} |k:Z\rangle\langle k,k:Z| \qquad = \sum_{k\in\mathbb{Z}_p} |-k,-k:X\rangle\langle k:X| \qquad = \sum_{k\in\mathbb{Z}_p} |k,k:Z\rangle$$

and

$$= \sum_{k\in\mathbb{Z}_p} |k,k:Z\rangle\langle k:Z| \qquad = \sum_{k\in\mathbb{Z}_p} |-k:X\rangle\langle k,k:X| \qquad = \sum_{k\in\mathbb{Z}_p} \langle k,k:Z|$$

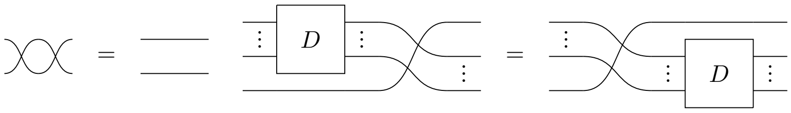
$$= \sum_{k\in\mathbb{Z}_p} |k:Z\rangle\langle k:Z| \qquad = \sum_{k\in\mathbb{Z}_p} |-k:X\rangle\langle k:X| \qquad = \sum_{k\in\mathbb{Z}_p} |k:Z\rangle\langle k:Z|$$

$[\![\star]\!] = -1$. Then, by the functoriality of the standard interpretation, we deduce that  $= \sum_{k\in\mathbb{Z}_p} \omega^{2^{-1}(xk+yk^2)} |k:Z\rangle^{\otimes n} \langle k:Z|^{\otimes m}$ and  $= \sum_{k\in\mathbb{Z}_p} \omega^{2^{-1}(xk+yk^2)} |-k:X\rangle^{\otimes n} \langle k:X|^{\otimes m}$.

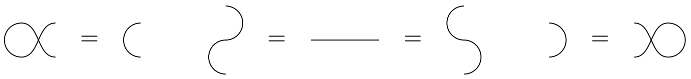
▶ **Theorem 3** (Universality). *The standard interpretation $[\![-]\!]$ is universal for the qupit stabiliser fragment, i.e. for any stabiliser operation $C : \mathbb{C}^{pm} \to \mathbb{C}^{pn}$ there is a diagram $D \in \mathsf{ZX}_p^{\mathrm{Stab}}$ such that $[\![D]\!] = C$. Put formally, the co-restriction of $[\![-]\!]$ to Stab_p is full.*

3.3 Axiomatisation

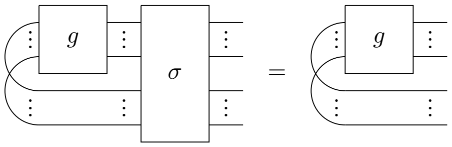
We now begin to introduce rewrite rules with which to perform a purely diagrammatic reasoning. By doing so we are in fact describing a PROP by generators and relations [7], thus the swap is required to verify the following properties:



Note that the last equation is required to hold for any diagram $D : n \to m$. This property states that our diagrams form a symmetric monoidal category. Furthermore, we want this category to be self-dual compact-closed, hence the cup and cap must verify:



Furthermore, as long as the connectivity of the diagram remains the same, vertices can be freely moved around without changing the standard interpretation of the diagram. This is a consequence of the fact that we require our generators to be *flexsymmetric*, as shown in [15]. This amounts to imposing that all generators except the swap verify:



where $\sigma : n+m \to n+m$ stands for any permutation of the wires involving swap maps. We will consider all the previous rules as being purely structural and will not explicitly state their use. Using these rules, we can in fact deduce that both the green and red spiders (and

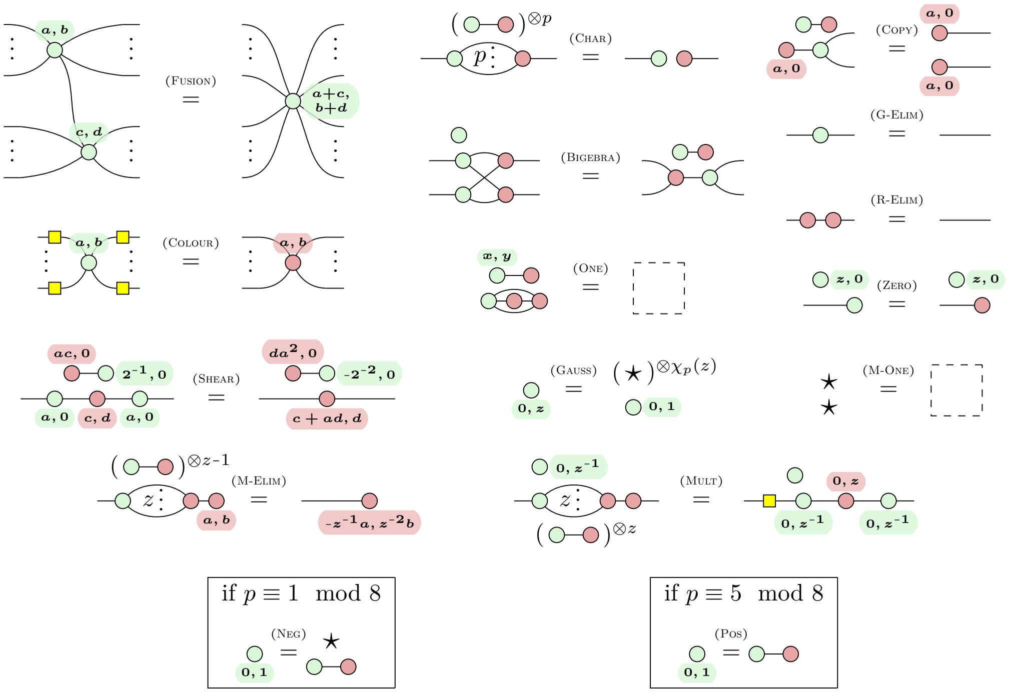


Figure 1 A presentation of the equational theory zx_p, which is sound and complete for the stabiliser fragment. The equations hold for any $a, b, c, d \in \mathbb{Z}_p$ and $z \in \mathbb{Z}_p^*$. χ_p is the characteristic function of the complement of the set of squares in $\mathbb{Z}_p$, defined in equation (1).

their labelled varieties) are themselves flexsymmetric. This means that the language follows the OCM meta-rule, and we can formally treat any $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram as a graph[1] whose vertices are the spiders, and whose edges are labelled by the $1 \to 1$ generators of the language.[2]

Figure 1 presents the remainder of the equational theory zx_p, which as we shall see axiomatises the stabiliser fragment of quantum mechanics in the qupit ZX-calculus. Firstly though, we must be sure that all of these rules are sound for the standard interpretation, i.e. it should not be possible to derive an equality of diagrams whose quantum mechanical interpretations are different.

▶ **Theorem 4** (Soundness). *The equational theory* zx_p *is sound for* $[\![-]\!]$, *i.e., for any* $A, B \in \mathsf{ZX}_p^{\mathrm{Stab}}$, $\mathrm{zx}_p \vdash A = B$ *implies* $[\![A]\!] = [\![B]\!]$. *Put formally,* $[\![-]\!]$ *factors through the projection* $\mathsf{ZX}_p^{\mathrm{Stab}} \to \mathsf{ZX}_p^{\mathrm{Stab}}/\mathrm{zx}_p$.

This set of rewriting rule also turns out to be also complete:

▶ **Theorem 5** (Completeness). *The equational theory* zx_p *is complete for* $[\![-]\!]$, *i.e., for any* $A, B \in \mathsf{ZX}_p^{\mathrm{Stab}}$, $[\![A]\!] = [\![B]\!]$ *implies* $\mathrm{zx}_p \vdash A = B$.

The proof of Theorem 5 will be the object of the following sections.

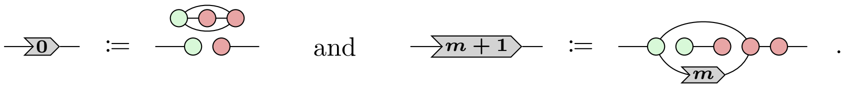
[1] Note that the graphs in question must be allowed to have loops and parallel edges, so are perhaps better called *pseudographs* or *multigraphs*.

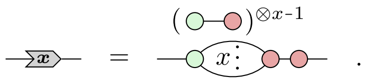
[2] There is a small ambiguity: $1 \to 1$ spiders can be treated as either edges or vertices. When considering diagrams, it matters little which, since any given graph is always to be understood as one of the many equivalent $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagrams constructed formally out of the generators. Any computer implementation of the calculus will have to carefully resolve this ambiguity in its internal representation.

3.4 Syntactic sugar for multi-edges

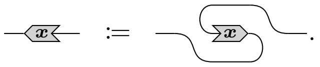
Before we move into the proof of completeness, we introduce some syntactic sugar to the language. Given axiom (Char), in $\mathsf{ZX}_p^{\mathrm{Stab}}$, unlike the qubit case, spiders of opposite colours can be connected by more that one edge, and these multi-edges cannot be simplified. We therefore add some syntactic sugar to represent such multi-edges. These constructions add no expressiveness to the language, and are simply used to reduce the size of some recurring diagrams. They are shamelessly stolen from previous work [13, 53, 17, 19], and we use them to obtain a particularly nice representation of qupit graph states [54]. Graph states a central role in our proof of completeness, as they have in previous completeness results of the stabiliser fragments for dimensions 2 and 3 [3, 30, 48]. In particular, these constructions permit a nice presentation of how graph states evolve under local Clifford operations.

Firstly, we extend $\mathsf{ZX}_p^{\mathrm{Stab}}$ by *multipliers*, which are defined inductively by:

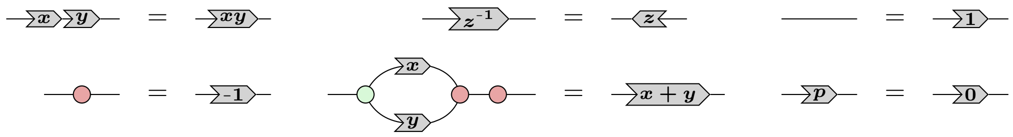


Explicitly, then, for $x \in \mathbb{Z}_p^*$, 

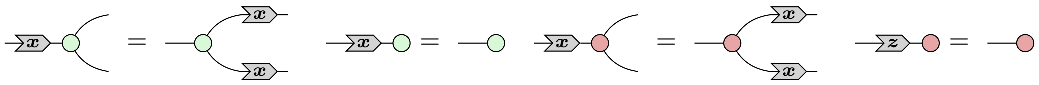
We also define inverted multipliers, using the standard notation for graphical languages based on symmetric monoidal categories:

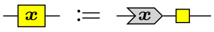
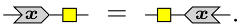


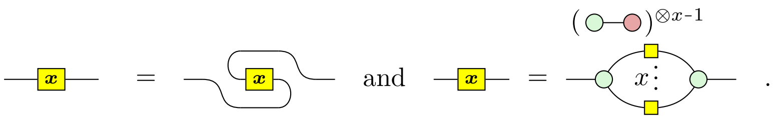
▶ **Proposition 6.** *Multipliers verify the following equations under* zx_p*: for any* $x, y \in \mathbb{Z}_p$ *and* $z \in \mathbb{Z}_p^*$*,*



which amounts to saying that the multipliers form a presentation of the field $\mathbb{Z}_p$*. They also verify the following useful copy and elimination identities:*

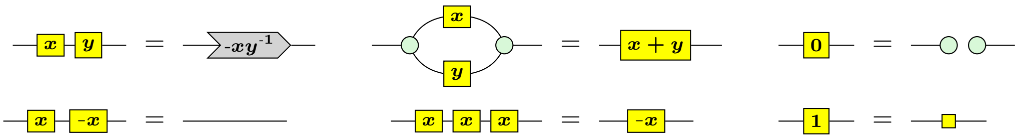


We can also unambiguously define *Fourier boxes*:  since . Fourier boxes exactly match the labelled "Hadamard" boxes introduced for the qutrit case [45] when $p = 3$. Owing to the flexsymmetry of the language, we have that for any $x \in \mathbb{Z}_p$,



(2)

▶ **Proposition 7.** zx_p *proves the following equations:*



(3)

4 Completeness

We now have all the necessary tools to show that our equational theory is complete. The structure of our proof is similar to the one used to show the completeness in the qubit case [3]. However, if the overall scheme is very similar, each step separately can involve different approaches more suited to the qupit situation. The plan is as follows. We identify a family of scalars, called *elementary scalars*, which correspond to those which appear when applying the rewrites of zx_p. We first show the completeness up to non-zero elementary scalars, which allows us to work with simpler diagrams without taking care of all the invertible scalars appearing along the way. Then, we show the completeness for elementary scalars independently, leading to a general proof of completeness. The proof of completeness up to non-zero elementary scalars goes something like this:

1. Take two diagrams with the same interpretation.
2. Put them into a simplified form, called the *GS+LC* form in [3].
3. Define the notion of simplified GS+LC form for a pair of diagrams in which some vertices are marked. Then show that in a simplified GS+LC pair if a vertex is marked on one side, it must also be marked on the other side, else the two diagrams cannot have the same interpretation.
4. Show that two diagram forming an rGS+LC pair such that their marked vertices matches and having the same interpretation are equal modulo the equational theory.

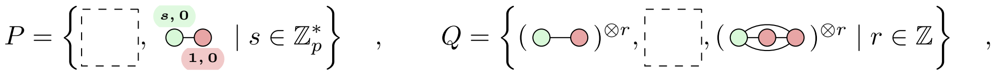
4.1 Elementary scalars

The following is standard from categorical quantum mechanics:

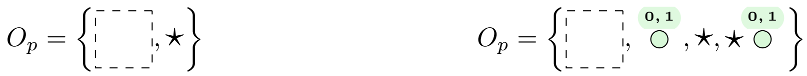
▶ **Lemma 8.** *If $A, B \in \mathsf{ZX}_p^{\mathrm{Stab}}[0,0]$, then $[\![A \otimes B]\!] = [\![A]\!] \cdot [\![B]\!] = [\![A \circ B]\!]$, where $\cdot$ is the usual multiplication on $\mathbb{C}$ restricted to the monoid $\mathbb{G}$.*

Now, as when we were defining the group of phases $\mathbb{P}_p$, the set of normal forms for phases must depend on the prime p in question:

▶ **Definition 9.** *An* elementary scalar *is a diagram $A \in \mathsf{ZX}_p^{\mathrm{Stab}}[0,0]$ which is a tensor product of diagrams from the collection $O_p \cup P \cup Q$: where*



if $p \equiv 1 \mod 4$, *if $p \equiv 3 \mod 4$,*



If $A, B \in \mathsf{ZX}_p^{\mathrm{Stab}}$, we say that A and B are equal up to an elementary scalar *if there is an elementary scalar C such that $A = B \otimes C$. In that case, we write $A \simeq B$.*

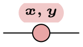
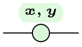
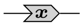
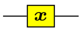
Comparing with the definition of $\mathbb{G}$, the interpretation of the elements of P correspond to powers of ω, the elements of Q to (possible negative) powers of $\sqrt{p}$, and the elements of O_p to powers of -1 or i depending on the value of p. This remark will naturally lead to the normal for for scalars in a few sections.

Now, as written, equality up to an elementary scalar might seem like a relation that is not symmetric and therefore not an equivalence relation.

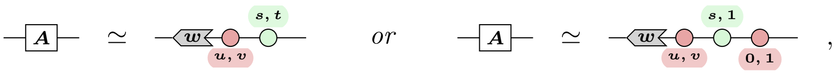
▶ **Proposition 10.** *Every elementary scalar $C \in \mathsf{ZX}_p^{\mathrm{Stab}}[0,0]$ has a multiplicative inverse, i.e. an elementary scalar $C^{-1} \in \mathsf{ZX}_p^{\mathrm{Stab}}[0,0]$ such that $C \otimes C^{-1} = C \circ C^{-1} =$* *.*

In light of this fact, if $A \simeq B$, there is an elementary scalar C such that $A = B \otimes C$, and then $B = B \otimes C \otimes C^{-1} = A \otimes C^{-1}$, so that $B \simeq A$.

▶ **Proposition 11.** *Every equation in zx_p can be loosened to equality up to an elementary scalar by erasing every part of the LHS and RHS diagrams which is disconnected from the inputs and outputs.*

Probably the most important case of equality up to elementary scalars is the completeness of the single-qupit Clifford groups, on which the entire proof of completeness of the calculus rests. The fragment of $\mathsf{ZX}_p^{\mathrm{Stab}}$ which corresponds to $\mathscr{C}_1$ is that generated by the $1 \to 1$ diagrams: , ,  and . We call any such diagram a *single-qupit Clifford diagram* or $\mathscr{C}_1$-diagram.

▶ **Proposition 12.** *If $A \in \mathsf{ZX}_p^{\mathrm{Stab}}[0,0]$ is a single-qupit Clifford diagram, then zx_p proves that*

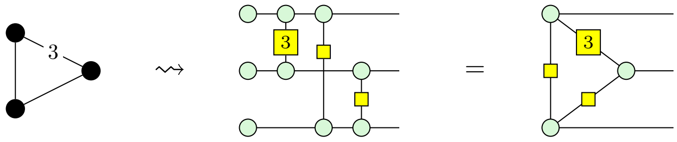


(4)

for some $s, t, u, v \in \mathbb{Z}_p$ and $w \in \mathbb{Z}_p^*$. Furthermore, this form is unique.

4.2 Relating stabiliser diagrams to graphs

The completeness proof begins by relating every diagram to simpler one in the form of a graph state diagram. Graph states are defined as usual – diagrammatically, we have an embedding of graphs given (informally) by:



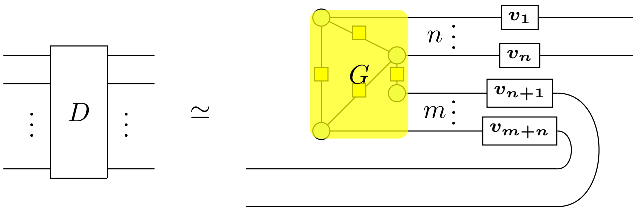
(5)

More specifically, we want to relate the diagram to the following form:

▶ **Definition 13** ([3, 48])**.** *A* GS+LC diagram *is a $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram which consists of a graph state diagram with arbitrary single-qupit Clifford operations applied to each output. These associated Clifford operations are called* vertex operators.

▶ **Proposition 14.** *Every $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram $0 \to n$ can be rewritten, up to elementary scalars, to GS+LC form under zx_p.*

In other words, zx_p proves that, for any stabiliser $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram $D : m \to n$, there is a graph G on $m+n$ vertices and a set $(v_k)_{k=1}^{m+n}$ of $\mathscr{C}_1$-diagrams such that



(6)

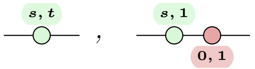
and we need only consider the question of whether zx_p can prove the equality of two GS+LC diagrams.

4.3 Completeness modulo elementary scalars

Now, as a first step, we show that zx_p can normalise any pair of diagrams with equal interpretations, up to elementary scalars. In particular, as was shown in the previous section, we can relax zx_p to reason about equality up to elementary scalars by simply ignoring the scalar part of each rule, and make free use of the "scalarless" equational theory. We will take care of the resulting scalars in the next section.

This part of the completeness proof follows the general ideas of [3]. The first step on the way to completeness is to note that, considering a diagram in GS+LC form, where the vertex operators have been normalised, we can obtain a yet more reduced diagram by absorbing as much as possible of the vertex operators into local scalings and local complementations. We then obtain the following form for the vertex operators:

▶ **Definition 15** ([3]). *A $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram is in* reduced *GS+LC (or rGS+LC) form if it is in GS+LC form, and furthermore:*

1. *All vertex operators belong to the following set:* $R = \Bigg\{$ 

$\Bigg|\, s, t \in \mathbb{Z}_p \Bigg\}$.
2. *Two adjacent vertices do not have vertex operators that both include red spiders.*

▶ **Proposition 16.** *If $D \in \mathsf{ZX}_p^{\mathrm{Stab}}$ is a Clifford diagram, then there is a diagram $G \in \mathsf{ZX}_p^{\mathrm{Stab}}$ in rGS+LC form such that $\mathrm{zx}_p \vdash D \simeq G$.*

Then, given two diagrams with equal interpretations, taking them both to rGS+LC makes the task of comparing the diagrams considerably easier. In particular, we can guarantee that the corresponding vertex operators in each diagram always have matching forms:

▶ **Definition 17** ([3]). *A pair of rGS+LC of the same type (i.e. whose graphs have the same vertex set V) is said to be* simplified *if there is no pair of vertices $q, p \in V$ such that q has a red vertex operator in the first diagram but not the second, q has a red vertex operator in the second diagram but not the first, and q and p are adjacent in at least one of the diagrams.*

▶ **Proposition 18.** *Any pair A, B of rGS+LC diagrams of the same type (i.e. on the same vertex set) can be simplified.*

For the sake of clarity, we shall say that a vertex operator (or equivalently, the vertex itself) is *marked* if it contains a red spider (i.e. it belongs to the right-hand form of definition 15). Then, two diagrams with the same interpretation can always be rewriten so that the marked vertices match:

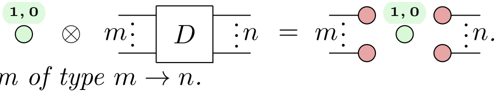
▶ **Proposition 19.** *Let $C, D \in \mathsf{ZX}_p^{\mathrm{Stab}}$ be a simplified pair in rGS+LC form, then $[\![C]\!] = [\![D]\!]$ only if the marked vertices in C and D are the same.*

We have enough control over the pair of diagrams to finish the completeness proof:

▶ **Theorem 20.** zx_p *is complete for $[\![-]\!]$, i.e. if for any pair of diagrams $A, B \in \mathsf{ZX}_p^{\mathrm{Stab}}[0, n]$ with $n \neq 0$, $[\![A]\!] = [\![B]\!]$, then $\mathrm{zx}_p \vdash A \simeq B$.*

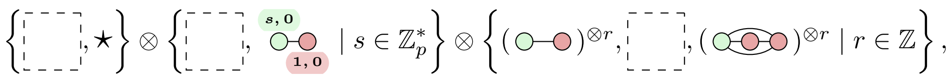
4.4 Completeness of the scalar fragment

Finally, we are ready to leap-frog off of the previous section into the full completeness (including scalars). First, we need to find a normal form for diagrams which evaluate to 0. In fact, we need pick one normal form for each type $m \to n$:

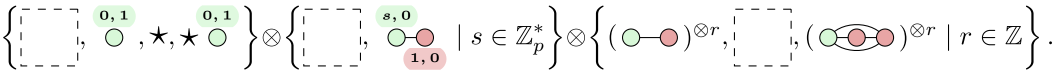
▶ **Proposition 21.** *The zero scalar "destroys" diagrams: for any $m, n \in \mathbb{N}$ and $D \in \mathsf{ZX}_p^{\mathrm{Stab}}[m,n]$,* 

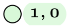
We take the RHS diagram to be the "zero" diagram of type $m \to n$.

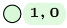
A scalar diagram is in *normal form* if it is either the zero scalar, or it belongs to the set



when $p \equiv 1 \mod 4$, or to the set



when $p \equiv 3 \mod 4$. It is straightforward to see, by evaluating $[\![-]\!]$ on each element, that these sets contain exactly one diagram for each scalar in $\mathbb{G}_p \setminus \{0\}$ (and the zero scalar  corresponds to $0 \in \mathbb{G}_p$).

▶ **Theorem 22.** zx_p *proves any scalar diagram equal to a scalar in normal form (depending on the congruence of p modulo 4), or to the zero scalar*  .

The completeness for the whole stabiliser fragment follows immediately, combining theorems 20 and 22:

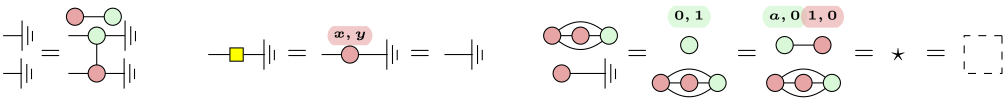
▶ **Theorem 23.** *The equational theory* zx_p *is complete for* Stab_p*, i.e. for any* $\mathsf{ZX}_p^{\mathrm{Stab}}$*-diagrams A and B, if $[\![A]\!] = [\![B]\!]$, then $\mathrm{zx}_p \vdash A = B$.*

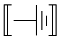
5 Mixed states and co-isotropic relations

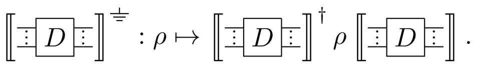
In this last section we use the work of [18] to extend our completeness result to the mixed-state case. We then unravel the connection to the Lagrangian relation investigated in [26].

5.1 A complete graphical language for CPM(Stab_p)

We now extend our completeness result form Stab_p to CPM(Stab_p), the category of completely positive maps corresponding to mixed state stabiliser quantum mechanics, see [44, 23] for a formal definition. We will rely on the discard construction of [18] to define a graphical language $(\mathsf{ZX}_p^{\mathrm{Stab}})^{\bar{\bar{\pm}}}$. It consists in adding to the equational theory one generator, the discard $\bar{\bar{\pm}} : 1 \to 0$ and equations stating that this generator erases all isometries. In Stab_p, the isometries are generated by a small family of diagrams, the equations to add are then:



A new interpretation $[\![\_]\!] : \mathsf{ZX}_p^{\mathrm{Stab}\bar{\bar{\pm}}} \to \mathrm{CPM}(\mathsf{Stab}_p)$ is defined as  $: \rho \mapsto \mathrm{Tr}(\rho)$ for the ground and for all $\mathsf{ZX}_p^{\mathrm{Stab}}$-diagram $D : n \to m$:



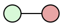
Corollary 22 of [18] provides a sufficient condition for the previous construction to extend to a universal and complete graphical language for Stab_p into a universal complete graphical language for $\mathrm{CPM}(\mathsf{Stab}_p)$. This condition is for Stab_p to have *enough isometries* in the sense of [18], By similar arguments as is the qubit case, we can show that Stab_2 indeed has enough isometries and then it follows that:

▶ **Theorem 24.** $(\mathsf{ZX}_p^{\mathrm{Stab}})^{\doteqdot}$ *is universal and complete for* $\mathrm{CPM}(\mathsf{Stab}_p)$.

5.2 Co-isotropic relations

It has been shown in [26, 25, 24] that $\mathrm{CPM}(\mathsf{Stab}_p)$ is equivalent to the category of affine co-isotropic relations up to scalars. More formally, we endow $\mathbb{Z}_p^2$ with the symplectic form: $\omega\left(\begin{bmatrix}a\\b\end{bmatrix},\begin{bmatrix}c\\d\end{bmatrix}\right) = ad - bc$, and $\mathbb{Z}_p^{2m} = \bigoplus_m \mathbb{Z}_p^2$ with the direct sum symplectic form.

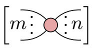
▶ **Definition 25.** *The symmetric monoidal category* $\mathsf{AffCoIsoRel}_{\mathbb{Z}_p}$ *has as objects* $\mathbb{N}$*, and as morphisms, relations* $R : \mathbb{Z}_p^{2m} \to \mathbb{Z}_p^{2n}$ *such that* R *viewed as a subset of* $\mathbb{Z}_p^m \times \mathbb{Z}_p^n$ *is an affine co-isotropic subspace thereof.*

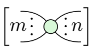
Since [26] works in the scalarless ZX-calculus, we need to add one extra axiom, which suffices to eliminate all remaining (non-zero) scalars in Stab_p: we impose the rule (Mod) that $p = 1$. Diagrammatically, this amounts to quotienting $(\mathsf{ZX}_p^{\mathrm{Stab}})^{\doteqdot}$ by the rule:  $=$.

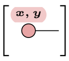
Then we can give an interpretation $[-]$ of $(\mathsf{ZX}_p^{\mathrm{Stab}})^{\doteqdot}$ making it universal and complete for $\mathsf{AffCoIsoRel}_{\mathbb{Z}_p}$, and which is defined uniquely by the commutative diagram:

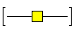
$$\begin{tikzcd}
(\mathsf{ZX}_p^{\mathrm{Stab}})^{\doteqdot} \arrow[r, dashed, "{[-]}"] \arrow[d, "{[\![-]\!]}"'] & \mathsf{AffCoIsoRel}_{\mathbb{Z}_p} \arrow[d, leftrightarrow] \\
\mathrm{CPM}(\mathsf{Stab}_p) \arrow[r, two heads] & \mathrm{CPM}(\mathsf{Stab}_p)/(\textsc{Mod})
\end{tikzcd}$$

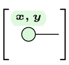
Explicitly, it is given by the identity on objects, $[m] = m$, and is defined on morphisms by: for $x, y \in \mathbb{Z}_p$ and $z \in \mathbb{Z}_p^*$,

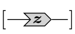

$$= \left\{\left(\bigoplus_{k=1}^m \begin{bmatrix}a\\b_k\end{bmatrix}, \bigoplus_{k=1}^n \begin{bmatrix}-a\\-c_k\end{bmatrix}\right) \,\middle|\, a, b_k, c_k \in \mathbb{Z}_p \quad \text{and} \quad \sum_k b_k = \sum_k c_k\right\}$$

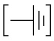

$$= \left\{\left(\bigoplus_{k=1}^m \begin{bmatrix}a_k\\c\end{bmatrix}, \bigoplus_{k=1}^n \begin{bmatrix}b_k\\c\end{bmatrix}\right) \,\middle|\, a, b_k, c_k \in \mathbb{Z}_p \quad \text{and} \quad \sum_k a_k = \sum_k b_k\right\}$$


$$= \left\{\left(\bullet, \begin{bmatrix}-1 & 0\\-y & -1\end{bmatrix}\begin{bmatrix}v\\0\end{bmatrix} + \begin{bmatrix}-x\\0\end{bmatrix}\right) \,\middle|\, v \in \mathbb{Z}_p\right\}$$


$$= \left\{\left(\mathbf{v}, \begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}\mathbf{v}\right) \,\middle|\, \mathbf{v} \in \mathbb{Z}_p^2\right\}$$


$$= \left\{\left(\bullet, \begin{bmatrix}1 & -y\\0 & 1\end{bmatrix}\begin{bmatrix}0\\v\end{bmatrix} + \begin{bmatrix}0\\x\end{bmatrix}\right) \,\middle|\, v \in \mathbb{Z}_p\right\}$$


$$= \left\{\left(\mathbf{v}, \begin{bmatrix}z & 0\\0 & z^{-1}\end{bmatrix}\mathbf{v}\right) \,\middle|\, \mathbf{v} \in \mathbb{Z}_p^2\right\}$$

Note that all of these are actually affine *Lagrangian* relations. The only generator which has a co-isotropic but not Lagrangian semantics is the discard map:  $= \{(\mathbf{v}, \bullet) \mid \mathbf{v} \in \mathbb{Z}_p^2\}$.

As pointed out in [7, 8, 26], the related category of affine Lagrangian relations over the field $\mathbb{R}[x,y]/(xy-1)$ can be used to represent a fragment of electrical circuits. We expect that the axiomatisation of figure 1 can be adapted to that setting, but leave this for a future article.

6 Conclusion

We have constructed a ZX-calculus which captures the stabiliser fragment in odd prime dimensions, whilst retaining many of the “nice” features of the qubit ZX-calculus. Of course, there are a few obvious questions that we leave for future work.

First amongst these is the question of whether a fully universal calculus can be obtained from the ideas we used here. The spiders we have used here labelled with elements $a, b \in \mathbb{Z}_p \times \mathbb{Z}_p$ and which can be interpreted as polynomials $x \mapsto ax + bx^2$ which parametrise the phases of the spider. Adding one additional term of degree 3 is already sufficient to obtain a universal calculus in prime dimensions (strictly) greater than 3 [37]. In fact, one might as well add all higher order of polynomials (mod p) since access to such higher degrees will hopefully prove useful in finding commutation relations for the resulting spiders.

Secondly, it remains to be seen how to formulate a universal ZX-calculus for non-prime dimensions, even for just the stabiliser fragment. For this, the methods in this article are clearly inadequate: for example local scaling is no longer an invertible operation and thus certainly not in the Clifford group.

Finally, the set of axioms we provide here is probably not minimal. It would be nice to see if a simplified version can be obtained, as was done in [6] for the qubit case.

References

1 Samson Abramsky and Bob Coecke. Categorical quantum mechanics, 2008. arXiv:0808.1023.

2 D. M. Appleby. Properties of the extended Clifford group with applications to SIC-POVMs and MUBs, 2009. arXiv:0909.5233.

3 Miriam Backens. The ZX-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, 2014. doi:10.1088/1367-2630/16/9/093021.

4 Miriam Backens. Making the stabilizer ZX-calculus complete for scalars. *Electronic Proceedings in Theoretical Computer Science*, 195:17–32, 2015. doi:10.4204/EPTCS.195.2.

5 Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski, and John van de Wetering. There and back again: A circuit extraction tale. *Quantum*, 5:421, 2021. doi:10.22331/q-2021-03-25-421.

6 Miriam Backens, Simon Perdrix, and Quanlong Wang. A Simplified Stabilizer ZX-calculus. *Electronic Proceedings in Theoretical Computer Science*, 236:1–20, 2017. doi:10.4204/EPTCS.236.1.

7 John C. Baez, Brandon Coya, and Franciscus Rebro. Props in Network Theory. *Theory and Application of Categories*, 33(25):727–783, 2018. arXiv:1707.08321.

8 John C. Baez and Brendan Fong. A Compositional Framework for Passive Linear Networks. *Theory and Application of Categories*, 33(38):1158–1222, 2018. arXiv:1504.05625.

9 Xiaoning Bian and Quanlong Wang. Graphical Calculus for Qutrit Systems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E98.A(1):391–399, 2015. doi:10.1587/transfun.E98.A.391.

10 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String Diagram Rewrite Theory II: Rewriting with Symmetric Monoidal Structure, 2021. doi:10.48550/arXiv.2104.14686.

11 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobocinski, and Fabio Zanasi. String Diagram Rewrite Theory I: Rewriting with Frobenius Structure. *Journal of the ACM (JACM)*, 2022. doi:10.1145/3502719.

12 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. String Diagram Rewrite Theory III: Confluence with and without Frobenius, 2022. doi:10.48550/arXiv.2109.06049.

13 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Interacting Hopf Algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, 2017. doi:10.1016/j.jpaa.2016.06.002.

14 Robert I. Booth and Titouan Carette. Complete ZX-calculi for the stabiliser fragment in odd prime dimensions, 2022. arXiv:2204.12531.

15 Titouan Carette. When Only Topology Matters, 2021. arXiv:2102.03178.

16 Titouan Carette. *Wielding the ZX-calculus, Flexsymmetry, Mixed States, and Scalable Notations.* Theses, Université de Lorraine, 2021. URL: https://hal.archives-ouvertes.fr/tel-03468027.

17 Titouan Carette, Dominic Horsman, and Simon Perdrix. SZX-calculus: Scalable Graphical Quantum Reasoning, 2019. arXiv:1905.00041.

18 Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of graphical languages for mixed states quantum mechanics. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 108:1–108:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.108.

19 Titouan Carette and Simon Perdrix. Colored props for large scale graphical reasoning, 2020. arXiv:2007.03564.

20 Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren, and Dominic Horsman. Graphical Structures for Design and Verification of Quantum Error Correction, 2018. arXiv:1611.08012.

21 Sean Clark. Valence bond solid formalism for d-level one-way quantum computation. *Journal of Physics A: Mathematical and General*, 39(11):2701–2721, 2006. doi:10.1088/0305-4470/39/11/010.

22 Bob Coecke and Ross Duncan. Interacting Quantum Observables: Categorical Algebra and Diagrammatics. *New Journal of Physics*, 13(4):043016, 2011. doi:10.1088/1367-2630/13/4/043016.

23 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.* Cambridge University Press, 2017. doi:10.1017/9781316219317.

24 Cole Comfort. Relational semantics for quantum algorithms.

25 Cole Comfort. A symplectic setting for mixed stabiliser circuits, 2021.

26 Cole Comfort and Aleks Kissinger. A Graphical Calculus for Lagrangian Relations, 2021. arXiv:2105.06244.

27 Niel de Beaudrap. A linearized stabilizer formalism for systems of finite dimension, 2012. arXiv:1102.3354.

28 Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. Fast and effective techniques for T-count reduction via spider nest identities, 2020. arXiv:2004.05164.

29 Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery, 2017. arXiv:1704.08670.

30 Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*, 4:279, 2020. doi:10.22331/q-2020-06-04-279.

31 Ross Duncan and Maxime Lucas. Verifying the Steane code with Quantomatic. *Electronic Proceedings in Theoretical Computer Science*, 171:33–49, 2014. doi:10.4204/EPTCS.171.4.

32 Ross Duncan and Simon Perdrix. Rewriting Measurement-Based Quantum Computations with Generalised Flow. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming*, volume 6199, pages 285–296. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-14162-1_24.

33 Liam Garvie and Ross Duncan. Verifying the Smallest Interesting Colour Code with Quantomatic. *Electronic Proceedings in Theoretical Computer Science*, 266:147–163, 2018. doi:10.4204/EPTCS.266.10.

34 Vlad Gheorghiu. Standard form of qudit stabilizer groups. *Physics Letters A*, page 5, 2014.

35 Daniel Gottesman. Fault-Tolerant Quantum Computation with Higher-Dimensional Systems. *Chaos, Solitons & Fractals*, 10(10):1749–1758, 1999. doi:10.1016/S0960-0779(98)00218-5.

36 Ladina Hausmann, Nuriya Nurgalieva, and Lídia del Rio. A consolidating review of Spekkens' toy theory. arXiv:2105.03277.

37 Mark Howard and Jiri Vala. Qudit versions of the qubit "pi-over-eight" gate. *Physical Review A*, 86(2):022316, 2012. doi:10.1103/PhysRevA.86.022316.

38 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A Complete Axiomatisation of the ZX-Calculus for Clifford+T Quantum Mechanics, 2017. arXiv:1705.11151.

39 Aleks Kissinger and John van de Wetering. Reducing T-count with the ZX-calculus, 2020. arXiv:1903.10477.

40 Aleks Kissinger, John van de Wetering, and Renaud Vilmart. Classical simulation of quantum circuits with partial and graphical stabiliser decompositions, 2022. arXiv:2202.09202.

41 Markus Nenhauser. An Explicit Construction of the Metaplectic Representation over a Finite Field. *Journal of Lie Theory*, 12(15), 2002.

42 Kang Feng Ng and Quanlong Wang. A universal completion of the ZX-calculus, 2017. arXiv:1706.09877.

43 André Ranchin. Depicting qudit quantum mechanics and mutually unbiased qudit theories. *Electronic Proceedings in Theoretical Computer Science*, 172:68–91, 2014. doi:10.4204/EPTCS.172.6.

44 Peter Selinger. Dagger compact closed categories and completely positive maps: (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 170:139–163, 2007. doi:10.1016/j.entcs.2006.12.018.

45 Alex Townsend-Teague and Konstantinos Meichanetzidis. Classifying Complexity with the ZX-Calculus: Jones Polynomials and Potts Partition Functions, 2021. arXiv:2103.06914.

46 John van de Wetering. ZX-calculus for the working quantum computer scientist, 2020. arXiv:2012.13966.

47 Renaud Vilmart. A Near-Optimal Axiomatisation of ZX-Calculus for Pure Qubit Quantum Mechanics, 2018. arXiv:1812.09114.

48 Quanlong Wang. Qutrit ZX-calculus is Complete for Stabilizer Quantum Mechanics. *Electronic Proceedings in Theoretical Computer Science*, 266:58–70, 2018. doi:10.4204/EPTCS.266.3.

49 Quanlong Wang. A non-anyonic qudit ZW-calculus, 2021. arXiv:2109.11285.

50 Quanlong Wang. Qufinite ZX-calculus: A unified framework of qudit ZX-calculi, 2021. arXiv:2104.06429.

51 Quanlong Wang and Xiaoning Bian. Qutrit Dichromatic Calculus and Its Universality. *Electronic Proceedings in Theoretical Computer Science*, 172:92–101, 2014. doi:10.4204/EPTCS.172.7.

52 Yuchen Wang, Zixuan Hu, Barry C. Sanders, and Sabre Kais. Qudits and High-Dimensional Quantum Computing. *Frontiers in Physics*, 8, 2020. doi:10.3389/fphy.2020.589504.

53 Fabio Zanasi. *Interacting Hopf Algebras: The Theory of Linear Systems.* phdthesis, Ecole Normale Superieure de Lyon, 2018. arXiv:1805.03032.

54 D. L. Zhou, B. Zeng, Z. Xu, and C. P. Sun. Quantum computation based on d-level cluster states. *Physical Review A*, 68(6):062303, 2003. doi:10.1103/PhysRevA.68.062303.

Extending Partial Representations of Circle Graphs in Near-Linear Time

Guido Brückner
Karlsruhe Institute of Technology, Germany

Ignaz Rutter
University of Passau, Germany

Peter Stumpf
University of Passau, Germany

Abstract

The *partial representation extension problem* generalizes the recognition problem for geometric intersection graphs. The input consists of a graph G, a subgraph $H \subseteq G$ and a representation $\mathcal{H}$ of H. The question is whether G admits a representation $\mathcal{G}$ whose restriction to H is $\mathcal{H}$. We study this question for *circle graphs*, which are intersection graphs of chords of a circle. Their representations are called *chord diagrams*.

We show that for a graph with n vertices and m edges the partial representation extension problem can be solved in $O((n+m)\alpha(n+m))$ time, where α is the inverse Ackermann function. This improves over an $O(n^3)$-time algorithm by Chaplick, Fulek and Klavík [2019]. The main technical contributions are a canonical way of orienting chord diagrams and a novel compact representation of the set of all canonically oriented chord diagrams that represent a given circle graph G, which is of independent interest.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases circle graphs, partial representation extension, split decomposition tree, recognition algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.25

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Ru 1903/3-1.

1 Introduction

Geometric intersection representations of graphs are an important concept that establishes a strong connection between geometry, combinatorics and graph theory. In an intersection representation of a graph $G = (V, E)$ each vertex $v \in V$ is represented by a geometric object $R(v)$ whose intersections encode the edges of G, i.e., $\{u, v\} \in E$ if and only if $R(u)$ and $R(v)$ intersect. Different classes of graphs can be obtained by restricting the types of geometric objects used for the representation. For example *interval graphs* are intersection representations of intervals of the real line, *string graphs* are intersection graphs of curves in the plane and *circle graphs* are intersection graphs of chords of a circle; see Figure 1.

For a fixed class $\mathcal{C}$ of intersection graphs a natural question is the *recognition problem*, which asks whether a given graph G belongs to $\mathcal{C}$. For circle graphs the recognition problem has been studied for a long time, and has culminated in an algorithm with running time $O((n + m)\alpha(n + m))$ [10, 13], where n and m denote the number of vertices and edges of the input graph, respectively, and α denotes the slowly growing inverse of the Ackermann function. There is also an $O(n^2)$ time algorithm which is faster for very dense graphs [24].

A generalization of the basic recognition problem has attracted considerable attention: the *partial representation extension problem* [19, 1, 23, 2, 9, 18, 17, 21]. In this problem, the input consists of a triplet $(G, H, \mathcal{R}')$, where G is a graph, $H \subseteq G$ is an induced subgraph

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 25; pp. 25:1–25:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

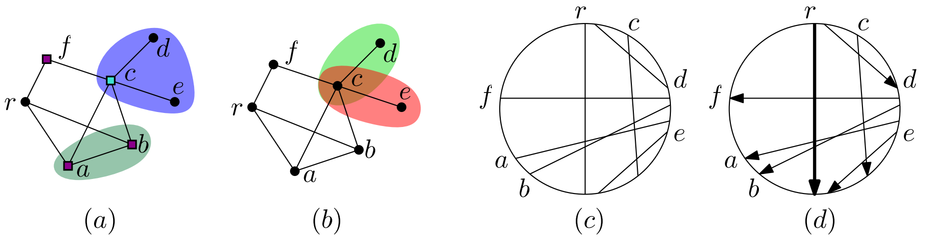


Figure 1 A graph G with good (a) and overlapping (b) splits. Only one side of each split is marked. The square vertices in (a) are the boundary of the blue split. (c) an undirected chord diagram of G (d) an oriented chord diagram of G with reference chord r.

of G, and $\mathcal{R}'$ is an intersection representation of H. The question is whether there exists a representation $\mathcal{R}$ of G that *extends* $\mathcal{R}'$, i.e., its restriction to H coincides with $\mathcal{R}'$. Following the notation of Chaplick, Fulek and Klavík [4], for a class $\mathcal{C}$ of intersection graphs, we denote the partial representation extension problem for $\mathcal{C}$ by RepExt($\mathcal{C}$).

Related Work

The recognition problem can be solved efficiently for a wide range of classes of intersection graphs. The partial representation extension problem was introduced by Klavík et al. [20], who gave an efficient algorithm for RepExt($\mathcal{I}$), where $\mathcal{I}$ denotes the class of interval graphs, which they improved to linear running time in their full version [19]. Angelini et al. [1] give a linear-time algorithm for planar topological graph drawings and Patrignani shows NP-hardness for planar straight-line drawings [23]. Recently, the problem has also been considered for simple topological and 1-planar drawings [2, 9]. In the meantime efficient algorithms are known for proper and unit interval graphs [18], permutation graphs and function graphs [17], as well as for trapezoid graphs [21]. Concerning the class Circle of circle graphs, Chaplick et al. [4] gave the first efficient algorithm for RepExt(Circle) with running time $O(n^3)$.

For other forms of representation, there are compact descriptions of all representations of a graph G, e.g., SPQR-trees [22, 8] for planar graphs and modular decomposition trees [11] for comparability graphs. Both descriptions express a representation of G by choosing representations for small graphs that are associated to the nodes of a tree in such a way that a bijection is obtained between the representations of G and the choices for the small graphs. For circle graphs, Cunningham and Edmonds introduced split-trees [7] that decompose a graph along its splits into smaller graphs from which G is assembled in a tree structure. Gioan and Paul [12] described split-trees using graph-labeled trees. Gioan et al. [13] observed that all possible chord diagrams of G can be obtained by choosing a chord diagram for each of the small graphs associated to the nodes of the split-tree of G and combining them suitably with each other.

Contribution and Outline

However, besides the choices of the chord diagrams, there are choices to be made when combining those diagrams, which Gioan et al. [13] express by basically directing the individual chords. These choices allow to obtain the same chord diagram starting with a different set of directed chord diagrams for the nodes of the split-tree; see Figure 2. We strengthen this connection by generalizing the concept of split-trees and introducing a canonical orientation of chord diagrams. In particular, for the canonical split-trees of Cunningham and Edmonds,

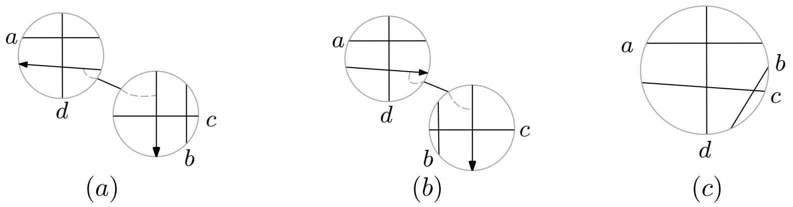


Figure 2 (a), (b) split-trees with different associated chord diagrams, only the relevant orientations are shown (c) the chord diagram resulting from the join for (a) and (b).

we establish a situation analogous to SPQR-trees or modular decomposition trees: the canonically oriented chord diagrams of G correspond bijectively to choices of canonically oriented chord diagrams for the nodes of the canonical split-tree.

In the $O(n^3)$-time algorithm of Chaplick et al. [4] for RepExt(Circle), they characterize all viable chord diagrams and solve the problem recursively, creating a decomposition that appears to be similar to the decomposition in our representation. To illustrate the usefulness of our representation, we improve over this result by showing how to extend partial chord diagrams in near-linear time $O((n+m)\alpha(n+m))$. This answers open questions of Chaplick et al. [4] and Kalisz et al. [16] who specifically ask whether such a representation of all chord diagrams exists and whether it can be used to solve RepExt(Circle) faster. We note that, given the data structure computed by the fastest known recognition algorithm by Gioan et al. [13], our algorithm runs in linear time.

We introduce notation and preliminary definitions in Section 2. In Section 3 we develop the compact description of representations. Section 4 shows how these results can be used to obtain an almost-linear time algorithm for RepExt(Circle). Lemmas and theorems marked with ($\star$) are proven in the full version.

2 Preliminaries

In this section we introduce important concepts that we use throughout the paper. In particular, we recall the concepts of circle graphs and chord diagrams, and we introduce a way to canonically orient them. Moreover, we also recall the notion of splits and split decompositions, which are a classic tool in circle graph recognition algorithms.

Circle Graphs and Chord Diagrams

An *(undirected) chord diagram* D consists of a set C of *chords* of the unit circle, i.e., undirected straight-line segments that connect pairwise distinct points on the unit circle. A chord diagram D naturally defines an intersection graph $G(D) = (C, E)$ of its chords, where $\{c, c'\} \in E$ if and only if the chords c and c' intersect in D, i.e., if and only if their endpoints alternate around the unit circle. A graph G is a *circle graph* if it is an intersection graph of the chords of a chord diagram.

While chord diagrams are geometric objects, we are mostly interested in their combinatorial structure, i.e., we identify chord diagrams with the same order of chord ends on the circle. To break certain symmetries we usually consider *oriented* chord diagrams, which additionally have a chord end as a *starting point*. Then an oriented chord diagram D can be encoded as the word over the set of chords C obtained by starting at the starting point and walking around the unit circle in clockwise direction and recording the encountered chords. For $c \in C$, we refer to the first end of c that we encounter with $\mathring{c}$ and to the second end with

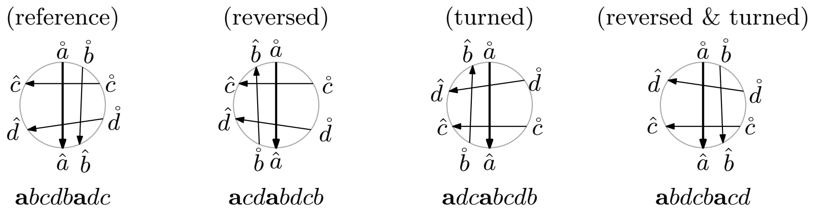


Figure 3 The turn and rev operation on a chord diagram with reference chord a.

$\hat{c}$. We consider the chord c as directed from $\mathring{c}$ to $\hat{c}$ in the geometric interpretation; see Figure 3 (reference). Unless mentioned otherwise, chord diagrams are always considered oriented and they are usually treated as their encodings.

We call the chord with the starting point the *reference chord* of D. We never change the reference chord. In the following, we consider changes that can be applied to every chord diagram without changing the represented graph or the reference chord. For a word w we write w^{rev} for the word obtained by reading w backwards. The *reverse* of a chord diagram $a\rho a\sigma$ is $\text{rev}(a\rho a\sigma) = a\sigma^{\text{rev}}a\rho^{\text{rev}}$. Geometrically, this corresponds to mirroring the chord diagram at the reference chord a; see Figure 3 (reversed). The *turn* of a chord diagram $a\rho a\sigma$ is $\text{turn}(a\rho a\sigma) = a\sigma a\rho$. Geometrically, this corresponds to choosing $\hat{a}$ as the new starting point. In a sense, this turns the chord diagram by 180 degrees; see Figure 3 (turned). Note that $\text{turn}(\text{turn}(a\rho a\sigma)) = a\rho a\sigma$, $\text{rev}(\text{rev}(a\rho a\sigma)) = a\rho a\sigma$ and $\text{turn}(\text{rev}(a\rho a\sigma)) = \text{turn}(a\sigma^{\text{rev}}a\rho^{\text{rev}})) = a\rho^{\text{rev}}a\sigma^{\text{rev}} = \text{rev}(a\sigma a\rho) = \text{rev}(\text{turn}(a\rho a\sigma))$, i.e., turn and rev are selfinverse and commute.

Splits and Split-Trees

Let $G = (V, E)$ be a graph. Consider a bipartition (X, Y) of V with $\emptyset \subsetneq X, Y \subsetneq V$ and let $B_X \subseteq X$ denote the subset of vertices of X that are adjacent to a vertex in Y and let $B_Y \subseteq Y$ denote the subset of vertices of Y that are adjacent to a vertex in X. The bipartition (X, Y) is called a *split* if all possible edges between B_X and B_Y exist in G, i.e., $\{\{x, y\} \mid x \in B_X, y \in B_Y\} \subseteq E$. Then (B_X, B_Y) is called the *split boundary*; see Figure 1. Observe that if X and Y are not connected, then B_X, B_Y are empty. A split (X, Y) is *trivial* if one of its sides consists of a single vertex. Following Courcelle [5], we call a split (X, Y) *good* if it does not overlap any other split, in the sense that for any other split (W, Z) at least one of $X \cap W$, $X \cap Z$, $Y \cap W$, $Y \cap Z$ is empty. A graph is *prime* if all its splits are trivial, and it is *degenerate* if every bipartition of the vertices yields a split. It is well known that the connected degenerate graphs are precisely cliques and stars [6].

Next, we define *split-trees*, introduced by Cunningham and Edmonds [7] as decomposition trees. We use the graph-labeled tree description of Gioan and Paul [12]. However we consider graph-labeled trees that correspond to general decompositions in the sense of Cunningham and Edmonds, while Gioan and Paul used the term split-tree for a unique structure which we later define as the canonical split-tree. To avoid confusion with the vertices and edges of our graphs, we refer to the vertices and edges of split-trees as nodes and arcs, respectively. A *split-tree* T is a tree where each inner node μ has a *skeleton graph* $\text{skel}(\mu)$ and a bijection corr_μ from $V(\text{skel}(\mu))$ to the nodes of T adjacent to μ; see Figure 4. Given an inner node μ and a neighbor ν of μ, we often need to refer to the vertex v of $\text{skel}(\mu)$ that represents ν. For convenience, we define $v_\mu(\nu)$ as the vertex v of $\text{skel}(\mu)$ with $\text{corr}_\mu(v) = \nu$.

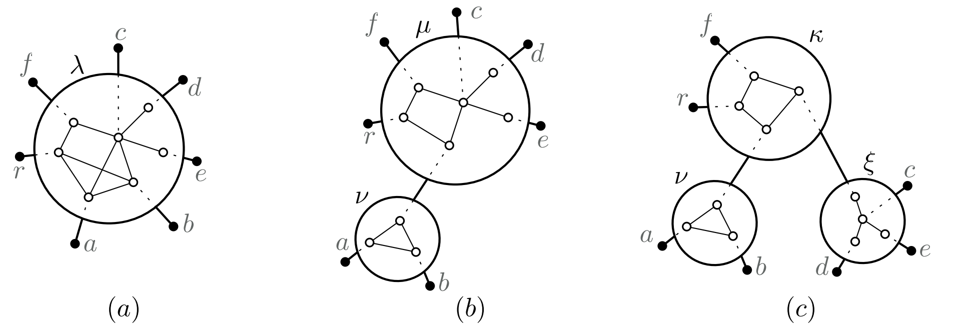


Figure 4 Split-trees of the graph from Figure 1a. (a) shows the base case, in (b) the node λ from (a) is decomposed into μ and ν along the split $(\{a,b\},\{c,d,e,r,f\})$. In (c) the node μ from (b) is further decomposed along the split that has $\{c,d,e\}$ on one side.

We define split-trees inductively. Let $G=(V,E)$ be a graph. In the base case; see Figure 4a, a split-tree T of G consists of one inner node λ and one leaf for each vertex $v \in V$ that is adjacent to λ. We identify the leaves of T with the vertices in V. Define the skeleton of λ as $\text{skel}(\lambda)=G$. For every vertex $v \in V(\text{skel}(\lambda))$, we define $\text{corr}_\lambda(v)$ as the leaf node v of T.

We can further decompose such a split-tree. Let λ be an inner node and let (X,Y) be a non-trivial split of $\text{skel}(\lambda)$. Let G_X denote the graph obtained from $\text{skel}(\lambda)$ by contracting Y into a single vertex y. Symmetrically, let G_Y denote the graph obtained from $\text{skel}(\lambda)$ by contracting X into a single vertex x. Then λ can be split into two nodes μ, ν connected by an arc $\{\mu,\nu\}$; see Figure 4b. Define $\text{skel}(\mu)=G_X$ and $\text{skel}(\nu)=G_Y$. Observe that y in G_X represents the graph G_Y and x in G_Y represents the graph G_X. We define $\text{corr}_\mu(y)=\nu$ and $\text{corr}_\nu(x)=\mu$. For any vertex $v \in V(\text{skel}(\lambda))$ we replace the arc $\{\lambda,\text{corr}_\lambda(v)\}$ with $\{\mu,\text{corr}_\lambda(v)\}$ if $v \in X$ or $\{\nu,\text{corr}_\lambda(v)\}$ if $v \in Y$. Finally, for each inner node κ adjacent to λ consider the unique vertex $v \in \text{skel}(\kappa)$ with $\text{corr}_\kappa(v)=\lambda$. We redefine $\text{corr}_\kappa(v)=\mu$ if $v_\lambda(\kappa) \in X$ and $\text{corr}_\kappa(v)=\nu$ if $v_\lambda(\kappa) \in Y$. Observe that the result is still a tree and corr_ξ is well defined for each inner node ξ. Hence, the inner nodes may be decomposed again. The split-trees of G are the split-trees that can be obtained in this way.

The inverse of a decomposition is a *join*. Let G_1, G_2 be two (vertex-disjoint) graphs with $v_2 \in G_1$, $v_1 \in G_2$. Then we define their *join at* v_2,v_1 as the graph obtained from $G_1 \cup G_2$ by connecting all neighbors of v_2 in G_1 with all neighbors of v_1 in G_2 and removing the vertices v_1, v_2. We denote the resulting graph by $G_1 \oplus_{v_2,v_1} G_2=(V,E)$. For a split-tree T with an arc $\{\mu,\nu\}$ let $b=v_\mu(\nu)$, $a=v_\nu(\mu)$. The nodes μ, ν can be joined into a single node λ with $\text{skel}(\lambda)=\text{skel}(\mu) \oplus_{b,a} \text{skel}(\nu)$. Moreover, for any inner node κ adjacent to μ or ν and the unique vertex $v \in V(\text{skel}(\kappa))$ with $\text{corr}_\kappa(v) \in \{\mu,\nu\}$ we redefine $\text{corr}_\kappa(v)=\lambda$.

Let T denote a split-tree and let $\{\mu,\nu\}$ be an arc of T. Removing $\{\mu,\nu\}$ separates T into two trees T_μ and T_ν where T_μ contains the node μ and T_ν contains the node ν. Let $L(T_\mu) \subseteq V$ denote the set of leaves of T_μ. By construction of the split-tree $(L(T_\mu),L(T_\nu))$ is a split of G, which is induced by the arc $\{\mu,\nu\}$. For example, the blue split from Figure 1a is represented by the arc $\{\kappa,\xi\}$ in Figure 4c. Let (A,B) be a non-trivial split of $\text{skel}(\mu)$. This split induces a split (L_A,L_B) of G with $L_A=\bigcup_{v\in A} L(T_{\text{corr}_\mu(v)})$ and $L_B=\bigcup_{v\in B} L(T_{\text{corr}_\mu(v)})$. An example of this is the red split from Figure 1b that is represented by a non-trivial split inside node ξ of Figure 4c. We call an inner node of a split-tree *degenerate* and *prime*, if its skeleton graph is degenerate and prime, respectively. Observe that in the split-tree in Figure 4c, the node κ is prime, and ν, ξ are degenerate.

In a *good-split-tree* every inner arc $\{\mu, \nu\}$ of T induces a good split. For a connected graph G, a *canonical split-tree* is a good-split-tree such that no skeleton has a non-trivial good split, no inner arc induces a trivial split, and any two arcs induce different splits. A canonical split-tree is obtained by decomposing G (in arbitrary order) along all its good splits. This is possible since the good splits form a laminar set. The resulting canonical split-tree is denoted by $\mathrm{ST}(G)$. It was first defined by Cunningham and Edmonds [7]. We use the graph-labeled-tree description by Gioan and Paul [12]. It is unique, and nodes have connected skeletons that are either prime or degenerate. Moreover, if two adjacent nodes μ, ν are both degenerate, then either one of them is a star and one is a clique, or they are both stars and the vertices $v_\mu(\nu)$ and $v_\nu(\mu)$ are either both leaves or both the star centers of their respective skeletons. By the following useful property, $\mathrm{ST}(G)$ represents all splits of G.

▶ **Proposition 1** ([6],[14, Theorem 2.18])**.** *A partition (A, B) of the vertex set of a connected graph G is a split of G if and only if it is induced by an arc of $\mathrm{ST}(G)$ or by a non-trivial split of a (skeleton of a) degenerate node of $\mathrm{ST}(G)$.*

3 Compact Representation of all Chord Diagrams

Let $G = (V, E)$ be a circle graph. In Section 3.1 we establish a correspondence between the chord diagrams of G and assignments of chord diagrams to the inner nodes of a split-tree of G. For canonical split-trees, this correspondence turns out to be a bijection. We further show that both directions of this bijection can be computed in linear time. In Section 3.2 we describe the possible choices for the chord diagrams for nodes of a canonical split-tree of G. Altogether, this gives the claimed compact representation for connected circle graphs.

3.1 Configurations of Split-Trees

Let $G = (V, E)$ be a circle graph, let $r \in V$ be the reference chord of G and let T be a split-tree of G. We root T at r and direct all arcs away from the root. For each inner node ν of T, we define the reference chord r_ν as the chord $v_\nu(\mu)$ associated with the parent μ of ν. A *configuration* c of T is a mapping that assigns to each inner node μ of T a chord diagram $c(\mu)$ of $\mathrm{skel}(\mu)$ with reference chord r_μ; see Figure 5. We also refer to $c(\mu)$ as the *configuration of* μ. Recall that a split decomposition can be used to split a node λ of T into two nodes μ, ν connected by a (directed) arc (μ, ν). In the reverse direction, a join composition can be used to join two nodes μ, ν connected by an arc (μ, ν) into a single node λ. The join operation extends to configurations.

The configurations of μ and ν induce a unique configuration of λ as follows. Let $v = v_\mu(\nu)$ and let $c(\nu) = r_\nu \rho r_\nu \sigma$. The induced configuration $c(\lambda) = c(\mu) \oplus_{v, r_\nu} c(\nu)$ of λ is obtained from $c(\mu)$ by replacing $\mathring{v}$ with ρ and $\hat{v}$ with σ. See Figure 5b. The order of multiple joins affects neither the resulting split-tree nor the resulting configuration. For a configuration c of a split-tree T we denote by $\mathcal{D}(c)$ the chord diagram obtained by joining the configurations of all inner nodes of T. For connected graphs, the two joined chord diagrams $c(\mu)$, $c(\nu)$ are fully determined by their join $c'(\lambda)$ since the four combined words ρ, σ, μ, ν are fully determined by $c'(\lambda)$. This implies that two different configurations yield two different chord diagrams.

If G is connected, a diagram D can be decomposed along a split (A, B) only if the endpoints of the chords in A and B appear suitably in D. We say that D *respects* the split (A, B) if D decomposes into two words over A and two words over B that alternate. Formally, this means we have $D = \rho_1 \sigma_1 \rho_2 \sigma_2 \rho_3$ where $\rho_2, (\rho_3 \rho_1)$ and σ_1, σ_2 are words over A and B.

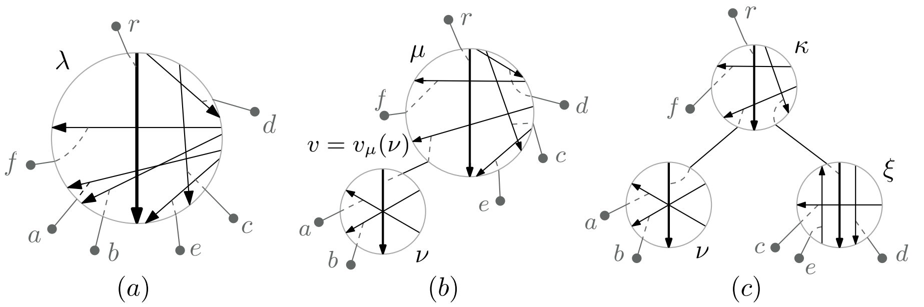


Figure 5 (a), (b), (c) show configurations of the corresponding split-trees in Figure 4 that yield the same chord diagram. We have $c'(\lambda) = rdcdf\mathbf{ba}ecer\mathbf{ba}f = rdcdf\mathbf{v}ecer\mathbf{v}f \oplus_{v,r_\nu} r_\nu bar_\nu ba = c(\mu) \oplus_{v,r_\nu} c(\nu)$, where $v = v_\mu(\nu)$.

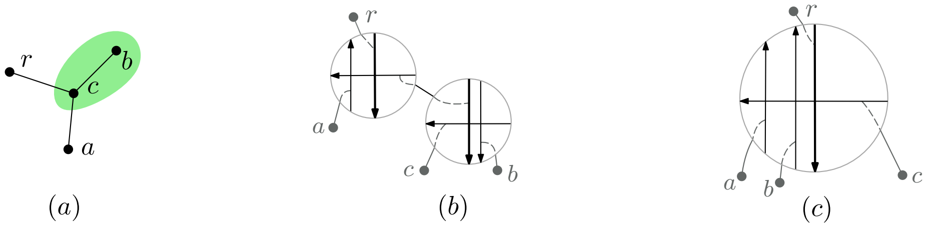


Figure 6 (a) A star G with a non-good split ({r,a},{b,c}). (b) A split-tree T of G with an arc corresponding to the bad split in (a). (c) A chord diagram of G that is not represented by T.

We show that a chord diagram can be recursively decomposed along several splits if it respects each of them. This yields the following lemma.

▶ **Lemma 2** ($\star$). *Let G be a (not necessarily connected) circle graph and let T be a split-tree of G rooted at some reference chord r. Then the mapping $\mathcal{D}$ that maps configurations of T to chord diagrams of G is surjective on the set of chord diagrams of G with reference chord r that respect all splits induced by arcs of T.*

Note that, for any configuration c of T, the chord diagram $\mathcal{D}(c)$ respects all splits induced by arcs of T. Since it is known that a chord diagram D of a connected circle graph G respects all good splits of G [5, Proposition 9], we conclude the following.

▶ **Theorem 3.** *Let T be a good-split-tree of a connected circle graph G rooted at some reference chord $r \in V(G)$. Then the mapping $\mathcal{D}$ that maps configurations of T to chord diagrams of G with reference chord r is a bijection.*

We can translate between configuration and chord diagram in linear time, which allows us to use this result algorithmically. Note that in the following theorem the split-trees are not required to be good-split-trees, which means there may be chord diagrams for their graph that they do not represent; see Figure 6. This will be useful when dealing with RepExt(Circle).

▶ **Theorem 4.** *Let T be a split-tree of a connected circle graph G rooted at some reference chord $r \in V(G)$. Then the mapping $\mathcal{D}$ can be computed in linear time. Conversely, given a chord diagram D of G with reference chord r, it can be tested in linear time whether there exists a configuration c of T with $\mathcal{D}(c) = D$. If it exists, the configuration c can also be computed in linear time.*

Proof. We store chord diagrams as circular doubly linked lists of endpoints of chords in clockwise order. We assume that each chord endpoint is equipped with a pointer to the corresponding vertex and each vertex has pointers to the two endpoints of its chord.

Let c be a configuration of T. We first store for each chord u in a chord diagram which of its ends is $\mathring{u}$. We then process the tree T in bottom-up order. If c contains only a single inner node, then the configuration of this node is the desired diagram $\mathcal{D}(c)$. Otherwise let μ be an inner node of T whose children are all leaves and let ν be its parent. Let T' be the split tree obtained by replacing in T the node μ together with all its leaves by a single leaf v and let c' be the configuration of T' that coincides with c for all nodes of T'. Clearly, c' and T' can be computed from c and T in $O(1)$ time. We now recursively compute $\mathcal{D}(c')$ in time linear in the size of T'. To obtain $\mathcal{D}(c)$, we replace in $\mathcal{D}(c')$ the endpoints $\mathring{v}, \hat{v}$ of v by the sequences π, σ, respectively, where $c(\mu) = r\pi r\sigma$ and r is the reference chord of μ. Since we maintain the order of the endpoints in doubly linked lists, this can be done in $O(1)$ time. Therefore we only spend $O(1)$ time per node of T.

Conversely assume that D is a chord diagram of G with reference chord r. We again process the tree T in bottom-up order. As before, if T contains only a single inner node μ then $c(\mu) = D$ is the desired configuration. Otherwise let μ be an inner node whose children are all leaves. We denote the set of leaves of μ by L. Note that $(L, V \setminus L)$ is a split of G. Let T' be the tree obtained from T by replacing μ and its leaves by a single leaf v. Using simple flags, we can decide for an endpoint of a chord of D in constant time whether it belongs to a vertex of L and if so, whether it has already been processed. We now treat the endpoints of the vertices in L one by one. For the first endpoint e we obtain in this way, we scan to the left and right in the doubly linked list of D starting at e. In this way we determine a maximal sublist $[e_1, e_2]$ of D (containing all endpoints that lie clockwise between the first and last points e_1, e_2 of that list) with $e \in [e_1, e_2]$ such that all elements of $[e_1, e_2]$ belong to vertices of L. We mark all endpoints that we encounter in this way as processed. We then scan further the leaves of L. We do find another endpoint f that is not yet processed since G is connected. We similarly determine a sublist $[f_1, f_2]$ around f so that all endpoints that lie clockwise between f_1, f_2 belong to vertices of L and the predecessor of f_1 and the successor of f_2 do not. If there is an unprocessed endpoint left, this means D does not respect split $(L, V \setminus L)$ and we can reject. Hence, assume no unprocessed endpoint remains.

We thus have partitioned the endpoints of the chords of L into two disjoint sublists $[e_1, e_2]$ and $[f_1, f_2]$ of D. Then let D' be the diagram obtained from D by replacing the sublists $[e_1, e_2]$, $[f_1, f_2]$ each with v. We recursively compute a configuration c' of T' with $\mathcal{D}(c') = D'$ where for each chord u the end $\mathring{u}$ is stored. If this succeeds, we obtain the desired configuration c as follows. For each inner node $\nu \neq \mu$ we set $c(\nu) = c'(\nu)$. Since we know from each predecessor and successor u of $\mathring{v}$, $\hat{v}$ whether it is $\mathring{u}$, we can set $c(\mu) = r[e_1, e_2]r[f_1, f_2]$ or $c(\mu) = r[f_1, f_2]r[e_1, e_2]$ accordingly, where r is the reference chord of $\operatorname{skel}(\mu)$. By construction it is $\mathcal{D}(c) = D$. Note that any other choice of $c(\mu)$ or $\mathcal{D}(c_{|T-\mu})$ results in a chord diagram different from D, since $[e_1, e_2]$ and $[f_1, f_2]$ are non-empty and separated by chords not in L. We can then first iterate through the list of $[e_1, e_2]$, $[f_1, f_2]$ that replaces $\mathring{v}$ and then other one to store each endpoint $\mathring{u}$.

The time spent to compute T', D' as well to modify c' into c is proportional to $|L|$. Therefore the algorithm runs in linear time. ◀

3.2 Configurations of Canonical Split-Trees

In a chord diagram D of a degenerate circle graph G, i.e., of a clique or a star, one half of D determines the other half. More precisely, we can describe a chord diagram of a connected degenerate circle graph $G = (V, E)$ with reference chord $r \in V$ with a cyclic permutation of V

using the following mapping $\phi_{G,r}$ to chord diagrams of G with reference chord r. If G is a clique, we set $\phi_{G,r}(r\rho) = r\rho r\rho$. If G is a star with center x, we set $\phi_{G,r}(x\rho\sigma) = \rho^{\text{rev}} x\rho\sigma x\sigma^{\text{rev}}$ where ρ ends with r or is empty if $x = r$.

▶ **Lemma 5** (⋆). *For a clique or a star $G = (V, E)$ with $r \in V$ the map $\phi_{G,r}$ is a bijection.*

Bouchet [3] showed that the undirected chord diagram of a connected prime circle graph is unique up to reversal. We can additionally choose the orientation of the reference chord. As a shorthand, we set $\text{tr} = \{\text{id}, \text{turn}, \text{rev}, \text{turn}(\text{rev})\}$ and for any chord diagram D we set $\text{tr}(D) = \{D, \text{turn}(D), \text{rev}(D), \text{turn}(\text{rev}(D))\}$. Note that we can have $|\text{tr}(D)| < 4$, e.g., for cliques we have $\text{turn}(D) = D$.

▶ **Lemma 6** (⋆). *Let G be a connected prime circle graph, $r \in V(G)$, and D a chord diagram with reference chord r. Then $|\text{tr}(D)| = 4$ and $\text{tr}(D)$ is the set of chord diagrams of G with reference chord r.*

By combining Theorem 3 with Lemmas 5, 6 we obtain the claimed compact representation of all chord diagrams of a connected circle graph G.

▶ **Theorem 7.** *Let G be a connected circle graph and let T be the canonical split-tree of G with reference chord $r \in V(G)$. Let each prime node μ be equipped with a chord diagram of $\text{skel}(\mu)$ with reference chord r_μ. There is a bijection between the chord diagrams of G with reference chord r and the choices of (i) applying an operation $\tau_\mu \in \text{tr}$ to each prime node μ and (ii) choosing a cyclic permutation of $V(\text{skel}(\mu))$ for each degenerate node μ.*

4 Partial Representation Extension

In this section, we solve RepExt(Circle) for a given circle graph $G = (V, E)$ and a chord diagram D_H of an induced subgraph $H \subseteq G$ in near-linear time. The idea is the following. Assume G is connected and let T be the canonical split-tree of G. By Theorem 3, all chord diagrams of G are represented by T. We project T and its configurations in a sense on H and obtain an enriched split-tree T_H of H. We show that T_H describes exactly the chord diagrams of H that can be extended to G. This means we just need to check whether T_H describes the given chord diagram D_H.

If G is disconnected and there is a connected component C with two predrawn chord ends a, b, and a distinct connected component C' with two predrawn chord ends c, d in D_H such that a, b and c, d alternate in D_H, then there is no extension of D_H to G since C, C' need to cross each other (even though the crossing chords might not be predrawn). Chaplick et al. [4] argue that otherwise an extension exists if and only if each connected component of G admits an extension. Testing this requirement as well as combining representation extensions of the different components can be done in linear time. Hence, we assume in the following that G is connected. Note that H may still be disconnected.

We start with a canonical split-tree T of G rooted at a reference chord $r \in V(H)$, which by Theorem 3 represents all chord diagrams of G. For a chord diagram D of G let $D|_H$ denote the chord diagram for H induced by D (i.e., the chords of H are placed as in D with the same starting point). Let T_H be the subtree of T whose leaves are the vertices of H and whose inner nodes are the inner nodes of T that lie on a path from r to some leaf in $V(H)$. For each inner node of T_H, we define $\text{skel}_{T_H}(\mu)$ as the subgraph of $\text{skel}_T(\mu)$ induced by the vertices $v_\mu(V(T_H))$, i.e., we keep exactly those chords that represent nodes that lead to at least one vertex of H (see Figure 7). Finally, we suppress nodes with K_2 as skeleton in T_H

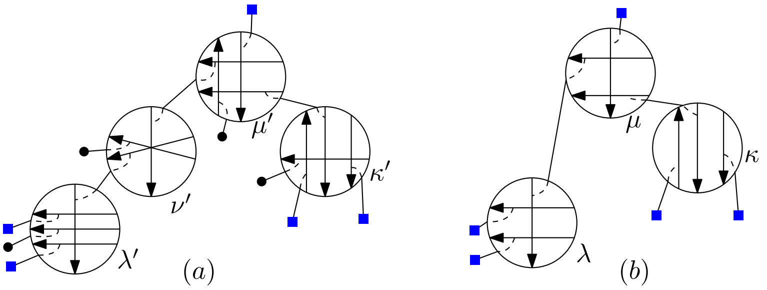


Figure 7 (a) Configuration of a canonical split-tree T. Leaves in $V(H)$ are blue squares. (b) split-tree T_H with the corresponding configuration. Node μ is degenerate in T_H while μ' is prime in T. $\mathrm{skel}(\kappa)$ is an isolated set while $\mathrm{skel}(\kappa')$ is a star.

by (iteratively) joining them with one of their neighbors. Note that T_H is a split-tree of H rooted at r, and each inner node μ of T_H stems from exactly one inner node μ' of T. We call T_H the *projection of T onto H*.

Let now c be a configuration of T. We define its *projection* c_H by setting $c_H(\mu) = c(\mu')|_{\mathrm{skel}_{T_H}(\mu)}$ for each node μ of T_H, i.e., it is the restriction of the chord diagram $c(\mu')$ to $\mathrm{skel}_{T_H}(\mu)$. Observe that the reference chord is not removed, and therefore $c_H(\mu)$ has the same reference chord as $c(\mu')$, i.e., c_H is a configuration of T_H. The following lemma shows that the projection $(T, c) \mapsto (T_H, c_H)$ commutes with all relevant operations.

▶ **Lemma 8.** *We have (i) $\mathcal{D}(c)|_H = \mathcal{D}(c_H)$ and (ii) for every inner node μ of T_H,* $\mathrm{turn}(c_H(\mu)) = \mathrm{turn}(c(\mu'))|_{\mathrm{skel}_{T_H}(\mu)}$ *and* $\mathrm{rev}(c_H(\mu)) = \mathrm{rev}(c(\mu'))|_{\mathrm{skel}_{T_H}(\mu)}$.

Proof. For Property (i) observe that it suffices to show that joining two diagrams commutes with the projection to a subgraph H. It then follows that $\mathcal{D}(c)|_H$, where the join is projected to H, is the same as $\mathcal{D}(c_H)$, where the skeletons are projected before the join, coincide.

More formally, let H_1, H_2 be two induced subgraphs of graphs G_1, G_2, respectively, and let $H = H_1 \oplus_{v_2,v_1} H_2$ and $G = G_1 \oplus_{v_2,v_1} G_2$. We show that for any chord diagrams D_1, D_2 of G_1, G_2, respectively, it is $(D_1 \oplus_{v_2,v_1} D_2)|_H = D_1|_{H_1} \oplus_{v_2,v_1} D_2|_{H_2}$. To see this, let $D_1 = \alpha v_2 \beta v_2 \gamma$ and $D_2 = v_1 \rho v_1 \sigma$ and let $\alpha', \beta', \gamma', \rho', \sigma'$ be the words obtained from $\alpha, \beta, \gamma, \rho, \sigma$ by removing all symbols for chords that are not in $V(H)$. Then we have $(D_1 \oplus_{v_2,v_1} D_2)|_H = (\alpha\rho\beta\sigma\gamma)|_H = \alpha'\rho'\beta'\sigma'\gamma'$. On the other hand, it is $D_1|_{H_1} \oplus_{v_2,v_1} D_2|_{H_2} = \alpha' v_2 \beta' v_2 \gamma' \oplus_{v_2,v_1} v_1 \rho' v_1 \sigma' = \alpha'\rho'\beta'\sigma'\gamma'$.

For Property (ii), let H be an induced subgraph of G. Let $D = r\rho r\sigma$ be a chord diagram for G with reference chord r and let ρ', σ' be the restrictions of ρ, σ to H, respectively. Then $\mathrm{turn}(D)|_H = \mathrm{turn}(r\rho r\sigma)|_H = (r\sigma r\rho)|_H = r\sigma' r\rho' = \mathrm{turn}(r\rho' r\sigma') = \mathrm{turn}((r\rho r\sigma)|_H) = \mathrm{turn}(D|_H)$ and $\mathrm{rev}(D)|_H = \mathrm{rev}(r\rho r\sigma)|_H = (r\sigma^{\mathrm{rev}} r\rho^{\mathrm{rev}})|_H = r\sigma^{\mathrm{rev}} r\rho^{\mathrm{rev}}|_H = r\sigma'^{\mathrm{rev}} r\rho'^{\mathrm{rev}} = \mathrm{rev}(r\rho' r\sigma') = \mathrm{rev}(D|_H)$. ◀

Let D_H be a chord diagram of H. By Theorem 3 there exists a chord diagram of G that extends D_H if and only if there exists a configuration c of T with $\mathcal{D}(c)|_H = D_H$. By Lemma 8(i) this holds if and only if there exists a configuration c of T whose projection c_H satisfies $\mathcal{D}(c_H) = \mathcal{D}(c)|_H = D_H$. We aim to find such a configuration c. To do this, we make use of the property from Lemma 8(ii) as follows. Let c' be an arbitrary configuration of T. By Theorem 7, the configuration c is obtained from c' by (i) choosing a configuration for each degenerate node of T and (ii) by choosing for each prime node μ one of the diagrams $c(\mu) \in \mathrm{tr}(c'(\mu))$. Note that induced subgraphs of cliques are themselves cliques and an induced subgraph of a star is either a star or an independent set. In the latter case

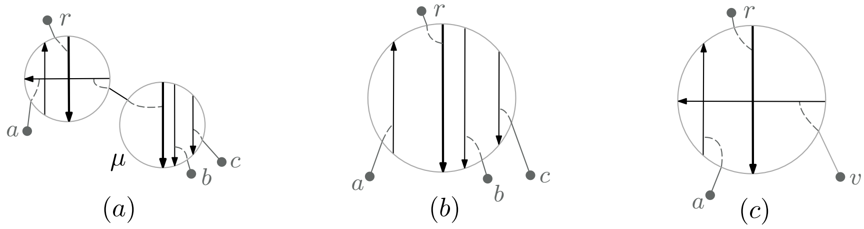


Figure 8 (a) a restricted split-tree for a graph H (b) a chord diagram D_H for H where the chord ends of the leaves of μ are contiguous. (c) the chord diagram we wish to create from D_H by replacing b,c by v. Note that the position of $\hat{v}$ is not neighboring the position of any leaf of μ in (b).

an induced chord diagram has the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$ where the center of the original star has one end between ρ^{rev} and ρ and one end between σ and σ^{rev}. By Lemma 8(ii) it follows that c_H is obtained from c'_H by (i) arbitrarily choosing a configuration for each node of T_H that stems from a degenerate node of T and is connected, (ii) choosing a configuration of the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$ for each node of T_H that stems from a degenerate node of T and is an independent set and (iii) by choosing for each node μ of T_H that stems from a prime node in T one of the diagrams $c_H(\mu) \in \mathrm{tr}(c'_H(\mu))$.

We condense these rules as follows. We label the nodes of T_H as degenerate if they stem from a degenerate node in T and as prime if they stem from a prime node of T. We call two configuration c_H, c'_H of T_H *equivalent* if $c_H(\mu) \in \mathrm{tr}(c'_H(\mu))$ for each prime-labeled node, and for each degenerate-labeled node ν where $\mathrm{skel}(\nu)$ is an independent set, $c_H(\nu)$ is of the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$. We call (T_H, c'_H) *the restricted split-tree of H with respect to G* and say that (T_H, c'_H) *represents* a diagram D_H of H if $D_H = \mathcal{D}(c_H)$ for some configuration c_H that is equivalent to c'_H. By the above observations, D_H can be extended to a diagram of G if and only if D_H is represented by (T_H, c'_H), where c'_H is the projection of a configuration of T.

If H is connected, this condition can be tested in linear time by computing the unique configuration c_H of T_H with $\mathcal{D}(c_H) = D_H$ using Theorem 4 and then checking whether it is equivalent to c'_H. If H is not connected, we use the following lemma to test whether (T_H, c'_H) represents D_H and to obtain a corresponding configuration c_H of T_H in that case.

▶ **Lemma 9** ($\star$). *Let (T_H, c'_H) be a restricted split-tree of a graph H with respect to a connected graph G and let D_H be a chord diagram of H with the root r of T_H as reference chord. It can be tested in linear time whether (T_H, c'_H) represents D_H. If so, a corresponding configuration c_H can also be computed in linear time.*

Sketch of proof. We use a bottom-up approach as in the proof of Theorem 4 to find c_H. Recall that in the proof of Theorem 4, we iteratively processed an inner node μ with only leaves as children and searched for subwords of D_H that correspond to a chord diagram D_μ of $\mathrm{skel}(\mu)$ to check whether D_μ can be part of a configuration and then replace these subwords with a chord that represents μ as a leave for the remaining split-tree. To find these subwords we started at arbitrary leaves of μ to find contiguous sublists of such leaves in D_H. However, if H is not connected, it can happen that when processing a node μ, all its leaves are contiguous in D_H; see Figure 8. For example, this happens if μ is a leftmost leaf of a star node in T that lost its center in T_H (something similar can happen in prime-labeled nodes). In this case the first found sublist $[e_1, e_2]$ of leaves of μ already contains all leaves of μ and we do not find a second sublist $[f_1, f_2]$. This means that we have no pointer to the correct place in the new chord diagram D' for the second end of the leaf chord v that replaces μ. We thus allow D_H to be a relaxed chord diagram where for some chords only one end is fixed. We then use the characterization of inner nodes of restricted split-trees to compute configurations for those nodes that match the given relaxed chord diagrams.

For example, if μ is a labeled degenerate and $\mathrm{skel}(\mu)$ is a star with center x, we start at an end of x in D_H and traverse simultaneously in both directions until all ends are traversed (except possibly the second end of x). At each end of a chord v with no second end, insert that second end where the other traversal is at that moment. When a chord with two ends is reached, wait in front of that chord until the other traversal also reaches that chord and then skip that chord in both traversals. If the traversals wait at different chords we reject. In that case D_H induces the subword $xaabb$ (or some cyclic shifted version of that word) which cannot be realized by a star with center x. If x has a second end, the traversals meet and end there. Otherwise, add the second end of x where the traversals meet. Thereby, all chords for leaves intersect x and no other intersection occurs. By construction we obtain a chord diagram $c_H(\mu)$ that realizes D_H.

Note that both ends of v can be neighbors in $\mathcal{D}(c_H|_{T_H-\mu})$. Then, moving the end $\hat{r}$ of the reference chord in $c_H(\mu)$ does not change $\mathcal{D}(c_H)$ and c_H is no longer required to be unique. ◀

▶ **Theorem 10.** *Given $T = \mathrm{ST}(G)$ and a chord diagram D of G, RepExt(Circle) can be solved in linear time. In the positive case a representation of G that extends the given diagram D_H of H can be computed in the same running time.*

Proof. From D we compute in linear time a configuration c' of T with $\mathcal{D}(c') = D$ using Theorem 4. From T and c', we compute in linear time the projection to the restricted split-tree (T_H, c'_H) of H. With Lemma 9, we test whether it represents the given diagram D_H of H and obtain a configuration c_H equivalent to c'_H with $\mathcal{D}(c_H) = D_H$. We now define a configuration c of T as follows. For each prime-labeled node μ of T_H, we define $c(\mu') = \tau(c'(\mu'))$ where $\tau \in \mathrm{tr}$ such that $c_H(\mu) = \tau(c'_H(\mu))$. For each degenerate-labeled node μ of T_H, we choose a configuration as follows. If $\mathrm{skel}_H(\mu)$ is connected we can extend $c'(\mu)$ arbitrarily. For a precise argument, we have by Lemma 5, $c_H(\mu) = \phi_{\mathrm{skel}_H(\mu),r}(\sigma)$ for some cyclic permutation σ of $V(\mathrm{skel}_{T_H}(\mu))$. We create a permutation σ' of $V(\mathrm{skel}_T(\mu))$ by appending the elements of $V(\mathrm{skel}_T(\mu')) \setminus V(\mathrm{skel}_{T_H}(\mu))$ to σ in an arbitrary order. We then set $c(\mu') = \phi_{H,r}(\sigma')$. If $\mathrm{skel}_{T_H}(\mu)$ is not connected, then $\mathrm{skel}_T(\mu')$ is a star where the reference chord $r_{\mu'}$ is not the center x and $\mathrm{skel}_{T_H}(\mu)$ does not contain x. In that case $c_H(\mu)$ is of the form $c_H(\mu) = \rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$ and we have to insert the other chords in parallel, such that x can intersect all chords. Then set $c(\mu') = \rho^{\mathrm{rev}}x\rho\sigma\alpha x\alpha^{\mathrm{rev}}$, where α are the elements of $V(\mathrm{skel}_T(\mu)) \setminus V(\mathrm{skel}_{T_H}(\mu)) \setminus \{x\}$. Finally, for each node μ' of T that is not contained in T_H, we set $c(\mu') = c'(\mu')$. By construction, we have that c_H is the projection of c, and therefore $\mathcal{D}(c)|_H = \mathcal{D}(c_H) = D_H$, i.e., $\mathcal{D}(c)$ is the desired representation of G. Clearly the amount of work per skeleton is linear, and therefore the overall running time is linear. ◀

Theorem 10 assumes that $\mathrm{ST}(G)$ and a chord diagram of G are available. If not, we can compute them in $O((n+m)\alpha(n+m))$ time [13, 14].

▶ **Corollary 11.** *RepExt(Circle) can be solved in $O((n+m)\alpha(n+m))$ time.*

5 Conclusion

We have developed a data structure that compactly represents all chord diagrams for a connected circle graph. As an application, we have shown how to solve the partial representation extension problem for circle graphs in almost linear time, improving over the $O(n^3)$ algorithm of Chaplick et al. [4]. Using a reduction of Chaplick et al. this also solves the extension problem for permutation graphs in near linear time, improving over two different $O(n^3)$ algorithms [4], [17]. Our data structure may also be useful when seeking restricted chord diagrams that satisfy additional constraints. For example, we believe that it is possible to significantly simplify the circular-arc graph recognition of Hsu et al. [15].

References

1 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Trans. Algorithms*, 11(4):32:1–32:42, 2015. doi:10.1145/2629341.

2 Alan Arroyo, Martin Derka, and Irene Parada. Extending simple drawings. In Daniel Archambault and Csaba D. Tóth, editors, *Proceedings of the 27th International Symposium on Graph Drawing and Network Visualization (GD'19)*, volume 11904 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2019. doi:10.1007/978-3-030-35802-0_18.

3 André Bouchet. Reducing prime graphs and recognizing circle graphs. *Combinatorica*, 7(3):243–254, 1987.

4 Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending partial representations of circle graphs. *J. Graph Theory*, 91(4):365–394, 2019.

5 Bruno Courcelle. Circle graphs and monadic second-order logic. *Journal of Applied Logic*, 6(3):416–442, September 2008. doi:10.1016/j.jal.2007.05.001.

6 William H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, June 1982. doi:10.1137/0603021.

7 William H Cunningham and Jack Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32(3):734–765, 1980.

8 Giuseppe Di Battista and Roberto Tamassia. On-line graph algorithms with SPQR-trees. In M.S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 598–611. Springer, 1990.

9 Eduard Eiben, Robert Ganian, Thekla Hamm, Fabian Klute, and Martin Nöllenburg. Extending partial 1-planar drawings. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, volume 168 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.43.

10 Csaba P Gabor, Kenneth J Supowit, and Wen-Lian Hsu. Recognizing circle graphs in polynomial time. *Journal of the ACM (JACM)*, 36(3):435–473, 1989.

11 Tibor Gallai. Transitiv orientierbare Graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18:25–66, 1967.

12 Emeric Gioan and Christophe Paul. Split decomposition and graph-labelled trees: characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Applied Mathematics*, 160(6):708–733, 2012.

13 Emeric Gioan, Christophe Paul, Marc Tedder, and Derek Corneil. Practical and efficient circle graph recognition. *Algorithmica*, 69(4):759–788, August 2014. doi:10.1007/s00453-013-9745-8.

14 Emeric Gioan, Christophe Paul, Mark Tedder, and Derek Corneil. Practical and efficient split decomposition via graph-labelled trees. *Algorithmica*, 69(4):789–843, 2014. doi:10.1007/s00453-013-9752-9.

15 Wen Lian Hsu. $o(m{\cdot}n)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM Journal on Computing*, 24(3):411–439, 1995. doi:10.1137/S0097539793260726.

16 Vít Kalisz, Pavel Klavík, and Peter Zeman. Circle graph isomorphism in almost linear time. *CoRR*, abs/1908.09151, 2019. arXiv:1908.09151.

17 Pavel Klavík, Jan Kratochvíl, Tomasz Krawczyk, and Bartosz Walczak. Extending partial representations of function graphs and permutation graphs. In Leah Epstein and Paolo Ferragina, editors, *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 671–682. Springer, 2012. doi:10.1007/978-3-642-33090-2_58.

18 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomáš Vyskočil. Extending partial representations of proper and unit interval graphs. In *Algorithm Theory–SWAT 2014*, pages 253–264. Springer, 2014.

19 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomáš Vyskočil. Extending partial representations of interval graphs. *Algorithmica*, 78(3):945–967, 2017.

20 Pavel Klavík, Jan Kratochvíl, and Tomáš Vyskočil. Extending partial representations of interval graphs. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation: 8th Annual Conference, TAMC 2011, Tokyo. Proceedings*, pages 276–285. Springer, 2011. doi:10.1007/978-3-642-20877-5_28.

21 Tomasz Krawczyk and Bartosz Walczak. Extending partial representations of trapezoid graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 10520 of *Lecture Notes in Computer Science*, pages 358–371. Springer, 2017. doi:10.1007/978-3-319-68705-6_27.

22 Saunders Mac Lane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3(3):460–472, 1937.

23 Maurizio Patrignani. On extending a partial straight-line drawing. *Int. J. Found. Comput. Sci.*, 17(5):1061–1070, 2006. doi:10.1142/S0129054106004261.

24 Jeremy Spinrad. Recognition of circle graphs. *Journal of Algorithms*, 16(2):264–282, 1994.

Boundaries to Single-Agent Stability in Additively Separable Hedonic Games

Martin Bullinger
Technical University of Munich, Germany

Abstract

Coalition formation considers the question of how to partition a set of agents into coalitions with respect to their preferences. Additively separable hedonic games (ASHGs) are a dominant model where cardinal single-agent values are aggregated into preferences by taking sums. Output partitions are typically measured by means of stability, and we follow this approach by considering stability based on single-agent movements (to join other coalitions), where a coalition is defined as stable if there exists no beneficial single-agent deviation. Permissible deviations should always lead to an improvement for the deviator, but they may also be constrained by demanding the consent of agents involved in the deviations, i.e., by agents in the abandoned or welcoming coalition. Most of the existing research focuses on the unanimous consent of one or both of these coalitions, but more recent research relaxes this to majority-based consent. Our contribution is twofold. First, we settle the computational complexity of the existence of contractually Nash-stable partitions, where deviations are constrained by the unanimous consent of the abandoned coalition. This resolves the complexity of the last classical stability notion for ASHGs. Second, we identify clear boundaries to the tractability of stable partitions under majority-based stability concepts by proving elaborate hardness results for restricted classes of ASHGs. Slight further restrictions lead to positive results.

2012 ACM Subject Classification Computing methodologies → Multi-agent systems; Theory of computation → Design and analysis of algorithms

Keywords and phrases Coalition Formation, Hedonic Games, Stability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.26

Funding *Martin Bullinger*: Deutsche Forschungsgemeinschaft, grants BR 2312/11-2, BR 2312/12-1.

Acknowledgements I would like to thank Felix Brandt and Leo Tappe for the helpful discussions.

1 Introduction

Coalition formation is a vibrant topic in multi-agent systems at the intersection of theoretical computer science and economic theory. Given a set of agents, e.g., humans or machines, the central concern is to determine a coalition structure, or partition, of the agents into subsets, or so-called coalitions. Agents have preferences over coalition structures, and therefore coalition formation naturally generalizes the matching problem under preferences [22]. As in the special case of matchings, a common assumption is that externalities outside one's own coalition play no role, i.e., agents are only concerned about the coalition they are part of. This assumption leads to the popular framework of hedonic games [18].

In contrast to matchings, the number of coalitions an agent can be part of is not polynomially bounded in coalition formation, and therefore, a lot of effort has been put into identifying reasonable and succinct classes of hedonic games (see, e.g., [2, 5, 8, 20]). In many such classes, agents extract cardinal preferences from a weighted and possibly directed graph by some aggregation method. Probably the most natural and thoroughly researched way to aggregate preferences is by taking the sum of the weights of edges towards agents in one's own coalition. This leads to the concept of additively separable hedonic games (ASHGs) [8]. This paper continues to investigate this class of hedonic games.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 26; pp. 26:1–26:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The desirability of an output, i.e., of a coalition structure, is frequently measured with respect to stability, which captures the prospect of agents maintaining their coalitions. A coalition structure is stable if no single agent or group of agents has an incentive to deviate by leaving their coalitions and joining other coalitions or forming new coalitions. Depending on the requirements that deviators need to meet, one can define various specific stability notions. In this paper, we focus on stability based on single-agent deviations. This means that a deviation consists of a single agent that abandons her current coalition to join another existing coalition or to form a new coalition of her own.

In this case, a reasonable minimum requirement is that a deviating agent should improve her coalition. If no such deviation is possible, then a coalition structure is said to be Nash-stable. However, this leads to an immensely strong stability concept because the deviation is only constrained weakly. As a consequence, Nash-stable outcomes hardly ever exist. For instance, consider a game with two agents x and y where x prefers to form a coalition with y over staying alone, whereas y prefers to stay alone. Then, x always has an incentive to join y whenever she is in a coalition of her own, whereas y would always leave x. Such run-and-chase situations occur in most classes of hedonic games.[1]

Therefore, various weakenings of Nash stability have been proposed. These restrict the possible deviations by adding further requirements on other agents involved in the deviation. Typically, two types of constraints are considered, namely the demanding of some kind of consent from the abandoned or the welcoming coalition. Most of the research has focused on the unanimous consent of these coalitions. This leads to the concepts of contractual Nash stability and individual stability where all agents in the abandoned or welcoming coalition have to approve the deviation. Still, unanimous consent of involved coalitions is a strong requirement. Hence, a reasonable compromise is to merely demand partial consent. Therefore, we also study stability where deviations are constrained by the approval of a majority vote of the abandoned or welcoming coalition.

1.1 Contribution

Our contribution is twofold. First, we settle the complexity of the existence problem of contractually Nash-stable coalition structures. Despite knowing for quite long that No-instances, i.e., additively separable hedonic games which do not admit a contractually Nash-stable coalition structure, exist [28], detailed computational investigations of single-agent stability during the last decade have left this problem open [10, 29]. Hence, we complete the picture of the complexity of unanimity-based single-agent stability concepts in ASHGs.

Second, we investigate majority-based stability concepts. We will show that, even under significant weight restrictions, stable coalition structures need not exist and we can leverage No-instances to obtain computational intractabilities. This complements very recent results by Brandt et al. [10] and resolves problems left open by this work. In particular, we completely pinpoint the complexity of majority-based stability notions in friends-and-enemies games and appreciation-of-friends games.

These results are in line with the repeatedly observed theme in hedonic games research that the existence of counterexamples is the key to computational intractabilities (see, e.g., [3, 10, 11, 16, 29]).[2] On the other hand, we demonstrate that the observed intractabilities lie at the computational boundary by carving out further weak restrictions that lead to the existence and efficient computability of stable states.

[1] Notably, Nash-stable coalition structures always exist in ASHGs if the input graph is symmetric [8], and in a generalization of this class of games called subset-neutral hedonic games [27].

[2] A notable exception is provided by Bullinger and Kober [13] who identify a class of hedonic games where partitions in the core always exist, but are still hard to compute.

1.2 Related Work

The study of hedonic games was initiated by Drèze and Greenberg [18] but was only popularized two decades later by Banerjee et al. [6], Cechlárová and Romero-Medina [15], and Bogomolnaia and Jackson [8]. Aziz and Savani [4] review many important concepts in their survey. Two important research questions concern the design of reasonable computationally manageable subclasses of hedonic games and the detailed investigation of their computational properties. The former has led to a broad landscape of game representations. Some of these representations [5, 20] are ordinal and fully expressive, i.e., they can, in principle, express every preference relation over coalitions. Still, representing certain preference relations requires exponential space. These representations are contrasted by cardinal representations based on weighted graphs [2, 8, 26], which are not fully expressive but only require polynomial space (except when weights are artificially large). Apart from the already discussed additively separable hedonic games, important aggregation methods consider the average of weights leading to the classes fractional hedonic games [2] and modified fractional hedonic games [26]. Additively separable hedonic games have important subclasses where the focus lies in distinguishing friends and enemies, and therefore only two different weights are present in the underlying graph [16].

The computational properties of hedonic games have been extensively studied and we focus on literature related to additively separable hedonic games. Various versions of stability have been investigated [1, 3, 10, 16, 29, 21]. The closest to our work are the detailed studies of single-agent stability by Sung and Dimitrov [29] and Brandt et al. [10]. Gairing and Savani [21] settle the complexity of single-agent stability for symmetric input graphs. Majority-based stability has only received little attention thus far [10, 21]. Apart from stability, other desirable axioms concern efficiency and fairness. Aziz et al. [3] cover a wide range of axioms, whereas Elkind et al. [19] and Bullinger [12] focus on Pareto optimality, and Brandt and Bullinger [9] investigate popularity, an axiom combining ideas from stability and efficiency which is also related to certain majority-based stability notions [10]. Finally, a recent trend in the research on coalition formation is to complement the static view of existence problems by considering dynamics based on stability concepts (see, e.g., [7, 10, 11, 14, 23]).

2 Preliminaries

In this section, we formally introduce hedonic games and our considered stability concepts.

2.1 Hedonic Games

Let $N = [n]$ be a set of $n \in \mathbb{N}$ agents, where we define $[n] = \{1, \dots, n\}$. The output of a coalition formation problem is a coalition structure, that is, a partition of the agents into different disjoint coalitions according to their preferences. A *partition* of N is a subset $\pi \subseteq 2^N$ such that $\bigcup_{C\in\pi} C = N$, and for every pair $C, D \in \pi$, it holds that $C = D$ or $C \cap D = \emptyset$. An element of a partition is called a *coalition* and, given a partition π, the unique coalition containing agent i is denoted by $\pi(i)$. We refer to the partition π given by $\pi(i) = \{i\}$ for every agent $i \in N$ as the *singleton partition*, and to $\pi = \{N\}$ as the *grand coalition*.

Let $\mathcal{N}_i$ denote all possible coalitions containing agent i, i.e., $\mathcal{N}_i = \{C \subseteq N : i \in C\}$. A *hedonic game* is a tuple $(N, \succsim)$, where N is an agent set and $\succsim = (\succsim_i)_{i\in N}$ is a tuple of weak orders $\succsim_i$ over $\mathcal{N}_i$ representing the preferences of the respective agent i. Hence, as mentioned before, agents express preferences only over the coalitions of which they are part without considering externalities. The strict part of an order $\succsim_i$ is denoted by $\succ_i$, i.e., $C \succ_i D$ if and only if $C \succsim_i D$ and not $D \succsim_i C$.

Additively separable hedonic games assume that every agent is equipped with a cardinal utility function that is aggregated by taking the sum of single-agent values. Formally, following [8], an *additively separable hedonic game* (ASHG) (N, v) consists of an agent set N and a tuple $v = (v_i)_{i \in N}$ of utility functions $v_i \colon N \to \mathbb{R}$ such that $\pi \succsim_i \pi'$ if and only if $\sum_{j \in \pi(i)} v_i(j) \geq \sum_{j \in \pi'(i)} v_i(j)$. Clearly, ASHGs are a subclass of hedonic games. When we specify ASHG utilities, we neglect, without loss of generality, $v_i(i)$ because the preferences do not depend on it and we implicitly assume that it is set to an appropriate constant if an ASHG has to fit into a certain subclass of games.

Every ASHG can be naturally represented by a complete directed graph $G = (N, E)$ with weight $v_i(j)$ on arc (i, j). There are various subclasses of ASHGs that allow a natural interpretation in terms of friends and enemies. An agent $j \in N$ is called a *friend* (or *enemy*) of agent $i \in N$ if $v_i(j) > 0$ (or $v_i(j) < 0$). An ASHG is called a *friends-and-enemies game* (FEG) if $v_i(j) \in \{-1, 1\}$ for every pair of agents $i, j \in N$ [10]. Further, following [16], an ASHG is called an *appreciation-of-friends game* (AFG) (or an *aversion-to-enemies game* (AEG)) if $v_i(j) \in \{-1, n\}$ (or $v_i(j) \in \{-n, 1\}$). In such games, agents seek to maximize their number of friends while minimizing their number of enemies, where these goals have a different priority in each case. Based on the friendship of agents, we define the *friendship relation* (or *enemy relation*) as the subset $R \subseteq N \times N$ where $(i, j) \in R$ if and only if $v_i(j) > 0$ (or $v_i(j) < 0$).

2.2 Single-Agent Stability

We want to study stability under single agents' incentives to perform deviations. A *single-agent deviation* performed by agent i transforms a partition π into a partition π' where $\pi(i) \neq \pi'(i)$ and, for all agents $j \neq i$,

$$\pi'(j) = \begin{cases} \pi(j) \setminus \{i\} & \text{if } j \in \pi(i), \\ \pi(j) \cup \{i\} & \text{if } j \in \pi'(i), \text{ and} \\ \pi(j) & \text{otherwise.} \end{cases}$$

We write $\pi \xrightarrow{i} \pi'$ to denote a single-agent deviation performed by agent i transforming partition π to partition π'.

We consider myopic agents whose rationale is to only engage in a deviation if it immediately makes them better off. A *Nash deviation* is a single-agent deviation performed by agent i making her better off, i.e., $\pi'(i) \succ_i \pi(i)$. Any partition in which no Nash deviation is possible is said to be *Nash-stable* (NS).

Following [10], we introduce consent-based stability concepts via favor sets. Let $C \subseteq N$ be a coalition and $i \in N$ an agent. The *favor-in set* of C with respect to i is the set of agents in C (excluding i) that strictly favor having i inside C rather than outside, i.e., $F_{\text{in}}(C, i) = \{j \in C \setminus \{i\} \colon C \cup \{i\} \succ_j C \setminus \{i\}\}$. The *favor-out set* of C with respect to i is the set of agents in C (excluding i) that strictly favor having i outside C rather than inside, i.e., $F_{\text{out}}(C, i) = \{j \in C \setminus \{i\} \colon C \setminus \{i\} \succ_j C \cup \{i\}\}$.

An *individual deviation* (or *contractual deviation*) is a Nash deviation $\pi \xrightarrow{i} \pi'$ such that $F_{\text{out}}(\pi'(i), i) = \emptyset$ (or $F_{\text{in}}(\pi(i), i) = \emptyset$). Then, a partition is said to be *individually stable* (IS) or *contractually Nash-stable* (CNS) if it allows for no individual or contractual deviation, respectively. A related weakening of both stability concepts is contractual individual stability (CIS), based on deviations that are both individual and contractual deviations [8, 17].

Finally, we define hybrid stability concepts according to [10] where the consent of the abandoned or welcoming coalition is decided by a majority vote. A Nash deviation $\pi \xrightarrow{i} \pi'$ is

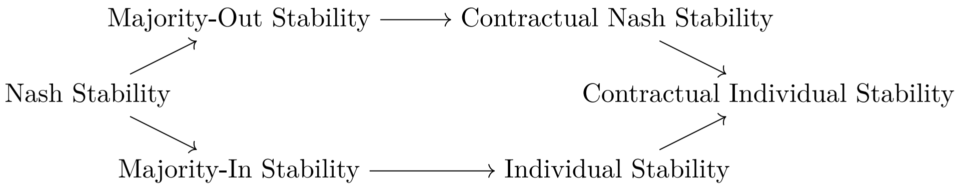


Figure 1 Logical relationships between stability notions. An arrow from concept S to concept S' indicates that if a partition satisfies S, it also satisfies S'. Conversely, this means that every S' deviation is also an S deviation.

called a *majority-in deviation* (or *majority-out deviation*) if $|F_{\text{in}}(\pi'(i), i)| \geq |F_{\text{out}}(\pi'(i), i)|$ (or $|F_{\text{out}}(\pi(i), i)| \geq |F_{\text{in}}(\pi(i), i)|$). Similar to before, a partition is said to be *majority-in stable* (MIS) or *majority-out stable* (MOS) if it allows for no majority-in or majority-out deviation, respectively. The concepts MIS and MOS are special cases of the voting-based stability notions by Gairing and Savani [21] for a threshold of 1/2. Brandt et al. [10] also consider stability concepts that require voting-based consent by both the abandoned and welcoming coalition, similar to CIS.

For a stability concept $S \in \{\text{NS}, \text{IS}, \text{CNS}, \text{MIS}, \text{MOS}\}$, we denote the deviation corresponding to S as *S deviation*, e.g., CNS deviation for a contractual deviation. A taxonomy of our related solution concepts is provided in Figure 1.

3 Contractual Nash Stability

Our first result settles the computational complexity of contractual Nash stability in ASHGs. All of our reductions in this and the subsequent sections are from the NP-complete problem Exact3Cover (E3C) [25]. An instance of E3C consists of a tuple (R, S), where R is a ground set together with a set S of 3-element subsets of R. A Yes-instance is an instance such that there exists a subset $S' \subseteq S$ that partitions R.

Before giving the complete proof, we briefly describe the key ideas. Given an instance (R, S) of E3C, the reduced instance consists of three types of gadgets. First, every element in R is represented by a subgame that does not contain a CNS partition. In principle, any such game can be used for a reduction, and we use the game identified by Sung and Dimitrov [28]. Moreover, we have further auxiliary gadgets that also consist of the same No-instance. The number of these auxiliary gadgets is equal to the number of sets in S that would remain after removing an exact cover of R, i.e., there are $|S| - |R|/3$ such gadgets. By design, the agents in the subgames corresponding to No-instances have to form coalitions with agents outside of their subgame in every CNS partition. The only agents that can achieve this are agents in gadgets corresponding to elements in S. A gadget corresponding to an element $s \in S$ can either prevent non-stability caused by exactly one auxiliary gadget, or by the three gadgets corresponding to the elements $r \in R$ with $r \in s$. Hence, the only possibility to deal with all No-instances simultaneously is if there exists an exact cover of R by sets in S. Then, the gadgets corresponding to elements in R can be dealt with by the cover and there are just enough elements in S to additionally deal with the other auxiliary gadgets.

▶ **Theorem 1.** *Deciding whether an ASHG contains a CNS partition is NP-complete.*

Proof. We provide a reduction from E3C. Let (R, S) be an instance of E3C and set $a = |S| - |R|/3$ (this is the number of additional sets in S if removing some exact cover). Without

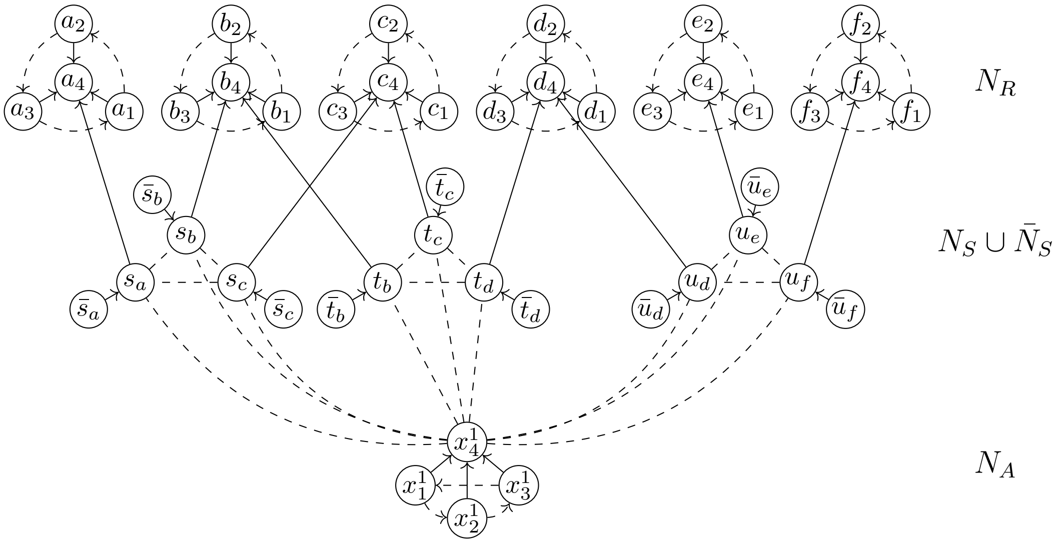


Figure 2 Schematic of the reduction from the proof of Theorem 1. We depict the reduced instance for the instance (R, S) of E3C where $R = \{a, b, c, d, e, f\}$, and $S = \{s, t, u\}$, with $s = \{a, b, c\}$, $t = \{b, c, d\}$, and $u = \{d, e, f\}$. Fully drawn edges mean a positive utility, which is usually 1 except between agents of the types $\bar{s}_r$ and s_r, where $v_{\bar{s}_r}(s_r) = 3$. Dashed edges represent a utility of 0. For agents in $\bar{N}_S$, only the single positive utility is displayed. Other omitted edges represent a negative utility of -4.

loss of generality, $a \geq 0$. We define an ASHG (N, v) as follows. Let $N = N_R \cup N_S \cup \bar{N}_S \cup N_A$ where

- $N_R = \cup_{r \in R} N_r$ with $N_r = \{r_i : i \in [4]\}$ for $r \in R$,
- $N_S = \cup_{s \in S} N_s$ with $N_s = \{s_r : r \in s\}$ for $s \in S$,
- $\bar{N}_S = \cup_{s \in S} \bar{N}_s$ with $\bar{N}_s = \{\bar{s}_r : r \in s\}$ for $s \in S$, and
- $N_A = \cup_{1 \leq j \leq a} N^j$ with $N^j = \{x_i^j : i \in [4]\}$ for $1 \leq j \leq a$.

We define valuations v as follows:

- For each $r \in R$, $i \in [3]$: $v_{r_i}(r_4) = 1$.
- For each $r \in R$, $(i, j) \in (1, 2), (2, 3), (3, 1)$: $v_{r_i}(r_j) = 0$.
- For each $1 \leq j \leq a$, $i \in [3]$: $v_{x_i^j}(x_4^j) = 1$.
- For each $1 \leq j \leq a$, $(i, k) \in (1, 2), (2, 3), (3, 1)$: $v_{x_i^j}(x_k^j) = 0$.
- For each $s \in S$, $r \in s$: $v_{s_r}(r_4) = 1$.
- For each $s \in S$, $r \in s$, $1 \leq j \leq a$: $v_{s_r}(x_4^j) = v_{x_4^j}(s_r) = 0$.
- For each $s \in S$, $r, r' \in s$: $v_{s_r}(s_{r'}) = 0$.
- For each $s \in S$, $r, r' \in s$, $r \neq r'$, $z \in (N_S \cup N_A) \setminus N_s$: $v_{\bar{s}_r}(s_r) = 3$, $v_{\bar{s}_r}(s_{r'}) = -2$, and $v_{\bar{s}_r}(z) = 0$.
- All other valuations are -4.

An illustration of the game is given in Figure 2. The agents in N_R in the reduced instance form gadgets consisting of a subgame without CNS partition for every element in R. The agents in N_A constitute further such gadgets. The agents in N_S consist of triangles for every set in S and are the only agents who can bind agents in the gadgets in any CNS partition. Finally, agents in $\bar{N}_S$ avoid having agents in N_S in separate coalitions to bind agents in N_A.

We claim that (R, S) is a Yes-instance if and only if (N, v) contains a CNS partition. Suppose first that $S' \subseteq S$ partitions R. Consider any bijection $\phi\colon S \setminus S' \to [a]$. Define a partition π by taking the union of the following coalitions:

- For every $r \in R, i \in [3]$, form $\{r_i\}$.
- For $s \in S', r \in s$, form $\{s_r, r_4\}$.
- For $s \in S \setminus S'$, form $\{s_r : r \in s\} \cup \{x_4^{\phi(s)}\}$.
- For $s \in S, r \in s$, form $\{\bar{s}_r\}$.
- For $1 \leq j \leq a, i \in [3]$, form $\{x_i^j\}$.

We claim that π is CNS. We will show that no agent can perform a deviation.

- For $r \in R$, $i \in [3]$, it holds that $v_{r_i}(\pi) = 0$ and joining any other coalition results in a negative utility. In particular, $v_{r_i}(\pi(r_4) \cup \{r_i\}) = -3$.
- For $r \in R$, r_4 is not allowed to leave her coalition.
- For $s \in S'$, $r \in s$, it holds that $v_{s_r}(\pi) = 1$ and joining any other coalition results in a negative utility. The agent s_r is in a most preferred coalition.
- For $s \in S \setminus S'$, $r \in s$, it holds that $v_{s_r}(\pi) = 0$ and joining any other coalition results in a negative utility. In particular, $v_{s_r}(\pi(r_4) \cup \{s_r\}) = -3$.
- For $s \in S'$, $r \in s$, the agent $\bar{s}_r$ obtains a non-positive utility by joining any other coalition. In particular, $v_{\bar{s}_r}(\pi(s_r) \cup \{\bar{s}_r\}) = -1$.
- For $s \in S \setminus S'$, $r \in s$, the agent $\bar{s}_r$ obtains a non-positive utility by joining any other coalition. In particular, $v_{\bar{s}_r}(\pi(s_r) \cup \{\bar{s}_r\}) = -1$.
- For $1 \leq j \leq a$, $i \in [3]$, it holds that $v_{x_i^j}(\pi) = 0$ and joining any other coalition results in a negative utility. In particular, $v_{x_i^j}(\pi(x_4^j) \cup \{x_i^j\}) = -11$.
- For $1 \leq j \leq a$, x_4^j is in a best possible coalition (achieving utility 0).

Conversely, assume that (N, v) contains a CNS partition π. Define $S' = \{s \in S \colon \pi(s_r) \cap N_R \neq \emptyset$ for some $r \in s\}$. We will show first that S' covers all elements in R and then show that $|S'| = |R|/3$.

Let $r \in R$. Then, for all $i \in [3]$, $\pi(r_i) \subseteq N_r$. This follows because there is no agent who favors r_i in her coalition. Therefore, she would leave any coalition with an agent outside N_r to receive non-negative utility in a singleton coalition. Further, if there is no $s \in S$ with $r \in s$ such that $r_4 \in \pi(r_s)$, then $\pi(r_4) \subseteq N_r$. Indeed, if r_4 forms any coalition except a singleton coalition, she will receive negative utility, and then there must exist an agent who favors her in the coalition. Consequently, if $r_4 \notin \pi(r_s)$ for all $s \in S$ with $r \in s$, then r_4 is in a singleton coalition, or there exists $i \in [3]$ with $r_4 \in \pi(r_i)$, for which we already know that $\pi(r_i) \subseteq N_r$.

Assume now that $\pi(r_4) \subseteq N_r$. For $i, i' \in [3]$, $r_i \notin \pi(r_{i'})$ because then one of them would receive a negative utility and could perform a CNS deviation to form a singleton coalition. If $\{r_4\} \in \pi$, then r_1 would deviate to join her. Hence, there exists exactly one $i \in [3]$ with $\{r_i, r_4\} \in \pi$. Suppose without loss of generality that $\{r_1, r_4\} \in \pi$. But then, r_3 would perform a CNS deviation to join them, a contradiction. We can conclude that there exists $s \in S$ with $r \in s$ such that $r_4 \in \pi(r_s)$. Hence, $s \in S'$ and we have shown that S' covers R.

To bound the cardinality of S', we will show that, for every $1 \leq j \leq a$, there exists $s \in S \setminus S'$ with $N_s \subseteq \pi(x_4^j)$. Let therefore $1 \leq j \leq a$ and let $C = \pi(x_4^j)$. Similar to the considerations about agents in N_r, we know that $\pi(x_i^j) \subseteq X^j$ for $i \in [3]$, and that it cannot happen that $C \subseteq X^j$, and therefore $C \cap X^j = \{x_4^j\}$. In particular, there must be an agent $y \in N \setminus X^j$ with $y \in C$. Since no agent in C favors x_4^j to be in her coalition, we know that $v_{x_4^j}(\pi) \geq 0$ and therefore $C \subseteq \{x_4^j\} \cup N_S$. Let $s \in S$ and $r \in s$ with $s_r \in C$. As we already know that $\bar{s}_r \notin C$, it must hold that $N_s \subseteq C$ to prevent her from joining. It follows that $s \notin S'$. Since $\pi(x_4^j) \cap \pi(x_4^{j'}) = \emptyset$ for $1 \leq j' \leq a$ with $j' \neq j$, we find an injective mapping $\phi\colon [a] \to S \setminus S'$ such that, for every $1 \leq j \leq a$, $N_{\phi(j)} \subseteq \pi(x_4^j)$. Consequently, $|S'| \leq |S| - |\phi([a])| \leq |S| - a = |R|/3$. Hence, S' covers all elements from R with (at most) $|R|/3$ sets and therefore is an exact cover. ◀

The reduction in the previous proof only uses a very limited number of different weights, namely the weights in the set $\{1, 0, -2, -4\}$, where the weight -4 may be replaced by an arbitrary smaller weight. By contrast, CNS partitions always exist if the utility functions of an ASHG assume at most one nonpositive value, and can be computed efficiently in this case [10, Theorem 4]. This encompasses for instance FEGs, AFGs, and AEGs. Hence, the hardness result is close to the boundary of computational feasibility.

4 Appreciation-of-Friends Games

In this section, we consider appreciation-of-friends games. Typically, these games behave well with respect to stability. In particular, IS, CNS, and MIS partitions always exist and can be computed efficiently, while it is only known that NS leads to non-existence and computational hardness among single-agent stability concepts [10, 16]. By contrast, we show in our next result that MOS partitions need not exist in AFGs. In other words, despite their conceptual complementarity, the stability concepts MOS and MIS lead to very different behavior in a natural class of ASHGs. The constructed game has a sparse friendship relation in the sense that almost all agents only have a single friend. After discussing the counterexample, we show how requiring slightly more sparsity yields a positive result. Due to space restrictions, some proofs are omitted or sketched.

▶ **Proposition 2.** *There exists an AFG without an MOS partition.*

Proof. We define the game formally. An illustration is given in Figure 3. Let $N = \{z\} \cup \bigcup_{x\in\{a,b,c\}} N_x$, where $N_x = \{x_i \colon i \in [5]\}$ for $x \in \{a, b, c\}$. In the whole proof, we read indices modulo 5, mapping to the respective representative in [5]. The utilities are given as:

- For all $i \in [5], x \in \{a, b, c\}\colon v_{x_i}(x_{i+1}) = n$.
- For all $x \in \{a, b, c\}\colon v_{x_1}(z) = n$.
- All other valuations are -1.

The AFG consists of 3 cycles with 5 agents each, together with a special agent that is liked by a fixed agent of each cycle and has no friends herself. The key insight to understanding why there exists no MOS partition is that agents of type x_1 where $x \in \{a, b, c\}$ have conflicting candidate coalitions in a potential MOS partition. Either, they want to be with z (a coalition that has to be small because z prefers to stay alone) or they want to be with x_2 which requires a rather large coalition containing their cycle.

Before going through the proof that this game has no MOS partition, it is instructional to verify that, for cycles of 5 agents, the unique MOS partition is the grand coalition, i.e., the unique MOS partition of the game restricted to N_x is $\{N_x\}$, where $x \in \{a, b, c\}$. This is a key idea of the construction and is implicitly shown in Case 2 of the proof for $x = b$.

Assume for contradiction that the defined AFG admits an MOS partition π. To derive a contradiction, we perform a case distinction over the coalition sizes of z.

Case 1. $|\pi(z)| = 1$.

In this case, it holds that $\pi(z) = \{z\}$. Then, $\pi(a_1) \in \{\{a_1, a_2\}, \{a_1, a_5\}\}$. Indeed, if $\pi(a_1) \neq \{a_1, a_2\}$, then a_1 has an NS deviation to join z, and is allowed to perform it unless $\pi(a_1) = \{a_1, a_5\}$. We may therefore assume that $\{a_i, a_{i+1}\} \in \pi$ for some $i \in \{1, 5\}$.

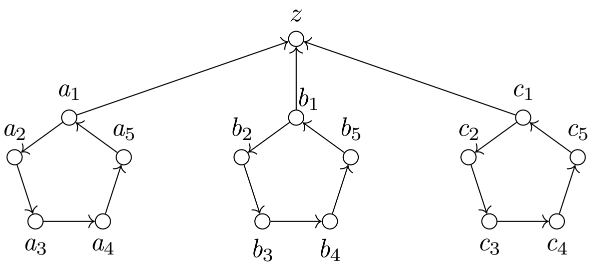


Figure 3 AFG without an MOS partition. The depicted (directed) edges represent friends, i.e., a utility of n, whereas missing edges represent a utility of -1.

Then, $\pi(a_{i-1}) = \{a_{i-1}, a_{i-2}\} =: C$. Otherwise, a_{i-1} can perform an MOS deviation to join $\{a_i, a_{i+1}\}$. But then a_{i+2} can perform an MOS deviation to join C. This is a contradiction and concludes the case that $|\pi(z)| = 1$.

Case 2. $|\pi(z)| > 1$.

Let $F := \{a_1, b_1, c_1\}$, i.e., the set of agents that have z as a friend. Note that z can perform an NS deviation to be a singleton. Hence, as π is MOS, $|F \cap \pi(v)| \geq |\pi(z)|/2$. In particular, there exists an $x \in \{a, b, c\}$ with $\pi(z) \cap N_x = \{x_1\}$. We may assume without loss of generality that $\pi(z) \cap N_a = \{a_1\}$. Then, $\pi(a_5) = \{a_4, a_5\}$. Otherwise, a_5 has an MOS deviation to join $\pi(z)$. Similarly, $\pi(a_3) = \{a_2, a_3\}$ (because of the potential deviation of a_3 who would like to join $\{a_4, a_5\}$). Now, note that $v_{a_1}(\{a_1, a_2, a_3\}) = n - 1$. We can conclude that $|\pi(z)| \leq 3$ as a_1 would join $\{a_2, a_3\}$ by an MOS deviation, otherwise. Hence, we find $x \in \{b, c\}$ with $N_x \cap \pi(z) = \emptyset$. Assume without loss of generality that $x = b$ has this property.

Assume first that $\pi(b_1) = \{b_1, b_5\}$. Then, $\pi(b_4) = \{b_3, b_4\}$. Otherwise, b_4 has an MOS deviation to join $\{b_1, b_5\}$. But then b_2 has an MOS deviation to join $\{b_3, b_4\}$, a contradiction. Hence, $\pi(b_1) \neq \{b_1, b_5\}$. Note that we have now excluded the only case where b_1 is not allowed to perform an NS deviation. In all other cases, no majority of agents prefers her to stay in the coalition. We can conclude that $b_2 \in \pi(b_1)$ because otherwise, b_1 can perform an MOS deviation to join $\pi(z)$. If $b_5 \notin \pi(b_1)$, then $\pi(b_5) = \{b_4, b_5\}$ (to prevent a potential deviation by b_5). But then b_3 has an MOS deviation to join them. Hence, $b_5 \in \pi(b_1)$. Similarly, if $b_4 \notin \pi(b_1)$, then $\pi(b_4) = \{b_3, b_4\}$ and b_2 has an MOS deviation to join $\{b_3, b_4\}$ (which is permissible because $b_5 \in \pi(b_1)$). Hence $\{b_1, b_2, b_4, b_5\} \subseteq \pi(b_1)$, and therefore even $N_b \subseteq \pi(b_1)$. Hence, b_1 has an MOS deviation to join $\pi(v)$ (recall that $|\pi(v)| \leq 3$). This is the final contradiction, and we can conclude that π is not MOS. ◀

Note that most agents in the previous example have at most 1 friend (only three agents have 2 friends). By contrast, if every agent has at most one friend, MOS partitions are guaranteed to exist. This is interesting because it covers in particular directed cycles, which cause problems for Nash stability. The constructive proof of the following proposition can be directly converted into a polynomial-time algorithm.

▶ **Proposition 3.** *Every AFG where every agent has at most one friend admits an MOS partition.*

Proof. We prove the statement by induction over n. Clearly, the grand coalition is MOS for $n = 1$. Now, assume that (N, v) is an AFG with $n \geq 2$ such that every agent has at most one friend. Consider the underlying directed graph $G = (N, A)$ where $(x, y) \in A$ if and only if $v_x(y) > 0$, i.e., y is a friend of x. By assumption, G has a maximum out-degree of 1, hence it can be decomposed into directed cycles and a directed acyclic graph.

Assume first that there exists $C \subseteq N$ such that C induces a directed cycle in G. We call an agent y *reachable* by agent x if there exists a directed path in G from x to y. Let $c \in C$ and define $R = \{x \in N \colon c \text{ reachable by } x\}$. Note that $C \subseteq R$ and that R is identical to the set of agents that can reach *any* agent in C. By induction, there exists an MOS partition π' of the subgame of (N, v) induced by $N \setminus R$ that is MOS. Define $\pi = \pi' \cup \{R\}$. We claim that π is MOS. Let $x \in N \setminus R$. By our assumptions on π', there exists no MOS deviation of x to join $\pi(y)$ for $y \in N \setminus R$. In particular, if x is allowed to perform a deviation, then x must have a non-negative utility (otherwise, she can form a singleton coalition contradicting that π' is MOS). So her only potential deviations are to a coalition where she has a friend. Note that x has no friend in R. Indeed, if y was a friend of x in R, then c is reachable for x in G through the concatenation of (x, y) and the path from y to c. Hence, x has no MOS deviation. Now, let $x \in R$. Then, $v_x(\pi) > 0$ because she forms a coalition with her unique friend. By assumption, x has no friend in any other coalition. Therefore, x has no MOS deviation either.

We may therefore assume that G is a directed acyclic graph. Hence, there exists an agent $x \in N$ with in-degree 0. If x has no friend, let $T = \{x\}$. If x has a friend y, we claim that there exists an agent w such that (i) w is the friend of at least one agent and (ii) every agent that has w as a friend has in-degree 0, i.e., such agents are not the friend of any agent. We provide a simple linear-time algorithm that finds such an agent. We will maintain a tentative agent w that will continuously fulfill (i) and update w until this agent also fulfills (ii). Start with $w = y$. Note that this agent w fulfills (i) because y is a friend of x. If w is the friend of some agent z that is herself the friend of some other agent, update $w = z$. For the finiteness (and efficient computability) of this procedure, consider a topological order σ of the agents N in the directed acyclic graph G [24], i.e., a function $\sigma \colon N \to [n]$ such that $\sigma(a) < \sigma(b)$ whenever $(a, b) \in A$. Note that if w is replaced by the agent z in the procedure, then $\sigma(z) < \sigma(w)$. Hence, w is replaced at most n times, and our procedure finds the desired agent w after a linear number of steps. Now, define $T = \{a \in N \colon w \text{ reachable by } a\}$, i.e., T contains precisely w and all agents that have w as a friend.

We are ready to find the MOS partition. By induction, we find a partition π' that is MOS for the subgame induced by $N \setminus T$. Consider $\pi = \pi' \cup \{T\}$. Then, $a \in T \setminus \{w\}$ has no incentive to deviate, because she has no friend in any other coalition and has w as a friend. Also, w is not allowed to perform a deviation, because the non-empty set of agents $T \setminus \{w\}$ unanimously prevents that. Possible deviations by agents in $N \setminus T$ can be excluded as in the first part of the proof because these agents have no friend in T. Together, we have completed the induction step and found an MOS partition. ◀

On the other hand, it is NP-complete to decide whether an AFG contains an MOS partition. For a proof, we use the game constructed in Proposition 2 as a gadget in a greater game. The difficulty is to preserve bad properties about the existence of MOS partitions because the larger game might allow for new possibilities to create coalitions with the agents in the counterexample.

▶ **Theorem 4.** *Deciding whether an AFG contains an MOS partition is NP-complete.*

5 Friends-and-Enemies Games

Friends-and-enemies games always contain efficiently computable stable coalition structures with respect to the unanimity-based stability concepts IS and CNS [10]. In this section, we will see that the transition to majority-based consent crosses the boundary of tractability.

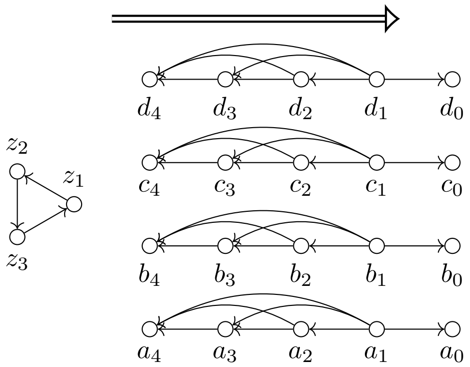


Figure 4 FEG without an MOS partition. The depicted (directed) edges represent friends. The double arrow means that every agent to the left of the tail of the arrow has every agent below the arrow as a friend.

The closeness to this boundary is also emphasized by the fact that it is surprisingly difficult to even construct No-instances for MOS and MIS, i.e., FEGs which do not contain an MOS or MIS partition, respectively. Indeed, the smallest such games that we can construct are games with 23 and 183 agents, respectively. We will start by considering MOS.

▶ **Proposition 5.** *There exists an FEG without an MOS partition.*

Proof sketch. We only give a brief overview of the instance by means of the illustration in Figure 4. The FEG consists of a triangle of agents together with 4 sets of agents whose friendship relation is complete and transitive, together with one additional agent each that gives a temptation for the agent of the transitive substructures with the most friends.

An important reason for the non-existence of MOS partitions is that there is a high incentive for the transitive structures to form coalitions. This gives incentive to agents z_i to join them. If z_1, z_2, and z_3 are in disjoint coalitions, then they would chase each other according to their cyclic structure. If they are all in the same coalition, then agents x_0 for $x \in \{a, b, c, d\}$ prevent the complete transitive structures to be part of this coalition and other transitive structures are more attractive. ◀

In the previous proof, it is particularly useful to establish disjoint coalitions of groups of agents who dislike each other. On the other hand, if we make the further assumption that one agent from every pair of agents likes the other agent, then this does not work anymore and the grand coalition is MOS. This condition essentially means completeness of the friendship relation.[3] Note that this proposition is not true for other stability concepts such as NS or even IS.

▶ **Proposition 6.** *The grand coalition is MOS in every FEG with complete friendship relation.*

Proof. Let (N, v) be an FEG with complete friendship relation, and let π be the grand coalition. We claim that π is MOS. Suppose that there is an agent $x \in N$ who can perform an NS deviation to form a singleton.

[3] Technically, the friendship relation may not be reflexive, but we can set $v_i(i) = 1$ for all $i \in N$ in an FEG to formally achieve completeness.

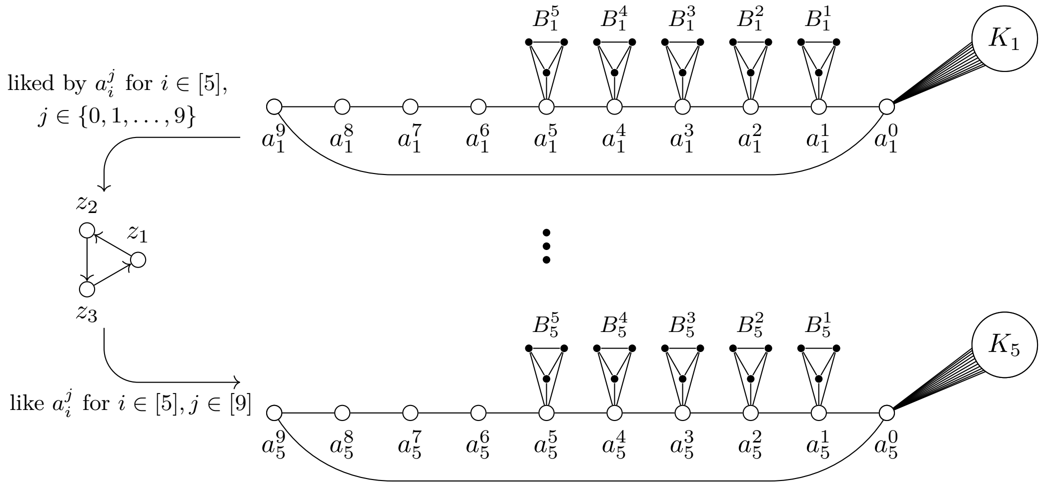


Figure 5 FEG without an MIS partition. The depicted edges represent friends. Undirected edges represent mutual friendship. For $i \in [5]$, some of the edges of agents in A_i are omitted. In fact, these agents form cliques. Also, each K_i represents a clique of 11 agents.

Then, $v_x(N) < 0$ and therefore $|\{y \in N \setminus \{x\} : v_x(y) = -1\}| > \{y \in N \setminus \{x\} : v_x(y) = 1\}|$. Hence,

$$\begin{aligned} |F_{\text{in}}(N, x)| &\geq |\{y \in N \setminus \{x\} : v_x(y) = -1\}| \\ &> |\{y \in N \setminus \{x\} : v_x(y) = 1\}| \\ &\geq |F_{\text{out}}(N, x)|. \end{aligned}$$

In the first inequality, we use that x is a friend of all of her enemies. In the final inequality, we use that x can only be an enemy of her friends. Hence, x is not allowed to perform an MOS deviation. ◀

Still, the non-existence of MOS partitions in FEGs shown in Proposition 5 can be leveraged to prove an intractability result. Interestingly, in contrast to the proofs of Theorem 1 and Theorem 4, the next theorem merely uses the existence of an FEG without an MOS partition to design a gadget and does not exploit the specific structure of a known counterexample.

▶ **Theorem 7.** *Deciding whether an FEG contains an MOS partition is NP-complete.*

In our next result, we construct an FEG without an MIS partition. Despite a lot of structure, the game is quite large encompassing 183 agents.

▶ **Proposition 8.** *There exists an FEG without an MIS partition.*

Proof sketch. We illustrate the example with the aid of Figure 5 and briefly discuss some key features. Again, the central element is a directed cycle of three agents. These agents are connected to five copies of the same gadget. This gadget consists of a main clique $\{a_i^0, \ldots, a_i^9\}$ of 10 mutual friends and further cliques that cause certain temptations for agents in the main clique. Cliques are linked by agents that have an incentive to be part of two cliques, which are part of disjoint coalitions. Since it is possible to balance all diametric temptations, the instance does not admit an MIS partition. ◀

Similar to Proposition 6, it is easy to see that the singleton partition is MIS in every FEG with complete enemy relation. Indeed, then an agent either has no incentive to join another agent, or the other agent will deny her consent. Hence, MIS can also prevent typical run-and-chase games which do not admit NS partitions. We are ready to prove hardness of deciding on the existence of MIS partitions in FEGs.

▶ **Theorem 9.** *Deciding whether an FEG contains an MIS partition is NP-complete.*

6 Discussion and Conclusion

We have investigated single-agent stability in additively separable hedonic games. Our main results determine strong boundaries to the efficient computability of stable partitions. Table 1 provides a complete picture of the computational complexity of all considered stability notions and subclasses of ASHGs, where our results close all remaining open problems. First, we resolve the computational complexity of computing CNS partitions, which considers the last open unanimity-based stability notion in unrestricted ASHGs. The derived hardness result stands in contrast to positive results when considering appropriate subclasses such as FEGs, AEGs, or AFGs [10]. Second, our intractability concerning AFGs stands in contrast to known positive results for all other consent-based stability notions, and can also be circumvented by considering AFGs with a sparse friendship relation. Finally, we provide sophisticated hardness proofs for majority-based stability concepts in FEGs. These turn into computational feasibilities when transitioning to unanimity-based stability, or under further assumptions to the structure of the friendship graph.

A key step of all hardness results in restricted classes of ASHGs was to construct the first No-instances, that is, games that do not admit stable partitions for the respective stability notion. This is no trivial task as can be seen from the complexity of the constructed games. Once No-instances are found, we can leverage them as gadgets of hardness reductions, which is a typical approach for complexity results about hedonic games. We have provided both reductions where the explicit structure of the determined No-instances is used as well as reductions where the mere existence of No-instances is sufficient and used as a black box.

Our results complete the picture of the computational complexity for all considered stability notions and game classes. Still, majority-based stability notions deserve further attention because they offer a natural degree of consent to perform deviations. Their thorough investigation in other classes of hedonic games might lead to intriguing discoveries.

Table 1 Overview of the computational complexity of single-agent stability concepts in different classes of ASHGs. The NP-completeness results concern deciding on the existence of a stable partition. Membership in Function-P means that the search problem of constructing a stable partition can be solved in polynomial time.

ASHG	Unrestricted	Friends-and-enemies games	Appreciation-of-friends games
NS	NP-complete [29]	NP-complete [10]	NP-complete [10]
IS	NP-complete [29]	Function-P [10]	Function-P [16]
CNS	NP-complete (Th. 1)	Function-P [10]	Function-P [10]
MIS	NP-complete [10]	NP-complete (Th. 9)	Function-P [10]
MOS	NP-complete [10]	NP-complete (Th. 7)	NP-complete (Th. 4)

References

1 H. Aziz and F. Brandl. Existence of stability in hedonic coalition formation games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 763–770, 2012.

2 H. Aziz, F. Brandl, F. Brandt, P. Harrenstein, M. Olsen, and D. Peters. Fractional hedonic games. *ACM Transactions on Economics and Computation*, 7(2):1–29, 2019.

3 H. Aziz, F. Brandt, and H. G. Seedig. Computing desirable partitions in additively separable hedonic games. *Artificial Intelligence*, 195:316–334, 2013.

4 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.

5 C. Ballester. NP-completeness in hedonic games. *Games and Economic Behavior*, 49(1):1–30, 2004.

6 S. Banerjee, H. Konishi, and T. Sönmez. Core in a simple coalition formation game. *Social Choice and Welfare*, 18:135–153, 2001.

7 V. Bilò, A. Fanelli, M. Flammini, G. Monaco, and L. Moscardelli. Nash stable outcomes in fractional hedonic games: Existence, efficiency and computation. *Journal of Artificial Intelligence Research*, 62:315–371, 2018.

8 A. Bogomolnaia and M. O. Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38(2):201–230, 2002.

9 F. Brandt and M. Bullinger. Finding and recognizing popular coalition structures. *Journal of Artificial Intelligence Research*, 74:569–626, 2022.

10 F. Brandt, M. Bullinger, and L. Tappe. Single-agent dynamics in additively separable hedonic games. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, 2022. Forthcoming.

11 F. Brandt, M. Bullinger, and A. Wilczynski. Reaching individually stable coalition structures in hedonic games. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 5211–5218, 2021.

12 M. Bullinger. Pareto-optimality in cardinal hedonic games. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 213–221, 2020.

13 M. Bullinger and S. Kober. Loyalty in cardinal hedonic games. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 66–72, 2021.

14 R. Carosi, G. Monaco, and L. Moscardelli. Local core stability in simple symmetric fractional hedonic games. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 574–582, 2019.

15 K. Cechlárová and A. Romero-Medina. Stability in coalition formation games. *International Journal of Game Theory*, 29:487–494, 2001.

16 D. Dimitrov, P. Borm, R. Hendrickx, and S. C. Sung. Simple priorities and core stability in hedonic games. *Social Choice and Welfare*, 26(2):421–433, 2006.

17 D. Dimitrov and S. C. Sung. On top responsiveness and strict core stability. *Journal of Mathematical Economics*, 43(2):130–134, 2007.

18 J. H. Drèze and J. Greenberg. Hedonic coalitions: Optimality and stability. *Econometrica*, 48(4):987–1003, 1980.

19 E. Elkind, A. Fanelli, and M. Flammini. Price of pareto optimality in hedonic games. *Artificial Intelligence*, 288:103357, 2020.

20 E. Elkind and M. Wooldridge. Hedonic coalition nets. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 417–424, 2009.

21 M. Gairing and R. Savani. Computing stable outcomes in symmetric additively separable hedonic games. *Mathematics of Operations Research*, 44(3):1101–1121, 2019.

22 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

23 M. Hoefer, D. Vaz, and L. Wagner. Dynamics in matching and coalition formation games with structural constraints. *Artificial Intelligence*, 262:222–247, 2018.

24 A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

25 R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

26 M. Olsen. On defining and computing communities. In *Proceedings of the 18th Computing: Australasian Theory Symposium (CATS)*, volume 128 of *Conferences in Research and Practice in Information Technology (CRPIT)*, pages 97–102, 2012.

27 W. Suksompong. Individual and group stability in neutral restrictions of hedonic games. *Mathematical Social Sciences*, 78:1–5, 2015.

28 S. C. Sung and D. Dimitrov. On myopic stability concepts for hedonic games. *Theory and Decision*, 62(1):31–45, 2007.

29 S. C. Sung and D. Dimitrov. Computational complexity in additive hedonic games. *European Journal of Operational Research*, 203(3):635–639, 2010.

Bounded Degree Nonnegative Counting CSP

Jin-Yi Cai
Department of Computer Sciences, University of Wisconsin-Madison, WI, USA

Daniel P. Szabo
Department of Computer Sciences, University of Wisconsin-Madison, WI, USA

Abstract

Constraint satisfaction problems (CSP) encompass an enormous variety of computational problems. In particular, all partition functions from statistical physics, such as spin systems, are special cases of counting CSP (#CSP). We prove a complete complexity classification for every counting problem in #CSP with nonnegative valued constraint functions that is valid when every variable occurs a bounded number of times in all constraints. We show that, depending on the set of constraint functions $\mathcal{F}$, every problem in the complexity class #CSP($\mathcal{F}$) defined by $\mathcal{F}$ is *either* polynomial time computable for all instances without the bounded occurrence restriction, *or* is #P-hard even when restricted to bounded degree input instances. The constant bound in the degree depends on $\mathcal{F}$. The dichotomy criterion on $\mathcal{F}$ is decidable. As a second contribution, we prove a slightly modified but more streamlined decision procedure (from [15]) for tractability. This enables us to fully classify a family of *directed* weighted graph homomorphism problems. This family contains both P-time tractable problems and #P-hard problems. To our best knowledge, this is the first family of such problems explicitly classified that are not *acyclic*, thereby the Lovász-goodness criterion of Dyer-Goldberg-Paterson [24] cannot be applied.

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

Keywords and phrases Computational Counting Complexity, Constraint Satisfaction Problems, Counting CSPs, Complexity Dichotomy, Nonnegative Counting CSP, Graph Homomorphisms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.27

Funding *Jin-Yi Cai*: Supported by NSF CCF-1714275.
Daniel P. Szabo: Supported by an REU supplement of NSF CCF-1714275.

Acknowledgements The authors thank the three anonymous reviewers for their valuable comments and suggestions.

1 Introduction

Constraint Satisfaction Problems (CSPs) have been a subject of immense interest due to their wide applicability and intrinsic elegance. In particular, counting CSPs, or #CSPs, have been an active subject in computational counting complexity [18, 19, 10, 9, 22, 7, 15, 13], including their approximate solutions [28, 30, 20, 42, 41]. Roughly speaking, an (unweighted) constraint satisfaction problem deals with the following scenario, where there is a set of variables, each taking values over some finite domain D, and a set of constraints, each applied on an (ordered) subsequence of these variables. The #CSP problem on an instance asks how many assignments there are of these variables that satisfy all of the given constraints.

Applications of CSP problems are wide-ranging and varied. They range from within computer science to physical sciences such as physics, chemistry, engineering, even music [52, 1, 38, 47]. Within computer science, belief propagation has been a popular research topic in AI, which are ultimately based on some forms of partition function evaluations [5, 39, 40, 46, 29, 48, 51]. The term partition function, which we define formally later, arises from statistical physics, where one can see special cases of (weighted) counting CSPs in the

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 27; pp. 27:1–27:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

form of spin systems such as the Ising and Potts models, e.g. [25]. As is the case in physical sciences as well as in applications within computer science, the instances of counting CSP problems that occur in practice are often with the additional restriction that variables occur a bounded number of times.

To define (unweighted) #CSP problems formally, let D be a finite domain set, Γ be a set of constraint relations Θ_i, where each Θ_i is a relation on D of arity $r_i = r(\Theta_i) \geq 1$. An instance of #CSP(Γ) is then defined by a set X of n variables over D, and a list of constraints Θ from Γ, and for each constraint Θ in the list a sequence of $r(\Theta)$ variables from X that the constraint is applied to. This defines an n-ary relation R in D^n on the input variables where an assignment $(x_1, \ldots, x_n) \in D^n$ is in R iff all constraints are satisfied. For any fixed Γ, the counting CSP problem #CSP(Γ) consists of all input instances using constraint relations from Γ. The computational problem is to compute the size of R given an arbitrary input instance, where the (worst case) computational complexity is measured in terms of size n of the set of variables and the size of the list of constraints. For a finite (fixed) Γ, this can be simplified to just n, up to a polynomial factor. A complexity dichotomy theorem can classify, depending on Γ, the problem #CSP(Γ) as either computable in polynomial time (P-time), or #P-complete, with no intermediate cases. Typically, the set Γ is a fixed finite set, which defines the #CSP problem – this Γ is the name of the problem. However, in most dichotomy theorems one can allow infinite sets, where in the P-time computable case we assume the specification of the constraints in the instances counts toward the input size, and in the #P-complete case there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that #CSP(Γ_0) is #P-hard.

For example, if we let $D = \{0, 1\}$ and $\Gamma = \{\mathrm{OR}_k | k \geq 1\} \cup \{\neq_2\}$, where OR_k is the k-ary OR function, and $\neq_2$ the binary disequality function, then the problem #CSP(Γ) is equivalent to #SAT, the counting Boolean satisfiability problem.

This formulation can be generalized to the weighted setting. In the most general case, the constraint functions can take real or complex values. In this paper we only consider #CSP defined by nonnegatively weighted constraint functions. This means that we replace the constraint language Γ by a set of constraint functions $\mathcal{F}$, where each $f_i \in \mathcal{F}$ has some arity $r_i \geq 1$ and maps D^{r_i} to nonnegative algebraic reals, denoted as $\mathbb{R}_+$.[1] Any given instance I defines a function $F_I : D^n \to \mathbb{R}_+$, such that on each assignment of variables, F_I takes the value the product over the constraint functions in I evaluated on the assignment. The solution to this instance I of #CSP($\mathcal{F}$) is then

$$Z_{\mathcal{F}}(I) = \sum_{(x_1, \ldots, x_n) \in D^n} F_I(x_1, \ldots, x_n). \tag{1}$$

This sum-of-products expression in (1) is called the partition function for an instance of #CSP, with the terminology coming from statistical physics [3]. When all functions in $\mathcal{F}$ are 0-1 valued, then the product is also 0-1 valued and is equivalent to the logical AND, and the partition function counts the number of satisfying assignments. Thus this $Z_{\mathcal{F}}(I)$ generalizes the unweighted case when $\mathcal{F}$ is a set of constraint relations Γ.

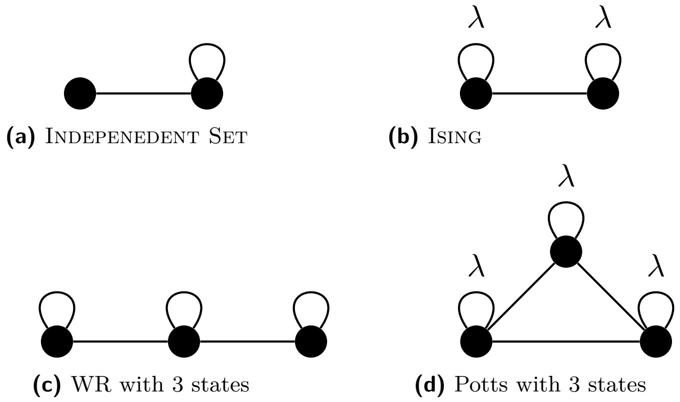
As a special case of #CSP, a q-state spin system is a problem on a domain $[q]$ with the constraint language having only a single binary constraint defined by the $q \times q$ interaction matrix A. An instance to this problem is a graph $G = (V, E)$, where the vertices (sites) are considered to be variables (spins) and the edges (bonds) correspond to the interactions between these vertices. The famous Ising model with parameter λ has domain size $q = 2$,

[1] Restricting to algebraic numbers is standard in this research area because we wish to state our results in the Turing machine model for strict bit complexity. See [14].

and is defined by its interaction matrix $A^{\lambda}_{\text{Ising}} = \begin{bmatrix} \lambda & 1 \\ 1 & \lambda \end{bmatrix}$ (see Figure 1b). The Potts model (Figure 1d) and Widom-Rowlinson model (Figure 1c) on 3 states are defined by the following interaction matrices respectively,

$$A^{\lambda}_{\text{3Potts}} = \begin{bmatrix} \lambda & 1 & 1 \\ 1 & \lambda & 1 \\ 1 & 1 & \lambda \end{bmatrix} \quad \text{and} \quad A_{\text{WR}} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Familiar problems in computer science can also be expressed in this model; e.g., independent set (IS) is defined by $A_{\text{IS}} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ (Figure 1a).



(a) Indepenedent Set **(b)** Ising

(c) WR with 3 states **(d)** Potts with 3 states

Figure 1 The graphs corresponding to some well known spin systems.

Bulatov [9] proved a sweeping complexity dichotomy for unweighted #CSPs in, which used deep results from universal algebra. His dichotomy theorem states that #CSP(Γ) is solvable in polynomial time if Γ satisfies a condition called congruence singularity; it is #P-complete otherwise. Dyer and Richerby [22] gave another proof of this dichotomy using a new P-time tractability criterion, which they proved to be equivalent to congruence singularity.

A nonnegative matrix is block-rank-1 if it becomes a block-diagonal matrix after a permutation of its rows and a permutation of its columns separately, such that all blocks are rank 1 except for possibly one all-zero block. (Here the blocks in the block-diagonal form of the matrix need not be square matrices.) For example, the following matrix (where blank entries are 0's)

$$\begin{bmatrix}
A_{0,0} & & A_{0,2} & & & & & \\
A_{1,0} & & A_{1,2} & & & & & \\
& & & & A_{2,4} & & A_{2,6} & \\
& & & & A_{3,4} & & A_{3,6} & \\
& A_{4,1} & & A_{4,3} & & & & \\
& A_{5,1} & & A_{5,3} & & & & \\
& & & & & A_{6,5} & & A_{6,7} \\
& & & & & A_{7,5} & & A_{7,7}
\end{bmatrix} \tag{2}$$

is block-rank-1 if each nonzero rectangle of the form $\begin{bmatrix} A_{i,j} & A_{i,j'} \\ A_{i',j} & A_{i',j'} \end{bmatrix}$ has rank 1.

For unweighted #CSP, the Dyer-Richerby condition in [22] for polynomial time tractability in the dichotomy theorem is *Strong Balance*. Let $d = |D|$ be the domain size. We say a constraint language Γ is *Strongly Balanced* if every n-ary relation R defined by an instance of #CSP(Γ) satisfies the following condition:

For any $a, b \geq 1$ and $c \geq 0$ with $a + b + c \leq n$, the following $d^a \times d^b$ matrix M is block-rank-1:

$$M(\boldsymbol{u}, \boldsymbol{v}) = \left|\{\boldsymbol{w} \in D^c : \exists \boldsymbol{z} \in D^{n-c-b-a} \text{ s.t. } (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{z}) \in R\}\right| .$$

(If $a + b + c = n$, then the quantified statement "$\exists \boldsymbol{z} \in D^{n-c-b-a}$ such that $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{z}) \in R$" simply means that $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \in R$.)

If we are dealing with $\mathcal{F}$ rather than Γ, and if $\mathcal{F}$ is not a set of 0-1 valued functions, then the existential quantified statement "$\exists \boldsymbol{z}$" has no meaning. It turns out that there are several equivalent notions of *Balance*, which when $\mathcal{F}$ is restricted to a set of 0-1 valued functions (i.e. when $\mathcal{F}$ can be identified with a constraint language Γ) are all equivalent to the notion of Strong Balance; see Lemma 9.4 in [15]. These notions of Balance do not use existential quantifiers (see Definition 2 in Section 2). These notions are central to the #CSP dichotomies for #CSP($\mathcal{F}$) for nonnegative valued $\mathcal{F}$.

The study of #CSPs is closely related to that of counting graph homomorphisms [35, 27, 4, 36]. For two graphs G and H, a graph homomorphism from G to H is a mapping $f : V(G) \to V(H)$ that preserves vertex adjacency. In other words, if $e = \{u, v\} \in E(G)$ then $e' = \{f(u), f(v)\} \in E(H)$, for all edges e in G. The question of interest in counting complexity is the number of graph homomorphisms from one graph to another, which can also be represented by a partition function. If we let A be the $m \times m$ adjacency 0-1 matrix of the graph H, then the number of homomorphisms from G to H can be represented as a sum-of-products partition function as follows,

$$Z_A(G) = \sum_{f:V(G)\to[m]} \prod_{\{u,v\}\in E(G)} A_{f(u),f(v)}.$$

Partition functions of graph homomorphism can represent important physical spin systems such as the Ising, Potts, or Widom-Rowlinson models, as well as many other well known problems in computer science.

Counting graph homomorphisms is a special case of #CSP. In fact, the vertex-edge incidence graph of G defines an input to a #CSP problem, where vertices $V(G)$ are variables and edges $E(G)$ are (applications of binary) constraints, and the constraint language consists of a single binary relation represented by the adjacency matrix A defining the graph homomorphism problem $G \mapsto Z_A(G)$. Just as in #CSPs, the counting graph homomorphism function $Z_A(G)$ can be generalized from the 0-1 unweighted case to the weighted case where A is a real or complex matrix. It is symmetric for an undirected graph H, in which case we also only consider undirected G; for directed graph homomorphisms, A need not be symmetric.

The first dichotomy on counting graph homomorphisms was due to Dyer and Greenhill [21] for undirected graphs. They showed that there is a simple criterion such that if A satisfies the criterion then $G \mapsto Z_A(G)$ is computable in P-time, otherwise it is #P-complete. In fact they proved that if A does not satisfy the criterion then the problem of evaluating $Z_A(G)$ remains #P-complete even when restricted to graphs G with bounded degree Δ, for some Δ depending on A. Computing $G \mapsto Z_A(G)$ when restricted to graphs G with bounded degree Δ is called $\text{EVAL}^{(\Delta)}(A)$. The Dyer-Greenhill dichotomy was extended to the nonnegatively weighted case by Bulatov and Grohe in [8]. This dichotomy was then referenced throughout the field, as many other discoveries, including the results on #CSPs, ended up applying it. However, the hardness part of the proof of the Bulatov-Grohe dichotomy theorem required input graphs that have unbounded degrees. When restricted to bounded degree graphs, the worst case complexity of the Bulatov-Grohe dichotomy was left open for 15 years, until it was finally resolved by Govorov, Cai, and Dyer in [26] for graph homomorphisms with nonnegative weights, and continued by Cai and Govorov in [16] for complex weights. Most problems in statistical physics [6, 11, 31] use bounded degree graphs, and also most of the approximation algorithms work on bounded degree graphs [2, 23, 32, 33, 37, 43, 44, 45, 49]. Over the Boolean domain where variables take 0-1 values, it is known that the #CSP dichotomy for complex valued constraint functions holds for input instances where each variable occurs at most three times [17].

It has been an open problem to extend the general domain #CSP dichotomies to include the bounded degree case, i.e. where each variable occurs a bounded number of times. It was open even for the 0-1 unweighted case. For the nonnegative cases, this would be the analogous Govorov-Cai-Dyer extension [26] of the Bulatov-Grohe dichotomy for graph homomorphism, but apply to a much broader class of problems, as graph homomorphism is the special case of $\#\mathrm{CSP}(\mathcal{F})$ where $\mathcal{F}$ consists of a single binary function.

In this paper we prove such a dichotomy for bounded degree nonnegative #CSPs. For any finite domain D, any finite set of nonnegative constraint functions $\mathcal{F}$ on D, and any integer $\Delta \geq 0$, we define $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$ to be the #CSP problem, where the input consists of n variables $x_1, \ldots, x_n$ over D and a sequence of constraint functions $f_1, \ldots, f_m \in \mathcal{F}$ each applied to a subsequence of the n variables, where each variable x_i appears no more than Δ times among $f_1, \ldots, f_m$. Note that in general, a function $f \in \mathcal{F}$ may occur multiple times among $f_1, \ldots, f_m$. We take $n + m$ as the input size. We prove that the same dichotomy criterion in [15] applies to the bounded degree case: if the P-time tractability criterion is not satisfied, then $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$ remains #P-hard for some $\Delta > 0$. The dichotomy criterion of [15] will be explained in more detail after we introduce some more definitions in Section 2. These notions are further explicated in Theorem 4, and a more technical statement of Theorem 1 is given in Theorem 6.

▶ **Theorem 1.** *For any finite domain D and any nonnegatively weighted constraint functions $\mathcal{F}$ on D, if $\mathcal{F}$ satisfies the tractability criterion in [15], then $\#\mathrm{CSP}(\mathcal{F})$ is P-time computable; otherwise, $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$ is #P-hard [2] for some $\Delta > 0$.*

For any fixed finite set $\mathcal{F}$ of constraint functions, the arities of $f \in \mathcal{F}$ are bounded. Viewing any instance as a bipartite graph, with the variables $x_1, \ldots, x_n$ on one side and constraints $f_1, \ldots, f_m \in \mathcal{F}$ on the other, with an edge between x_i and f_j if x_i is an input to the function f_j, we can see that the condition for a #CSP instance to be bounded degree corresponds exactly to this bipartite graph having bounded degree.

Our second contribution in this paper is a slightly modified but more streamlined decision procedure (compared to that of [15]) for polynomial time tractability. This enables us to fully classify a family of *directed* weighted graph homomorphism problems. This family contains both P-time tractable problems and #P-hard problems. To our best knowledge, this is the first family of such problems explicitly classified that are not *acyclic*, thereby the Lovász-goodness criterion of Dyer-Goldberg-Paterson [24] cannot be applied.

2 Balance

Several variants of the *Balance* condition have been used in the study of counting constraint satisfaction problems. In addition to the *Strong Balance* condition [22], the following conditions have been introduced in [15]. Recall that $d = |D|$ denotes the domain size.

[2] The problem $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$ is also no harder than #P under a polynomial-time Turing reduction for any $\mathcal{F}$. The statement for Theorem 1 does not state #P-complete only for the technical reason that by definition functions in #P take nonnegative integer values while the partition function in (1) need not.

▶ **Definition 2** (Various notions of Balance). *We have the following notions:*

1. *(Balance) We say $\mathcal{F}$ is* Balanced *if for any $n \geq 2$, any $a \geq 1$ and $b \geq 1$ with $a + b \leq n$, and any instance I of #CSP($\mathcal{F}$) which defines an n-ary function $F_I(x_1, \dots, x_n)$ over D^n, the following $d^a \times d^b$ matrix M_I is block-rank-1: The rows and columns of M_I are indexed by tuples $\boldsymbol{u} \in D^a$ and $\boldsymbol{v} \in D^b$ respectively, and*

$$M_I(\boldsymbol{u}, \boldsymbol{v}) = \sum_{\boldsymbol{w} \in D^{n-a-b}} F_I(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}),$$

for all $\boldsymbol{u} \in D^a, \boldsymbol{v} \in D^b$. If $a + b = n$ then the sum $\sum_{\boldsymbol{w} \in D^{n-a-b}} F_I(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ is simply $F_I(\boldsymbol{u}, \boldsymbol{v})$.

2. *(Weak Balance) We say $\mathcal{F}$ is* Weakly Balanced *if the definition for Balance holds for $b = 1$.*
3. *(Primitive Balance) We say $\mathcal{F}$ is* Primitively Balanced *if the definition for Balance holds for $a = b = 1$.*

While these three notions may seem to have varying strengths, all three are in fact equivalent by combining the proof in [15] and [34]. See Theorem 4 below. We need the following definition.

▶ **Definition 3** (Strong Rectangularity). *We say a matrix M is* Rectangular *if after a row permutation and a column permutation it is a block diagonal matrix where all diagonal blocks have no zero entries, except possibly one all zero block. We say a constraint language Γ over D is* Strongly Rectangular *if for any input instance I of #CSP(Γ) which defines an n-ary relation R_I over D^n and for any a and b such that $1 \leq a < b \leq n$, the following $|D|^a \times |D|^{b-a}$ matrix M is rectangular: The rows of M are indexed by $\boldsymbol{u} \in D^a$, the columns of M are indexed by $\boldsymbol{v} \in D^{b-a}$, and*

$$M(\boldsymbol{u}, \boldsymbol{v}) = \left|\{\boldsymbol{w} \in D^{n-b} : (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \in R_I\}\right|.$$

▶ **Theorem 4.** *The notions of Balance, Weak Balance, and Primitive Balance are equivalent, and can be taken as the P-time tractability criterion of the dichotomy in [15].*

Proof. For any $\mathcal{F}$ of nonnegative valued constraint functions, Cai, Chen and Lu proved in [15] that (1) if $\mathcal{F}$ is Balanced then it is Weakly Balanced and that the support constraint language of $\mathcal{F}$ satisfies Strong Rectangularity, and (2) the latter two conditions imply that #CSP($\mathcal{F}$) is P-time computable. Here the support constraint language of $\mathcal{F}$ is obtained by taking the support set of each function in $\mathcal{F}$. On the other hand they also proved that if $\mathcal{F}$ is not Balanced then #CSP($\mathcal{F}$) is #P-hard. Thus their dichotomy criterion is that $\mathcal{F}$ is Balanced. They also proved in [15] that Primitive Balance implies Weak Balance. Lin and Wang proved in [34] that Weak Balance implies Balance, thus unifying all three notions. ◀

3 Bounded Degree #CSPs

▶ **Lemma 5.** *Let M be a nonnegative matrix. If M is not block-rank-1 then neither is MM^T.*

Proof. If every two rows of M are either proportional or their nonzero entries are on disjoint subsets of columns, then M would be block-rank-1. Thus there are rows M_i and M_j, such that they are linearly independent, and the subsets of columns where their nonzero entries occur intersect. Being nonnegative, the latter condition implies that they are not orthogonal. So by the Cauchy-Schwarz inequality we have

$$0 < (M_i \cdot M_j)^2 < (M_i \cdot M_i)(M_j \cdot M_j).$$

Letting $A = MM^T$ we find four nonzero elements $A_{i,i}, A_{i,j} = A_{j,i}$, and $A_{j,j}$ satisfying $A_{i,i}A_{j,j} > A_{i,j}^2 > 0$, so A is not block-rank-1. ◀

We can now prove our main result, i.e., if a nonnegative constraint set $\mathcal{F}$ does not satisfy the Balance condition, then $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$ is #P-hard for some $\Delta > 0$.

▶ **Theorem 6.** *If $\mathcal{F}$ is Primitively Balanced, then the problem $\#\mathrm{CSP}(\mathcal{F})$ without degree restriction is computable in polynomial time, otherwise $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$ is #P-hard for some $\Delta > 0$.*

Proof. We only need to prove the hardness part. Let $\mathcal{F}$ be a finite set of nonnegatively weighted constraint functions that is not Primitively Balanced. Then for some instance I on n variables, the $|D| \times |D|$ matrix M defined by

$$M(x_1, x_2) = \sum_{(x_3,\ldots,x_n)\in D^{n-2}} F_I(x_1, x_2, x_3, \ldots, x_n)$$

is not block-rank 1. If we let $A = MM^T$, then A is symmetric, nonnegative, and not block-rank 1 by Lemma 5. This A defines a graph homomorphism problem. We know from [26] that the bounded degree nonnegative graph homomorphism problem $\mathrm{EVAL}^{(\Delta)}(A)$ is #P-hard for some $\Delta > 0$, where the constant Δ depends on A. Here we show a reduction $\mathrm{EVAL}^{(\Delta)}(A) \le_P \#\mathrm{CSP}^{(\Delta')}(\mathcal{F})$, for some $\Delta' > 0$, thereby showing that $\#\mathrm{CSP}^{(\Delta')}(\mathcal{F})$ is #P-hard for some $\Delta' > 0$.

To show that, consider graphs G with maximum degree at most Δ as input instances of $\mathrm{EVAL}^{(\Delta)}(A)$. We can compute the value $Z_A(G)$ by expressing it as the partition function $Z_{\mathcal{F}}(I(G))$ for some instance $I(G)$ of polynomial size in $\#\mathrm{CSP}^{(\Delta')}(\mathcal{F})$. We will use the instance I that defines the matrix M as having constant size, as it does not depend on G. We construct $I(G)$, an input to $\#\mathrm{CSP}(\mathcal{F})$, with the additional property that every variable occurs at most Δ' times, such that $Z_{\mathcal{F}}(I(G)) = Z_A(G)$, as follows.

We note that each entry in A is a dot product of two row vectors in M, and every entry of M is a sum over $|D|^{n-2}$ evaluations of F_I.

We will define a (binary) gadget, which is an instance of $\#\mathrm{CSP}(\mathcal{F})$ of bounded size, with two specially labelled variables called x^* and x^{**}. Copies of this gadget will be used in the construction of (global) $\#\mathrm{CSP}(\mathcal{F})$ instances. A (binary) gadget may have other variables, but in the global $\#\mathrm{CSP}(\mathcal{F})$ instances all constraints applied to the variables other than x^* and x^{**} in each copy are from within the gadget. We define $I(G)$ by replacing every edge in G by a copy of this gadget. Formally, the construction is as follows, where the gadget simulates the edge weights in A in the #CSP setting.

1. Define a variable x_v over D for every $v \in V(G)$.
2. For each $e = uv \in E(G)$ we add $2n-3$ variables y_e, $\boldsymbol{z}_e = (z_{e,3}, \ldots, z_{e,n})$, and $\boldsymbol{z}'_e = (z'_{e,3}, \ldots, z'_{e,n})$ over the same domain D. We then apply two copies of I as constraints, one copy over the variables $(x_u, y_e, \boldsymbol{z}_e)$ and another over $(x_v, y_e, \boldsymbol{z}'_e)$. There are no other constraints applied on the variables y_e, $\boldsymbol{z}_e$ and $\boldsymbol{z}'_e$.

Thus one copy of the binary gadget is used to replace each edge $uv \in E(G)$, with the two specially labelled variables x^* and x^{**} identified with x_u and x_v. This gadget defines the following constraint function with input variables x_u and x_v taking values in D,

$$\begin{aligned}
&\sum_{y_e\in D}\left(\sum_{\boldsymbol{z}_e\in D^{n-2}} F_I(x_u, y_e, \boldsymbol{z}_e) \sum_{\boldsymbol{z}'_e\in D^{n-2}} F_I(x_v, y_e, \boldsymbol{z}'_e)\right)\\
&= \sum_{y_e\in D} M_I(x_u, y_e) M_I(x_v, y_e)\\
&= (MM^T)(x_u, x_v)\\
&= A(x_u, x_v).
\end{aligned}$$

Therefore the gadget defines the edge constraint function represented by the matrix A, which is exactly the edge weights in $Z_A(G)$.

Since G has bounded degrees, and I also has constant size, $I(G)$ also has bounded degrees. The variables y_e are "local" to each edge $e \in E(G)$ in the sense that there are no other constraints on them except in the definition of the gadget for this edge e, which has constant size. The same is true for $\boldsymbol{z}_e$ and $\boldsymbol{z}'_e$.

Also, the size of $I(G)$ is linear in the size of G, so this is a polynomial time reduction. That $Z_{\mathcal{F}}(I(G)) = Z_A(G)$ follows from the fact that A is the edge constraint function by our construction. ◀

The constant Δ in Theorem 1 depends on $\mathcal{F}$. In [21], Dyer and Greenhill conjectured that a universal constant $\Delta = 3$ suffices for $\mathrm{EVAL}^{(\Delta)}(A)$ where A is a 0-1 symmetric matrix. This is still open. It is open whether a universal constant Δ, or a constant that only depends on the domain size $|D|$, may suffice for even the 0-1 case, for both $\mathrm{EVAL}^{(\Delta)}(A)$ and $\#\mathrm{CSP}^{(\Delta)}(\mathcal{F})$. In [17] it is known that the constant 3 suffices for the Boolean domain. Xia [50] proved that a universal Δ *does not* exist for $\mathrm{EVAL}^{(\Delta)}(A)$ for complex symmetric matrices A, assuming #P does not collapse to P.

4 Effective Dichotomy and a Family of Directed GH

The condition of Balance (Definition 2) in the dichotomy refers to all instances I of $\#\mathrm{CSP}(\mathcal{F})$, which is an infinitary statement. Thus it is not immediate that the tractability condition in Theorem 6 is decidable. However the condition is the same as the one in [15] for the unbounded degree case, and in that paper a decision procedure is given. Here we give a slight modification of the same decision procedure for the dichotomy in Theorem 6. This form is more symmetric and allows us to apply the procedure more effectively.

▶ **Theorem 7.** *The polynomial-time tractability condition of balance in Theorem 6 can be tested by the following two conditions. Measured in the size of D and $\mathcal{F}$, this shows that the decision problem for testing balance is in* NP*.*

(A) *There is a Mal'tsev polymorphism $\varphi : D^3 \to D$ for (the support of) every function in $\mathcal{F}$. This means that φ preserves all relations defined as the support of some function in $\mathcal{F}$ (this is called a polymorphism), and satisfies $\varphi(a,a,b) = \varphi(b,a,a) = b$ for all $a, b \in D$ (Mal'tsev property). The existence of such a mapping φ is equivalent to Strong Rectangularity.*

(B) *For all $\alpha \neq \beta, \kappa \neq \lambda \in D$ there is a bijection $\pi : D^6 \to D^6$ satisfying the following three properties:*

1. $\pi((\alpha,\alpha,\alpha,\beta,\beta,\beta)) = (\alpha,\alpha,\alpha,\beta,\beta,\beta)$.
2. $\pi((\kappa,\lambda,\kappa,\lambda,\kappa,\lambda)) = (\lambda,\kappa,\lambda,\kappa,\lambda,\kappa)$.
3. *Any function $f \in \mathcal{F}$ with arity r is invariant under π, that is, for any sequence $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_r)$ of length r of 6-tuples where $\boldsymbol{y}_i = (y_{i,1}, \ldots, y_{i,6}) \in D^6$ for $1 \le i \le r$, the following holds:*

$$\prod_{j \in [6]} f(y_{1,j}, \ldots, y_{r,j}) = \prod_{j \in [6]} f(\pi(\boldsymbol{y}_1)_j, \ldots, \pi(\boldsymbol{y}_r)_j). \tag{3}$$

The only difference in the statement of this decision criterion compared to the one stated in [15] is a more symmetric expression for condition *B2* of (B). We omit the proof here as it closely follows the proof in [15][3] However, this more symmetric form makes the criterion

[3] Alternatively, one can apply a perm $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 2 & 4 & 6 & 5 \end{pmatrix}$ which is an automorphism of the relational structure $(\mathfrak{D}, \mathfrak{F})$, the 6th power of $\#\mathrm{CSP}(\mathcal{F})$, to derive this form of the decision criterion from

more easily applicable. We demonstrate this by proving new tractable and intractable cases of directed graph homomorphisms that were previously unknown. The tractability or intractability of these problems were decidable in principle by previous methods; however, the decision procedure of the previous method was in practice too complicated to be useful.

Dyer, Goldberg and Paterson in [24] proved a decidable complexity dichotomy for (unweighted) directed graph homomorphisms that is restricted to directed *acyclic* graphs. Their polynomial-time tractability criterion is an interesting condition of being *layered* and *Lovász-good* for directed acyclic graphs. They state in [24] that "An interesting feature of the dichotomy, which is absent from previously-known dichotomy results, is that there is a rich supply of tractable graphs H with complex structure". Going beyond directed *acyclic* graphs, as it is done with Theorem 6 and 7, is expected to yield even more polynomial-time tractable problems. However, up until now we don't have any interesting concrete examples. (Part of the reason is probably that testing for the tractability criterion, while decidable, is not a simple matter; see below.) The dichotomy theorem in this paper applies more generally without the *acyclicity* restriction. We now give a family of non-acyclic directed graph homomorphism problems that we can completely classify using our tractability criterion in Theorem 7. To our best knowledge, this is the first such explicit family that can be classified, going beyond the Lovász-goodness criterion [24].

To start, if we take all nonzero $A_{i,j} = 1$ in equation (2), we get a binary relation that defines a polynomial-time tractable problem. This represents an adjacency matrix of a directed graph H illustrated in Figure 2. In fact, we can give an infinite family of tractable #CSP based on a weighted binary constraint function given in equation (4). These problems were considered in [12], where it was shown that, while the complexity of the #CSP defined by these relations is provably decidable, the decision criterion was yet too complicated, therefore for what values of $A_{i,j}$ in equation (2) the problem it defines is tractable for #CSP was not resolved.

We will show that a nonnegative binary constraint function given in the form of equation (2) with positive entries $A_{i,j}$ defines a tractable #CSP iff the constraint function is a positive multiple of the function in equation (4)

$$A = \begin{bmatrix} 1 & & v & & & & & \\ u & & uv & & & & & \\ & & & & y & & yv & \\ & & & & yu & & yuv & \\ & x & & xv & & & & \\ & xu & & xuv & & & & \\ & & & & & z & & zv \\ & & & & & zu & & zuv \end{bmatrix} \tag{4}$$

for some positive reals u, v, x, y, z, with the condition that $z = xy$.

The #CSP problem it defines is on a domain of size 8 and has a constraint function set $\mathcal{F}$ consisting of a single binary (but not symmetric) constraint function given by the matrix in (4). The directed graph defined by the support of the function is not acyclic.

We first prove the relation in (4) for any positive u, v, x, y and $z = xy$ defines a tractable problem. After that we prove the reverse direction.

To apply Theorem 7, we treat $D = \{0, \ldots, 7\}$ as a vector space $(\mathrm{GF}[2])^3$ of size 8, represented by three bit strings $\{0,1\}^3$. Then we can take the Mal'tsev polymorphism $\varphi : D^3 \to D$ where $\varphi(x, y, z) = x - y + z$. (Here $-$ is the same as $+$ in $\mathrm{GF}[2^3]$.) It is Mal'tsev because $\varphi(a, a, b) = \varphi(b, a, a) = b$ for all $a, b \in D$. The directed edge relation given by the matrix (2) (by setting all 16 nonzero entries $A_{i,j} = 1$) is given symbolically as follows:

$$A_{i,j} = 1 \text{ where } i = i_1 i_2 i_3, j = j_1 j_2 j_3 \in D \iff j_1 = i_2 \text{ and } j_3 = i_1. \tag{5}$$

the form proved in [15]. For detailed definitions of $(\mathfrak{D}, \mathfrak{F})$ and automorphism of relational structures see page 2190 in [15].

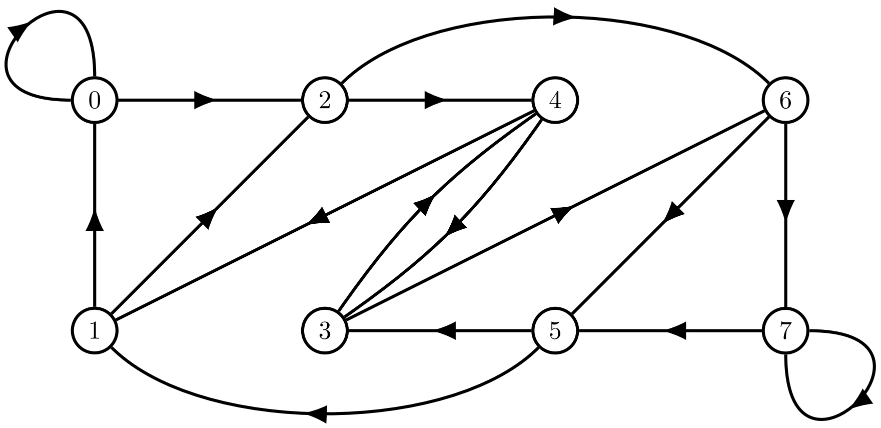


Figure 2 A tractable binary relation represented by a directed graph. The adjacency matrix is given in equation (4).

One can easily check that φ is a polymorphism, i.e. for any (x,x'), (y,y') and (z,z'), if $A_{x,x'} = A_{y,y'} = A_{z,z'} = 1$ then $A_{\varphi(x,y,z),\varphi(x',y',z')} = 1$.

For the second requirement (B), there are $|D|^6! = 262144! > 10^{1306590}$ bijections from D^6 to D^6, so it is infeasible to enumerate them. However the following map π works.

Let $M : \{0,1\}^6 \to \{0,1\}^6$ be the bijection that swaps (010101) and (101010) and acts as the identity on the rest. In particular, M preserves the Hamming weight. Let M_i be the ith output bit of M, and let $x_1, x_2, \ldots, x_6 \in D$ where each x_i consists of three bits $x_i = a_i b_i c_i \in \{0,1\}^3$.

We write $\boldsymbol{x} = (x_1, \ldots, x_6) = (a_1b_1c_1, \ldots, a_6b_6c_6) \in D^6$. We will also represent $\boldsymbol{x}$ bitwise using $\boldsymbol{a} = (a_1, \ldots, a_6) \in \{0,1\}^6$, $\boldsymbol{b} = (b_1, \ldots, b_6) \in \{0,1\}^6$, and $\boldsymbol{c} = (c_1, \ldots, c_6) \in \{0,1\}^6$, and write $\boldsymbol{x} = \boldsymbol{abc}$.

Then we define

$$\begin{aligned} \pi(\boldsymbol{x}) &= \pi(x_1, x_2, \ldots, x_6) = \pi(a_1b_1c_1, a_2b_2c_2, \ldots, a_6b_6c_6) \\ &= (M_1(\boldsymbol{a})M_1(\boldsymbol{b})M_1(\boldsymbol{c}),\ M_2(\boldsymbol{a})M_2(\boldsymbol{b})M_2(\boldsymbol{c}),\ \ldots,\ M_6(\boldsymbol{a})M_6(\boldsymbol{b})M_6(\boldsymbol{c})) \end{aligned}$$

▷ Claim 8. The mapping $\pi : D^6 \to D^6$ satisfies properties B1, B2, and B3 of (B) in Theorem 7 for all $\alpha \neq \beta, \kappa \neq \lambda \in D$. [4]

Proof Sketch. Property B1 holds by construction, because M fixes pointwise 0^6, 1^6, $0^3 1^3$, $1^3 0^3$. It satisfies property B2 because in addition it swaps (010101) and (101010).

For property B3, equation (3) is expressed as (more details are given below)

$$u^{\sum c_i} v^{\sum d_i} x^{\sum a_i} y^{\sum b_i} \left(\frac{z}{xy}\right)^{\sum a_i b_i} = u^{\sum M_i(\boldsymbol{c})} v^{\sum M_i(\boldsymbol{d})} x^{\sum M_i(\boldsymbol{a})} y^{\sum M_i(\boldsymbol{b})} \left(\frac{z}{xy}\right)^{\sum M_i(\boldsymbol{a}) M_i(\boldsymbol{b})} \tag{6}$$

[4] In Theorem 7 the mapping π may depend on $\alpha \neq \beta, \kappa \neq \lambda \in D$, but the π in Claim 8 is in fact the same for all $\alpha, \beta, \kappa, \lambda$.

where all sums (as well as those below) range from $i = 1$ to 6. Because M preserves Hamming weight, we get $\sum_{i=1}^{6} a_i = \sum_{i=1}^{6} M_i(\boldsymbol{a})$, and similarly for $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$, and since $z = xy$, this equation holds. ◀

We now investigate *all* possible tractable cases of A in (2) with positive entries $A_{i,j}$. By the necessary condition of Balance applied to the binary function A itself, all relevant four 2×2 blocks must be of rank 1, in order to be tractable, i.e., it takes the form in (4) with some positive u, v, x, y and z, up to a global positive factor. We prove that, up to a global positive factor, a nonnegative matrix A in (2) with the given support structure defines a tractable partition function $Z_{\mathcal{F}}(\cdot)$ where $\mathcal{F} = \{A\}$ iff A has the form in (4) for some positive reals u, v, x, y and $z = xy$; otherwise $Z_{\mathcal{F}}(\cdot)$ is #P-hard.

This #CSP problem has $\mathcal{F}$ consisting of a single binary (nonsymmetric) constraint function defined by the matrix A. By its support structure and the Mal'tsev polymorphism we already satisfied condition (A) of Theorem 7. So, the problem is tractable if and only if for all $\alpha \neq \beta, \kappa \neq \lambda \in D$, there is a bijection $\pi : D^6 \to D^6$ that satisfies the following three properties:

1. $\pi((\alpha, \alpha, \alpha, \beta, \beta, \beta)) = (\alpha, \alpha, \alpha, \beta, \beta, \beta)$.
2. $\pi((\kappa, \lambda, \kappa, \lambda, \kappa, \lambda)) = (\lambda, \kappa, \lambda, \kappa, \lambda, \kappa)$.
3. For the binary function represented by the matrix A, and any 6-tuples $\boldsymbol{x}, \boldsymbol{y} \in D^6$, where $\boldsymbol{x} = (x_1, \ldots, x_6)$ and $\boldsymbol{y} = (y_1, \ldots, y_6)$, we have the following invariance under π,

$$\prod_{i \in [6]} A_{x_i, y_i} = \prod_{i \in [6]} A_{\pi(\boldsymbol{x})_i, \pi(\boldsymbol{y})_i}. \tag{7}$$

Suppose there is a bijection $\pi : D^6 \to D^6$ that satisfies these properties. Let $\pi(x_1, \ldots, x_6) = (\pi_1(x_1, \ldots, x_6), \ldots, \pi_6(x_1, \ldots, x_6))$, where $\pi_i : D^6 \to D$, and $\pi_i(x_1, \ldots, x_6)$ is the ith output entry in D of π. Denote the three bits of $\pi_i(x_1, \ldots, x_6)$ as $f_i(x_1, \ldots, x_6)$, $g_i(x_1, \ldots, x_6)$, and $h_i(x_1, \ldots, x_6)$. To satisfy property 3, we need each π_i to preserve the edge relation 5, i.e., preserve the support set. Since π is a bijection, if we verify that a nonzero LHS of (7) implies a nonzero RHS of (7), we will also have proved that it maps a zero LHS to a zero RHS; thus it preserves the support set. So, consider arbitrary

$$\boldsymbol{x} = (x_1, \ldots x_6) = (a_1 b_1 c_1, \ldots, a_6 b_6 c_6) \in D^6, \quad \boldsymbol{y} = (y_1, \ldots, y_6) = (b_1 d_1 a_1, \ldots, b_6 d_6 a_6) \in D^6.$$

This is a generic pair of tuples such that $A_{x_i, y_i} \neq 0$, for $1 \leq i \leq 6$. We need $A_{\pi_i(\boldsymbol{x}), \pi_i(\boldsymbol{y})} \neq 0$ for each i. As before we will also represent $\boldsymbol{x}$ bitwise using $\boldsymbol{a} = (a_1, \ldots, a_6) \in \{0,1\}^6$, $\boldsymbol{b} = (b_1, \ldots, b_6) \in \{0,1\}^6$, $\boldsymbol{c} = (c_1, \ldots, c_6) \in \{0,1\}^6$ and $\boldsymbol{d} = (d_1, \ldots, d_6) \in \{0,1\}^6$, and write $\boldsymbol{x} = \boldsymbol{abc}$ and $\boldsymbol{y} = \boldsymbol{bda}$.

Therefore, by the edge relation, we have $f_i(\boldsymbol{abc}) = h_i(\boldsymbol{bda})$. Hence f_i is independent of the third part of the input $\boldsymbol{c}$. Also, $g_i(\boldsymbol{abc}) = f_i(\boldsymbol{bda})$, so f_i is also independent of the second part of the input, and therefore is in fact a function on the first part of the input only. Thus there is a function $f_i' : \{0,1\}^6 \to \{0,1\}$, such that $f_i(\boldsymbol{abc}) = f_i'(\boldsymbol{a})$. Then, from $f_i'(\boldsymbol{a}) = h_i(\boldsymbol{bda})$, we know that h_i is actually a function of its third part of the input only. From $g_i(\boldsymbol{abc}) = f_i'(\boldsymbol{b})$, we know that g_i is a function of its second part of the input only. Thus, there are functions $g_i', h_i' : \{0,1\}^6 \to \{0,1\}$, such that $g_i(\boldsymbol{abc}) = g_i'(\boldsymbol{b})$ and $h_i(\boldsymbol{abc}) = h_i'(\boldsymbol{c})$. Putting these together, we see $f_i'(\boldsymbol{a}) = g_i'(\boldsymbol{a}) = h_i'(\boldsymbol{a})$. Since $\boldsymbol{a} \in \{0,1\}^6$ is arbitrary, we get $f_i' = g_i' = h_i'$. We now rename these as $M_i := f_i' = g_i' = h_i'$. In other words, $\pi : D^6 \to D^6$ has the form $\pi = (\pi_1, \pi_2, \ldots, \pi_6)$ where $\pi_i(\boldsymbol{abc}) = M_i(\boldsymbol{a}) M_i(\boldsymbol{b}) M_i(\boldsymbol{c})$. We will name the mapping $M = (M_1, M_2, \ldots, M_6) : \{0,1\}^6 \to \{0,1\}^6$ with M_i being its ith bit output. Since π is a bijection, so must be M.

Now we pick $\alpha = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \beta = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \kappa = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \lambda = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \in D$. (We write them as column vectors to visually aid the readers.) Then clearly $\alpha \neq \beta, \kappa \neq \lambda$. We have $(\alpha, \alpha, \alpha, \beta, \beta, \beta) = \begin{bmatrix} 0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0 \end{bmatrix}$. For any π defined by a bijection M as above, it satisfies property 1. above iff M pointwise fixes 000000, 000111 and 111000. We also have $(\kappa, \lambda, \kappa, \lambda, \kappa, \lambda) = \begin{bmatrix} 1\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,0\,1 \\ 1\,1\,1\,1\,1\,1 \end{bmatrix}$, and $(\lambda, \kappa, \lambda, \kappa, \lambda, \kappa) = \begin{bmatrix} 0\,1\,0\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0 \\ 1\,1\,1\,1\,1\,1 \end{bmatrix}$. Hence π satisfies property 2. above iff M fixes 111111 and swaps 010101 with 101010. Below we assume M is a bijection that satisfies these properties.

It is easy to verify that for any bijection $M : \{0,1\}^6 \to \{0,1\}^6$, the mapping π defined above preserves the support (defined by nonzero values of the LHS in (7)). Since M is a bijection, in the following we only need to verify that (7) holds for any nonzero LHS of (7) (as any zero LHS automatically has a zero RHS).

Now we show that equation (7) in property 3 is the same as (6).

To see that, take any nonzero of the LHS in (7) with $\boldsymbol{x} = \boldsymbol{abc}$ and $\boldsymbol{y} = \boldsymbol{bda}$, then the LHS is evaluated as

$$\begin{aligned} & \prod_{i \in [6]} (u^{c_i} v^{d_i})^{(1-a_i)(1-b_i)} (y u^{c_i} v^{d_i})^{(1-a_i)b_i} (x u^{c_i} v^{d_i})^{a_i(1-b_i)} (z u^{c_i} v^{d_i})^{a_i b_i} \\ = & \prod_{i \in [6]} u^{c_i} v^{d_i} x^{a_i} y^{b_i} \left(\frac{z}{xy}\right)^{a_i b_i} \\ = & \; u^{\sum c_i} v^{\sum d_i} x^{\sum a_i} y^{\sum b_i} \left(\frac{z}{xy}\right)^{\sum a_i b_i} \end{aligned}$$

The expression for the RHS is nearly identical, with $M_i(\boldsymbol{a})$ substituting a_i, and so on.

Now it is clear that if $z = xy$, then the partition function $Z_{\mathcal{F}}(\cdot)$ is tractable, witnessed by any π defined by a bijection on $\{0,1\}^6$ that preserves Hamming weight, pointwise fixes 000000, 111111, 000111, 111000, and swaps 010101 and 101010.

Next, assume $z \neq xy$. We can multiply both sides of (6) over all 2^6 possible $\boldsymbol{c}$ and all 2^6 possible possible $\boldsymbol{d}$, and using the fact that M is a bijection, to get

$$\begin{aligned} & u^{6 \cdot 2^{11}} v^{6 \cdot 2^{11}} x^{2^{12} \sum a_i} y^{2^{12} \sum b_i} \left(\frac{z}{xy}\right)^{2^{12} \sum a_i b_i} = \\ & \quad u^{6 \cdot 2^{11}} v^{6 \cdot 2^{11}} x^{2^{12} \sum M_i(\boldsymbol{a})} y^{2^{12} \sum M_i(\boldsymbol{b})} \left(\frac{z}{xy}\right)^{2^{12} \sum M_i(\boldsymbol{a}) M_i(\boldsymbol{b})}, \end{aligned}$$

which is equivalent to

$$x^{\sum a_i} y^{\sum b_i} \left(\frac{z}{xy}\right)^{\sum a_i b_i} = x^{\sum M_i(\boldsymbol{a})} y^{\sum M_i(\boldsymbol{b})} \left(\frac{z}{xy}\right)^{\sum M_i(\boldsymbol{a}) M_i(\boldsymbol{b})}. \tag{8}$$

Then multiplying over all 2^6 possible 6-tuples $\boldsymbol{b}$,

$$x^{2^6 \sum a_i} y^{6 \cdot 2^{11}} \left(\frac{z}{xy}\right)^{2^5 \sum a_i} = x^{2^6 \sum M_i(\boldsymbol{a})} y^{6 \cdot 2^{11}} \left(\frac{z}{xy}\right)^{2^5 \sum M_i(\boldsymbol{a})},$$

we get $\left(\frac{xz}{y}\right)^{\sum a_i} = \left(\frac{xz}{y}\right)^{\sum M_i(\boldsymbol{a})}$.

Suppose $xz \neq y$, it follows that $\sum_{i=1}^{6} a_i = \sum_{i=1}^{6} M_i(\boldsymbol{a})$ for any $\boldsymbol{a}$, i.e., M preserves Hamming weight. Then it follows from (6) that

$$\left(\frac{z}{xy}\right)^{\sum a_i b_i} = \left(\frac{z}{xy}\right)^{\sum M_i(\boldsymbol{a}) M_i(\boldsymbol{b})}.$$

But if we take $\boldsymbol{a} = 000111$ and $\boldsymbol{b} = 010101$, we have $M(\boldsymbol{a}) = \boldsymbol{a}$ and $M(\boldsymbol{b}) = 101010$. Then $\sum_{i=1}^{6} a_i b_i = 2$, but $\sum_{i=1}^{6} M_i(\boldsymbol{a}) M_i(\boldsymbol{b}) = 1$. This is a contradiction to (6), since $z \neq xy$. Hence we conclude that the partition function $Z_{\mathcal{F}}(\cdot)$ is #P-hard.

Next, suppose that $xz = y$, then (8) is simplified to

$$x^{\sum a_i} y^{\sum b_i} x^{-2\sum a_i b_i} = x^{\sum M_i(\boldsymbol{a})} y^{\sum M_i(\boldsymbol{b})} x^{-2\sum M_i(\boldsymbol{a}) M_i(\boldsymbol{b})}. \tag{9}$$

Multiplying over all possible $\boldsymbol{a}$, this becomes

$$x^{2^5} y^{2^6 \sum b_i} x^{-2\cdot 2^5 \sum b_i} = x^{2^5} y^{2^6 \sum M_i(\boldsymbol{b})} x^{-2\cdot 2^5 \sum M_i(\boldsymbol{b})}.$$

This simplifies to

$$\left(\frac{y}{x}\right)^{\sum b_i} = \left(\frac{y}{x}\right)^{\sum M_i(\boldsymbol{b})}.$$

If $x \neq y$, M preserves weight and we are done by the same argument as for when $xz \neq y$.

Otherwise if $x = y$, (9) becomes

$$x^{\sum a_i + \sum b_i - 2\sum a_i b_i} = x^{\sum M_i(\boldsymbol{a}) + \sum M_i(\boldsymbol{b}) - 2\sum M_i(\boldsymbol{a}) M_i(\boldsymbol{b})}.$$

This is equivalent to

$$x^{\frac{1}{2} - \frac{1}{2}\sum (2a_i - 1)(2b_i - 1)} = x^{\frac{1}{2} - \frac{1}{2}\sum (2M_i(\boldsymbol{a}) - 1)(2M_i(\boldsymbol{b}) - 1)},$$

which can be written as

$$x^{\sum a_i' b_i'} = x^{\sum M_i(\boldsymbol{a})' M_i(\boldsymbol{b})'}$$

where $a_i' = 2a_i - 1 \in \{-1, 1\}$, and similarly for b_i', and $M_i(\boldsymbol{a})'$, $M_i(\boldsymbol{b})'$. We can fix the same $\boldsymbol{a}$ and $\boldsymbol{b}$ as above, and this gives $\boldsymbol{a}' = (-1, -1, -1, 1, 1, 1)$ and $\boldsymbol{b}' = (-1, 1, -1, 1, -1, 1)$. Then we get $2 = \sum_{i=1}^{6} a_i' b_i' \neq \sum_{i=1}^{6} M_i(\boldsymbol{a})' M_i(\boldsymbol{b})' = -2$. Thus we must have $x = 1$, and then $y = 1$ and $z = 1$, which contradicts $z \neq xy$. We have proved that if $z \neq xy$ the partition function $Z_{\mathcal{F}}(\cdot)$ is #P-hard.

References

1 Carlo Baldassi, C. Borgs, J. Chayes, A. Ingrosso, Carlo Lucibello, Luca Saglietti, and R. Zecchina. Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes. *Proceedings of the National Academy of Sciences*, 113:E7655–E7662, 2016.

2 Alexander Barvinok. *Combinatorics and Complexity of Partition Functions*, volume 30. Springer, 2016. `doi:10.1007/978-3-319-51829-9`.

3 Rodney J Baxter. *Exactly solved models in statistical mechanics*. Elsevier, 2016.

4 Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006.

5 A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: an algorithm for satisfiability. *Random Struct. Algorithms*, 27:201–226, 2005.

6 Graham R. Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of Combinatorial Theory, Series B*, 77(2):221–262, 1999. `doi:10.1006/jctb.1999.1899`.

7 Andrei Bulatov, Martin Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby. The complexity of weighted and unweighted #CSP. *Journal of Computer and System Sciences*, 78(2):681–688, 2012. Games in Verification. doi:10.1016/j.jcss.2011.12.002.

8 Andrei Bulatov and Martin Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2):148–186, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004). doi:10.1016/j.tcs.2005.09.011.

9 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008*, volume 5125 of *Lecture Notes in Computer Science*, pages 646–661. Springer, 2008. Also in J. ACM 60.5 (October 2013). doi:10.1007/978-3-540-70575-8_53.

10 Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Information and Computation*, 205(5):651–678, 2007. doi:10.1016/j.ic.2006.09.005.

11 Robert Burton and Jeffrey E. Steif. Non–uniqueness of measures of maximal entropy for subshifts of finite type. *Ergodic Theory and Dynamical Systems*, 14(2):213–235, 1994. doi:10.1017/S0143385700007859.

12 Jin-Yi Cai and Xi Chen. A decidable dichotomy theorem on directed graph homomorphisms with non-negative weights. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 437–446. IEEE Computer Society, 2010. Also in Comput. Complex. 28.3 (2019). doi:10.1109/FOCS.2010.49.

13 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 909–920. ACM, 2012. Also in J. ACM 64.3 (2017). doi:10.1145/2213977.2214059.

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

15 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Non-negatively weighted #CSP: An effective complexity dichotomy. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 45–54. IEEE Computer Society, 2011. Also in SIAM J. Comput. 45.6 (2016). doi:10.1109/CCC.2011.32.

16 Jin-Yi Cai and Artem Govorov. Dichotomy for graph homomorphisms with complex values on bounded degree graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1103–1111. IEEE, 2020. doi:10.1109/FOCS46700.2020.00106.

17 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. The complexity of complex weighted boolean #CSP. *J. Comput. Syst. Sci.*, 80(1):217–236, 2014. doi:10.1016/j.jcss.2013.07.003.

18 Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Information and computation*, 125(1):1–12, 1996.

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

20 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM Journal on Computing*, 39(3):843–873, 2009. doi:10.1137/07068062X.

21 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000. doi:10.1002/1098-2418(200010/12)17:3/4<260::AID-RSA5>3.0.CO;2-W.

22 Martin Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM Journal on Computing*, 42(3):1245–1274, 2013. doi:10.1137/100811258.

23 Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM J. Comput.*, 31(5):1527–1541, 2002. doi:10.1137/S0097539701383844.

24 Martin E. Dyer, Leslie Ann Goldberg, and Mike Paterson. On counting homomorphisms to directed acyclic graphs. In *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 38–49. Springer, 2006. Also in J. ACM 54.6 (December 2007). doi:10.1007/11786986_5.

25 Leslie Ann Goldberg and Mark Jerrum. A complexity classification of spin systems with an external field. *Proceedings of the National Academy of Sciences*, 112(43):13161–13166, 2015. doi:10.1073/pnas.1505664112.

26 Artem Govorov, Jin-Yi Cai, and Martin E. Dyer. A dichotomy for bounded degree graph homomorphisms with nonnegative weights. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 66:1–66:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.66.

27 P. Hell and J. Nesetril. *Graphs and Homomorphisms*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2004. URL: https://books.google.com/books?id=bJXWV-qK7kYC.

28 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, July 2001. doi:10.1145/502090.502098.

29 A. Ihler and David A. McAllester. Particle belief propagation. In *AISTATS*, 2009.

30 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX–CUT and other 2–variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

31 J. L. Lebowitz and G. Gallavotti. Phase transitions in binary lattice gases. *Journal of Mathematical Physics*, 12(7):1129–1133, 1971.

32 Liang Li, Pinyan Lu, and Yitong Yin. Approximate counting via correlation decay in spin systems. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 922–940. SIAM, 2012.

33 Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 67–84. SIAM, 2013.

34 Jiabao Lin and Hanpin Wang. The complexity of Boolean Holant problems with nonnegative weights. *SIAM J. Comput.*, 47(3):798–828, 2018. doi:10.1137/17M113304X.

35 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3-4):321–328, 1967.

36 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

37 Pinyan Lu and Yitong Yin. Approximating the partition function of two–spin systems. In *Encyclopedia of Algorithms*, pages 117–123. Springer, 2016. doi:10.1007/978-1-4939-2864-4_750.

38 M. MacDonald and Mark S. Seidenberg. Constraint satisfaction accounts of lexical and sentence comprehension, 2006.

39 Elitza Maneva, Elchanan Mossel, and Martin J. Wainwright. A new look at survey propagation and its generalizations. *J. ACM*, 54(4):17–es, July 2007. doi:10.1145/1255443.1255445.

40 Nima Noorshams and M. Wainwright. Belief propagation for continuous state spaces: stochastic message-passing with quantitative guarantees. *J. Mach. Learn. Res.*, 14:2799–2835, 2013.

41 P. Raghavendra and D. Steurer. How to round any CSP. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 586–594, 2009.

42 Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 245–254, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374414.

43 Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two–state anti–ferromagnetic spin systems on bounded degree graphs. *Journal of Statistical Physics*, 155(4):666–686, 2014.

44 Alistair Sinclair, Piyush Srivastava, and Yitong Yin. Spatial mixing and approximation algorithms for graphs with bounded connective constant. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 300–309. IEEE, 2013.

45 Allan Sly and Nike Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 361–369. IEEE, 2012.

46 L. Song, A. Gretton, D. Bickson, Y. Low, and Carlos Guestrin. Kernel belief propagation. In *AISTATS*, 2011.

47 Mauricio Toro, C. Rueda, Carlos Agón, and G. Assayag. GELISP: A framework to represent musical constraint satisfaction problems and search strategies. *Journal of theoretical and applied information technology*, 86:327–331, 2016.

48 M. Wainwright, T. Jaakkola, and A. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51:3697–3717, 2005.

49 Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 140–149, 2006.

50 Mingji Xia. Holographic reduction: A domain changed application and its partial converse theorems. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 666–677. Springer, 2010. doi:10.1007/978-3-642-14165-2_56.

51 Jonathan S. Yedidia, W. Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, 2000.

52 Hai-Jun Zhou. Spin glass approach to the feedback vertex set problem. *The European Physical Journal B*, 86:1–9, 2013.

Continuous Rational Functions Are Deterministic Regular

Olivier Carton
IRIF, Université Paris Cité, France

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

Abstract

A word-to-word function is rational if it can be realized by a non-deterministic one-way transducer. Over finite words, it is a classical result that any rational function is regular, i.e. it can be computed by a deterministic two-way transducer, or equivalently, by a deterministic streaming string transducer (a one-way automaton which manipulates string registers).

This result no longer holds for infinite words, since a non-deterministic one-way transducer can guess, and check along its run, properties such as infinitely many occurrences of some pattern, which is impossible for a deterministic machine. In this paper, we identify the class of rational functions over infinite words which are also computable by a deterministic two-way transducer. It coincides with the class of rational functions which are continuous, and this property can thus be decided. This solves an open question raised in a previous paper of Dave et al.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases infinite words, rational functions, determinization, continuity, streaming string transducers, two-way transducers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.28

Related Version *Full Version*: https://arxiv.org/abs/2204.11235

Funding This work was supported by the DeLTA ANR project (ANR-16-CE40-0007).

1 Introduction

Transducers are finite-state machines obtained by adding outputs to finite automata. They are very useful in a lot of areas like coding, computer arithmetic, language processing or program analysis, and more generally in data stream processing. In this paper, we study transducers which compute partial functions. They are either deterministic, or non-deterministic but unambiguous (they have at most one accepting run on a given input).

Over finite words, a deterministic two-way transducer (2-dT) consists of a deterministic two-way automaton which can produce outputs. Such machines realize the class of *regular functions*, which is often considered as one of the functional counterparts of regular languages. It coincides with the class of functions definable by monadic second-order transductions [7], or copyless deterministic streaming string transducers (dSST), which is a model of one-way automata manipulating string registers [1]. On the other hand, the model of non-deterministic one-way transducers (1-nT) describe the well-known class of *rational functions*. It is well known that any rational function is regular, but the converse does not hold.

Infinite words. The class of *regular functions over infinite words* was defined in [2] using monadic second-order transductions. It coincides with the class of functions realized by 2-dT with ω-regular lookahead, or by copyless dSST with some Müller conditions. However, the use of ω-regular lookaheads (or Müller conditions for dSST) is necessary to capture the expressive power of monadic second-order logic on infinite words, in order to check properties such as infinitely many occurrences of some pattern. Similarly, the model of 1-nT with Büchi acceptance conditions defines the subclass of *rational functions over infinite words*.

Even if regular and rational functions give very natural frameworks for specification (due to their connections with logic), not all these functions can effectively be computed by a deterministic machine without lookaheads. It turns out that the regular functions which can be computed by a deterministic Turing machine (doing an infinite computation on its infinite input) are exactly those which are continuous for the Cantor topology [5]. Furthermore continuity can be decided, which was has been known for rational functions since [10].

The authors of [5] conjecture that any continuous rational (or even regular) function can in fact be computed by a 2-dT (without lookahead), instead of a Turing machine. A partial answer was obtained in [8], whose results imply that 2-dT can be built for a subclass of rational functions defined by 1-nT where some forms of non-determinism are prohibited. Their proof is based on game-theoretic techniques.

Contributions. This paper shows that any continuous rational function over infinite words can be extended to a function which is computable by a 2-dT (without lookaheads). Since the converse also holds, this result completely characterizes rational functions which can be computed by 2-dTs, up to an extension of the domain. Furthermore, this property is decidable and our construction of a 2-dT is effective.

This result is tight, in the sense that two-way moves cannot be avoided. Indeed, one-way deterministic transducers (describing the class of *sequential functions*) cannot realize all continuous rational functions, even when only considering total functions (contrary to what happens for the subclass of rational functions studied in [8]).

In order to establish this theorem, we first study the expressive power of 2-dT over infinite words. We introduce the class of *deterministic regular functions* as the class of functions computed by 2-dT (as opposed to the regular functions, which are not entirely deterministic since they use lookaheads to guess the future). Following the aforementioned equivalences between two-way and register transducers, we prove that deterministic rational functions are exactly the functions which are realized by copyless dSST (without Müller conditions). Hence our problem is reduced to showing that any continuous rational function can be realized by a copyless dSST. Building a copyless dSST is also relevant for practical applications, since it corresponds to a streaming algorithm over infinite strings.

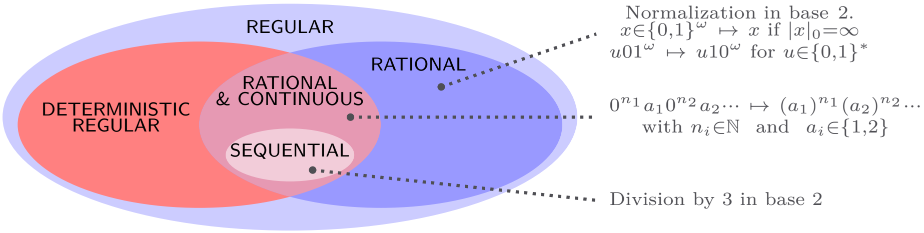

REGULAR
RATIONAL
DETERMINISTIC REGULAR
RATIONAL & CONTINUOUS
SEQUENTIAL
Normalization in base 2.
$x{\in}\{0,1\}^\omega \mapsto x$ if $|x|_0{=}\infty$
$u01^\omega \mapsto u10^\omega$ for $u{\in}\{0,1\}^*$
$0^{n_1}a_1 0^{n_2}a_2 \cdots \mapsto (a_1)^{n_1}(a_2)^{n_2}\cdots$
with $n_i{\in}\mathbb{N}$ and $a_i{\in}\{1,2\}$
Division by 3 in base 2

Figure 1 Classes of partial functions over infinite words studied in this paper.

Then we introduce various new concepts in order to transform a 1-nT computing a continuous function into a dSST. This determinization procedure is rather involved. The main difficulty is that even if the 1-nT is unambiguous, it might not check its guesses after reading only a finite number of letters. In other words, a given input can label several infinite runs, even if only one of them is accepting. However, a deterministic machine can never determine which run is the accepting one, since it requires to check whether a property occurs infinitely often. This intuition motivates our key definition of *compatible sets* among the states of a 1-nT. Such sets are the sets of states which have a “common infinite future”. The restriction of 1-nT considered in [8] leads to compatible sets which are always singletons (hence their condition defines a natural special case). We show that when the function computed by the 1-nT is continuous, the outputs produced along finite runs which end in a compatible set enjoy several combinatorial properties.

We finally describe how to build a dSST which realizes the continuous function given by a 1-nT. Its construction is is rather complex, and it crucially relies on the aforementioned properties of compatible sets. These sets are manipulated by the dSST in an original tree-like fashion. To the knowledge of the authors, this construction of this dSST is completely new (in particular, it is not based on the constructions of [5] nor of [8]).

Outline. We recall in Section 2 the definitions of rational functions and one-way transducers. In Section 3, we present the new class of deterministic regular functions and give the various transducer models which capture it. Our main result which relates continuous rational and deterministic regular functions is given in Section 4. The proof is sketched in sections 4 and 5.

2 Rational functions

Letters A, B denote alphabets, i.e. finite sets of letters. The set A^* (resp. A^+, A^ω) denotes the set of finite words (resp. non-empty finite words, infinite words) over the alphabet A. If $u \in A^* \cup A^\omega$, we let $|u| \in \mathbb{N} \cup \{\infty\}$ be its length. For $a \in A$, $|u|_a$ denotes the number of a in u. For $1 \leqslant i \leqslant |u|$, $u[i] \in A$ is the i-th letter of u. If $1 \leqslant i \leqslant j \leqslant |u|$, $u[i{:}j]$ stands for $u[i]u[i{+}1] \cdots$ until j. We write $w[i{:}]$ for $u[i{:}|u|]$. If $j > |u|$ we let $u[i{:}j] := u[i{:}|u|]$. If $j < i$ we let $u[i{:}j] := \varepsilon$. We write $u \sqsubseteq v$ (resp. $u \sqsubset v$) when u is a (resp. strict) prefix of v. Given two words u, v, we let $u \wedge v$ be their longest common prefix. We say that u, v are *mutual prefixes* if $u \sqsubseteq v$ or $v \sqsubseteq u$. In this case we let $u \vee v$ be the longest of them. A function f between two sets S, T is denoted by $f : S \to T$. If f is a *partial function* (i.e. possibly with non-total domain), it is denoted $f : S \rightharpoonup T$. Its domain is denoted $\mathsf{Dom}(f)$.

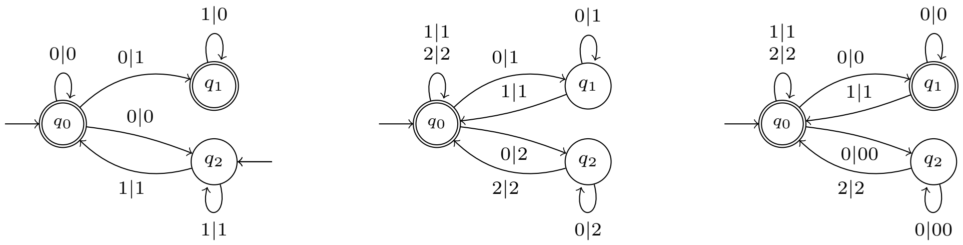
▶ **Definition 2.1.** *A* one-way non-deterministic transducer (1-nT) $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ *is:*

- *a finite input (respectively output) alphabet A (respectively B);*
- *a finite set of states Q with $I \subseteq Q$ initial and $F \subseteq Q$ final;*
- *a transition relation $\Delta \subseteq Q \times A \times Q$;*
- *an output function $\lambda : \Delta \to B^*$ (defined for each transition).*

We write $q \xrightarrow{a|\alpha} q'$ whenever $(q, a, q') \in \Delta$ and $\lambda(q, a, q') = \alpha$. A *run* labelled by some $x \in A^* \cup A^\omega$ is a sequence of consecutive transitions $\rho := q_0 \xrightarrow{x[1]|\alpha_1} q_1 \xrightarrow{x[2]|\alpha_2} q_2 \cdots$. The *output* of ρ is the word $\alpha_1\alpha_2 \cdots \in A^* \cup A^\omega$. If $x \in A^\omega$, we also write $q_0 \xrightarrow{x|\alpha_1\alpha_2\cdots} \infty$ to denote an infinite run starting in q_0. The run ρ is *initial* if $q_0 \in I$, *final* if $x \in A^\omega$ and $q_i \in F$ infinitely often (Büchi condition), and *accepting* if both initial and final. $\mathcal{T}$ computes the *relation* $\{(x, y) : y \in B^\omega$ is output along an accepting run on $x\}$. It is *functional* if this relation is a (partial) function. In this case, $\mathcal{T}$ can be transformed in an equivalent *unambiguous* 1-nT (a transducer which has at most one accepting run on each $x \in A^\omega$) [3, Corollary 3]. A function $f : A^\omega \rightharpoonup B^\omega$ is said to be *rational* if it can be computed by a (unambiguous) 1-nT.

▶ **Example 2.2.** In Figure 2, we describe 1-nTs which compute the following functions:

- $\mathsf{normalize} : \{0,1\}^\omega \rightharpoonup \{0,1\}^\omega$ mapping $x \mapsto x$ if $|x|_0 = \infty$ and $u01^\omega \mapsto u10^\omega$ if $u \in \{0,1\}^*$;
- $\mathsf{replace} : \{0,1,2\}^\omega \rightharpoonup \{1,2\}^\omega$ with $\mathsf{Dom}(\mathsf{replace}) = \{x : |x|_1 = \infty$ or $|x|_2 = \infty\}$ and mapping $0^{n_1}a_1 0^{n_2}a_2 \cdots \mapsto {a_1}^{n_1+1}{a_2}^{n_2+1} \cdots$ if $a_i \in \{1,2\}$, $n_i \in \mathbb{N}$;
- $\mathsf{double} : \{0,1,2\}^\omega \to \{0,1,2\}^\omega$ mapping $0^{n_1}a_1 0^{n_2}a_2 \cdots \mapsto 0^{a_1 n_1}a_1 0^{a_2 n_2}a_2 \cdots$ and $0^{n_1}a_1 \cdots 0^{n_m}a_m 0^\omega \mapsto 0^{a_1 n_1}a_1 \cdots 0^{a_m n_m}a_m 0^\omega$ (if finitely many 1 or 2).



(a) 1-nT computing normalize. **(b)** 1-nT computing replace. **(c)** 1-nT computing double.

Figure 2 Unambiguous, clean and trim 1-nTs computing the functions of Example 2.2.

▶ Remark 2.3. The functions mentioned in Example 2.2 are not *sequential*, i.e. they cannot be computed by deterministic one-way transducers (i.e. deterministic 1-nTs).

A 1-nT is *trim* if any state is both accessible and co-accessible, or equivalently if it occurs in some accepting run. It is *clean* if the production along any accepting run is infinite.

▶ **Lemma 2.4.** *A trim* 1-nT *is clean if and only if for all $q \in F$, the existence of a cycle $q \xrightarrow{u|\alpha} q$ for $u \in A^+$ implies $\alpha \neq \varepsilon$. Given an unambiguous* 1-nT*, one can build an equivalent unambiguous, clean and trim* 1-nT*.*

3 Deterministic regular functions

We now introduce the new class of *deterministic regular* functions, which are computed by *deterministic two-way transducers*. Contrary to 1-nTs, such machines cannot test ω-regular properties of their input. Hence they describe continuous (and computable) functions.

▶ **Definition 3.1.** *A* deterministic two-way transducer (2-dT) $\mathcal{T} = (A, B, Q, q_0, \delta, \lambda)$ *is:*

- *an input alphabet A and an output alphabet B;*
- *a finite set of states Q with an initial state $q_0 \in Q$;*
- *a transition function $\delta : Q \times (A \uplus \{\vdash\}) \rightharpoonup Q \times \{\triangleleft, \triangleright\}$;*
- *an output function $\lambda : Q \times (A \uplus \{\vdash\}) \rightharpoonup B^*$ with same domain as δ.*

If the input is $x \in A^\omega$, then $\mathcal{T}$ is given as input the word $\vdash x$. The symbol $\vdash$ is used to mark the beginning of the input. We denote by $x[0] := \vdash$. A *configuration* over $\vdash x$ is a tuple (q, i) where $q \in Q$ is the current state and $i \geqslant 0$ is the current position of the reading head. The *transition relation* $\rightarrow$ is defined as follows. Given a configuration (q, i), let $(q', \star) := \delta(q, w[i])$. Then $(q, i) \rightarrow (q', i')$ whenever either $\star = \triangleleft$ and $i' = i - 1$ (move left), or $\star = \triangleright$ and $i' = i + 1$ (move right). A *run* is a (finite or infinite) sequence of configurations $(q_1, i_1) \rightarrow (q_2, i_2) \rightarrow \cdots$. An *accepting* run is an infinite run which starts in $(q_0, 0)$ and such that $i_n \rightarrow \infty$ when $n \rightarrow \infty$ (otherwise the transducer repeats the same loop).

The partial function $f : A^\omega \rightharpoonup B^\omega$ computed by $\mathcal{T}$ is defined as follows. Let $x \in A^\omega$ be such that there exists a (unique) accepting run $(q_0^x, i_0^x) \rightarrow (q_1^x, i_1^x) \rightarrow \cdots$ labelled by x. Let $y := \prod_{j=1}^{\infty} \lambda(q_j^x, w[i_j^x]) \in B^* \cup B^\omega$ be the concatenation of the outputs produced along this run. If $y \in B^\omega$, we define $f(x) := y$. Otherwise $f(x)$ is undefined.

▶ **Example 3.2.** The function replace from Example 2.2 can be computed by 2-dT. For each $i \geqslant 1$, this 2-dT crosses the block 0^{n_i} to determines a_i, and then crosses the block once more and outputs ${a_i}^{n_i+1}$. The function double can be computed using similar ideas. However, an important difference is that the 2-dT must output the block 0^{n_i} when it crosses it for the first time, in order to ensure that the production over 0^ω is 0^ω.

There exists deterministic regular functions which are not rational, for instance the function which reverses (mirror image) a prefix of its input.

Over finite words, it is known that two-way transducers are equivalent to copyless streaming string transducers [1]. Over infinite words, a similar equivalence holds between two-way transducers with lookahead and copyless streaming string transducers with Müller output conditions [2]. These models define the class of *regular functions* over infinite words. However, lookaheads enable two-way transducers to check ω-regular properties of their input (and thus non-computable behaviors). Hence our deterministic regular functions form a strict subclass of these regular functions over infinite words.

We now introduce a model of streaming string transducer to describe deterministic regular functions, in the spirit of the aforementioned results. In our setting, it consists of a one-way deterministic automaton with a finite set $\mathfrak{R}$ of registers that store words from B^*. We use a distinguished register $\mathfrak{out}$ to store the output produced when reading an infinite word. The registers are modified using *substitutions*, i.e. mappings $\mathfrak{R} \to (B \uplus \mathfrak{R})^*$. We denote by $\mathcal{S}^B_{\mathfrak{R}}$ the set of these substitutions. They can be extended morphically from $(B \uplus \mathfrak{R})^*$ to $(B \uplus \mathfrak{R})^*$ by preserving the elements of B. They can be composed (see Example 3.3).

▶ **Example 3.3.** Let $\mathfrak{R} = \{\mathfrak{r}, \mathfrak{s}\}$ and $B = \{b\}$. Consider $\sigma_1 := \mathfrak{r} \mapsto b, \mathfrak{s} \mapsto b\mathfrak{r}\mathfrak{s}b$ and $s_2 := \mathfrak{r} \mapsto \mathfrak{r}b, \mathfrak{s} \mapsto \mathfrak{r}\mathfrak{s}$, then $\sigma_1 \circ \sigma_2(\mathfrak{r}) = s_1(\mathfrak{r}b) = bb$ and $\sigma_1 \circ \sigma_2(\mathfrak{s}) = \sigma_1(\mathfrak{r}\mathfrak{s}) = bb\mathfrak{r}\mathfrak{s}b$.

▶ **Definition 3.4.** *A* deterministic streaming string transducer (dSST) *is:*

- *a finite input (resp. output) alphabet A (resp. B);*
- *a finite set of states Q with $q_0 \in Q$ initial;*
- *a transition function $\delta : Q \times A \rightharpoonup Q$;*
- *a finite set of registers $\mathfrak{R}$ with a distinguished output register $\mathfrak{out} \in \mathfrak{R}$;*
- *an update function $\lambda : Q \times A \rightharpoonup \mathcal{S}^B_{\mathfrak{R}}$ such that for all $(q, a) \in \mathsf{Dom}(\lambda) = \mathsf{Dom}(\delta)$:*
  - *$\lambda(q, a)(\mathfrak{out}) = \mathfrak{out} \cdots$;*
  - *there is no other occurence of $\mathfrak{out}$ in $\{\lambda(q, a)(\mathfrak{r}) : \mathfrak{r} \in \mathfrak{R}\}$.*

We denote it $\mathcal{T} = (A, B, Q, q_0, \delta, \mathfrak{R}, \mathfrak{out}, \lambda)$.

This machine defines a function $f : A^* \rightharpoonup B^*$ as follows. For $i \geqslant 0$ let $q^x_i := \delta(q_0, x[1{:}i])$ (when defined). For $i \geqslant 1$, we let $\lambda^x_i := \lambda(q^x_{i-1}, x[i])$ (when defined) and $\lambda^x_0(\mathfrak{r}) = \varepsilon$ for all $\mathfrak{r} \in \mathfrak{R}$. For $i \geqslant 0$, define the substitution $[\![\cdot]\!]^x_i := \lambda^x_0 \circ \cdots \circ \lambda^x_i$. By construction we get $[\![\mathfrak{out}]\!]^x_i \sqsubseteq [\![\mathfrak{out}]\!]^x_{i+1}$ (when defined). If $[\![\mathfrak{out}]\!]^x_i$ is defined for all $i \geqslant 0$ and $|[\![\mathfrak{out}]\!]^x_i| \to +\infty$, we let $f(x) := \bigvee_i [\![\mathfrak{out}]\!]^x_i$ (it denotes the unique infinite word y such that $[\![\mathfrak{out}]\!]^x_i \sqsubseteq y$ for all $i \geqslant 0$). Otherwise $f(x)$ is undefined.

We say that a substitution $\sigma \in \mathcal{S}^B_{\mathfrak{R}}$ is *copyless* (resp. K-bounded) if for all $\mathfrak{r} \in \mathfrak{R}$, $\mathfrak{r}$ occurs at most once in $\{\sigma(\mathfrak{s}) : \mathfrak{s} \in \mathfrak{R}\}$ (resp. for all $\mathfrak{r}, \mathfrak{s} \in \mathfrak{R}$, $\mathfrak{r}$ occurs at most K times in $\sigma(\mathfrak{s})$).

▶ **Definition 3.5** (Copy restrictions). *We say that a* dSST $\mathcal{T} = (A, B, Q, q_0, \delta, \mathfrak{R}, \mathfrak{out}, \lambda)$ *is* copyless *(resp.* K-bounded*) if for all $x \in A^\omega$ and $i \leqslant j$ such that $\lambda^x_i \circ \cdots \circ \lambda^x_j$ is defined, this substitution is copyless (resp. K-bounded).*

▶ **Example 3.6.** The function replace from Example 2.2 can be computed by a copyless dSST. For all $i \geqslant 1$, it crosses the block 0^{n_i} and computes 1^{n_i} and 2^{n_i} in two registers. Once it sees a_i it adds in $\mathfrak{out}$ the register storing ${a_i}^{n_i}$. The function double can be computed using similar ideas. However, an important difference is that the dSST must directly output the block 0^{n_i} while crossing it, in order to ensure that the production over 0^ω is 0^ω.

The proof of the next result is quite involved, but it is largely inspired by the techniques used for regular functions over finite or infinite words (see e.g. [4, 6]).

▶ **Theorem 3.7.** *The following machines compute the same class of functions $A^\omega \rightharpoonup B^\omega$:*
1. *deterministic two-way transducers* (2-dT)*;*
2. *K-bounded deterministic streaming string transducers* (*K-bounded* dSST)*;*
3. *copyless deterministic streaming string transducers* (*copyless* dSST)*.*

Furthermore, all the conversions are effective.

▶ Remark 3.8. Even if this result is a variant of existing results over finite or infinite words, it requires a proof on its own. Indeed, the authors are not aware of a direct proof which would enable to deduce it from the existing similar results.

Let us now describe the domains of deterministic regular functions. We say that a language is *Büchi deterministic* if it is accepted by a deterministic Büchi automaton [9].

▶ **Proposition 3.9.** *If f is deterministic regular, then* $\mathsf{Dom}(f)$ *is Büchi deterministic.*

We finally give a closure property of deterministic regular functions under pre-composition.

▶ **Definition 3.10.** *A* restricted 1-nT *is a* 1-nT *whose states all are final.*

The semantics of a restricted 1-nT $\mathcal{N} = (A, B, Q, I, \Delta, \lambda)$ is defined so that it *always* computes a function $f : A^\omega \rightharpoonup B^\omega$. The domain $\mathsf{Dom}(f)$ is the set of $x \in A^\omega$ such that $\mathcal{N}$ has a unique accepting run labelled by x, and such that the output along this unique run is infinite. In this case, we let $f(x)$ be the output of along this run. Intuitively, such a transducer expresses the ability to make non-deterministic guesses, as long as these guesses can be verified after reading a finite number of letters (i.e. there are no two possible infinite runs).

▶ **Theorem 3.11.** *Given a restricted* 1-nT *computing a function $f : A^\omega \rightharpoonup B^\omega$ and a deterministic regular function $g : B^\omega \rightharpoonup C^\omega$, $g \circ f$ is (effectively) deterministic regular.*

4 Continuous rational functions are deterministic regular

We now state the main result of this paper, which shows that a rational function can be extended to a deterministic regular function. Using an extension of the original function is necessary since not all ω-regular languages are Büchi deterministic (see Proposition 3.9). Note that Theorem 4.2 is in fact an equivalence, in the sense that a rational function which can be extended to a deterministic regular function is obviously continuous.

We recall that a function $f : A^\omega \rightharpoonup B^\omega$ is *continuous* if and only if for all $x \in \mathsf{Dom}(f)$ and $n \geqslant 0$, there exists $p \geqslant 0$ such that $\forall y \in \mathsf{Dom}(f)$, $|x \wedge y| \geqslant p \Rightarrow |f(x) \wedge f(y)| \geqslant n$.

▶ **Example 4.1.** The functions replace and double are continuous, but normalize is not.

▶ **Theorem 4.2.** *Given a continuous rational function $f : A^\omega \rightharpoonup B^\omega$, one can build a deterministic regular function f' which extends f (i.e. for all $x \in \mathsf{Dom}(f)$, $f(x) = f'(x)$).*

To prove Theorem 4.2, it is enough by theorems 3.7 and 3.11 to show that f' can be computed as a composition of a restricted 1-nT and a K-bounded dSST (see Subsection 4.2, the construction will in fact give a 1-bounded transducer).

4.1 Properties of continuous rational functions

We first describe some structural properties of 1-nT computing continuous functions. In this subsection, we let $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ be an unambiguous, clean and trim 1-nT computing a continuous function $f : A^\omega \rightharpoonup B^\omega$. It is well known that $\mathcal{T}$ verifies Lemma 4.3. This property is in fact equivalent to the continuity of f (see e.g. [10] or [5]).

▶ **Lemma 4.3.** *For all $q_1, q_2 \in I$, $q_1' \in F$, $q_2' \in Q$, $u \in A^*$, $u' \in A^+$, $\alpha_1, \alpha_1', \alpha_2, \alpha_2' \in B^*$ such that $q_i \xrightarrow{u|\alpha_i} q_i' \xrightarrow{u'|\alpha_i'} q_i'$ for $i \in \{1, 2\}$ we have (note that $\alpha_1' \neq \varepsilon$ since $\mathcal{T}$ is clean):*

- *if $\alpha_2' \neq \varepsilon$, then $\alpha_1 {\alpha_1'}^\omega = \alpha_2 {\alpha_2'}^\omega$;*
- *if $\alpha_2' = \varepsilon$, $x \in A^\omega$, $\beta \in B^\omega$ and $q_2' \xrightarrow{x|\beta} \infty$ is final, then $\alpha_1 {\alpha_1'}^\omega = \alpha_2 \beta$.*

Empty cycles $q \xrightarrow{u|\varepsilon} q$ for $q \notin F$ cannot be avoided in a 1-nT. However, we shall see in Lemma 4.4 that such cycles can be avoided if the function is continuous. Formally, we say that the clean $\mathcal{T}$ is *productive* if the hypotheses of Lemma 4.3 imply $\alpha_2' \neq \varepsilon$.

▶ **Lemma 4.4.** *Given $\mathcal{T}$, one can build an equivalent unambiguous, trim and productive* 1-nT.

Compatible sets and steps. We now introduce the key notion of a *compatible set* which is a set of states having a "common future" and such that one of the future runs is accepting.

▶ **Definition 4.5** (Compatible set). *We say that a set of states $C \subseteq Q$ is* compatible *whenever there exists $x \in A^\omega$ and infinite runs ρ_q for each $q \in C$ labelled by x such that:*

- *$\forall q \in C$, ρ_q starts from q;*
- *$\exists q \in C$ such that ρ_q is final.*

Let Comp be the set of compatible sets. If $S \subseteq Q$, let $\mathsf{Comp}(S)$ be the set $2^S \cap \mathsf{Comp}$.

▶ **Definition 4.6** (Pre-step). *We say that C, u, D is a* pre-step *if $C, D \in \mathsf{Comp}$, $u \in A^*$ and for all $q \in D$, there exists a unique state $\mathsf{pre}^u_{C,D}(q) \in C$ such that $\mathsf{pre}^u_{C,D}(q) \xrightarrow{u} q$.*

Note that for all $D' \in \mathsf{Comp}(D)$, we have $\mathsf{pre}^u_{C,D}(D') \in \mathsf{Comp}$.

▶ **Definition 4.7** (Step). *We say that a pre-step C, u, D is a* step *if $\mathsf{pre}^u_{C,D}$ is surjective.*

Given $q \in D$, let $\mathsf{prod}^u_{C,D}(q)$ be the output $\alpha \in B^*$ produced along the run $\mathsf{pre}^u_{C,D}(q) \xrightarrow{u|\alpha} q$. We say that a (pre-)step is *initial* whenever $C \subseteq I$. We first claim that the productions along the runs of an initial step are mutual prefixes. Lemma 4.3 is crucial here.

▶ **Lemma 4.8.** *Let J, u, C be an initial step. Then $\mathsf{prod}^u_{J,C}(q)$ for $q \in C$ are mutual prefixes.*

▶ **Example 4.9.** In Figure 2b, if a step is initial, it is of the form $\{q_0\}, u, \{q_i\}$ for some $i \in \{0, 1, 2\}$. In Figure 2c, $\{q_0\}, 0^n, \{q_1, q_2\}$ is an initial step for all $n \geqslant 0$.

▶ **Definition 4.10** (Common, advance). *Let J, u, C be an initial step. We define:*

- *the* common *$\mathsf{com}^u_{J,C} \in B^*$ as the longest common prefix $\bigwedge_{q \in C} \mathsf{prod}^u_{J,C}(q)$;*
- *for all $q \in C$, its* advance *$\mathsf{adv}^u_{J,C}(q) \in B^*$ as $(\mathsf{com}^u_{J,C})^{-1}\, \mathsf{prod}^u_{J,C}(q)$;*
- *the* maximal advance *$\mathsf{max}\text{-}\mathsf{adv}^u_{J,C}$ as the longest advance, i.e. $\bigvee_{q \in C} \mathsf{adv}^u_{J,C}(q)$.*

Definition 4.10 makes sense by Lemma 4.8, and furthermore $\mathsf{prod}^u_{J,C}(q) = \mathsf{com}^u_{J,C}\, \mathsf{adv}^u_{J,C}(q)$ for all $q \in C$. Now let $M := \max_{q,q' \in Q, a \in A} |\lambda(q, a, q')|$ and $\Omega := M|Q|^{|Q|}$. We say that a compatible set C is *separable* if there exists an initial step which ends in C, and such that the lengths of the productions along two of its runs differ of at least Ω.

▶ **Definition 4.11** (Separable set). *Let $C \in \mathsf{Comp}$, we say that C is* separable *if there exists an initial step J, u, C and $p, q \in C$ such that $\big||\mathsf{adv}^u_{J,C}(p)| - |\mathsf{adv}^u_{J,C}(q)|\big| > \Omega$.*

▶ Remark 4.12. In other words, it means that $|\mathsf{max}\text{-}\mathsf{adv}^u_{J,C}| > \Omega$.

It is easy to see (by a pumping argument) that one can decide if a set is separable. We now show that the productions along the initial steps which end in a separable set are forced to "iterate" some value θ if the step is pursued. The following lemma is the key ingredient for showing that a rational function is deterministic regular (see Section 4).

▶ **Lemma 4.13** (Looping futures). *Let $C \in \mathsf{Comp}$ be separable and J, u, C be an initial step (not necessarily the one which makes C separable). There exists $\tau, \theta \in B^*$ with $|\tau| \leqslant 3\Omega$, and $|\theta| = \Omega!$ which can be uniquely determined from C and $\mathsf{adv}^u_{J,C}(q)$ for $q \in C$, such that:*

- $\tau \sqsubseteq \mathsf{max\text{-}adv}^u_{J,C} \sqsubseteq \tau\theta^\omega$;
- *for all step C, v, D and $q \in D$, $\mathsf{prod}^v_{C,D}(q) \sqsubseteq (\mathsf{adv}^u_{J,C}(p))^{-1}\tau\theta^\omega$ with $p := \mathsf{pre}^v_{C,D}(q)$.*

▶ Remark 4.14. Since $\mathsf{adv}^u_{J,C}(p) \sqsubseteq \mathsf{max\text{-}adv}^u_{J,C} \sqsubseteq \tau\theta^\omega$, the second item makes sense.

▶ **Example 4.15.** In Figure 2c, the compatible set $C := \{q_1, q_2\}$ is separable. For all step C, v, D we have $D = C$ thus $v = 0^n$, $\mathsf{prod}^{0^n}_{C,D}(q_1) = 0^n$ and $\mathsf{prod}^{0^n}_{C,D}(q_2) = 0^{2n}$.

4.2 Composition of a restricted 1-nT and a 1-bounded dSST

In the rest of this paper, we let $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ be an unambiguous, productive and trim 1-nT computing a continuous $f : A^\omega \rightharpoonup B^\omega$. Our goal is to rewrite f as the composition of a restricted 1-nT and a 1-bounded dSST. We first build the restricted 1-nT, which computes an over-approximation of the accepting run of $\mathcal{T}$ in terms of compatible sets.

▶ **Lemma 4.16.** *One can build a restricted 1-nT $\mathcal{N}$ computing $g : A^\omega \rightharpoonup (\mathsf{Comp} \uplus A)^\omega$ such that $\mathsf{Dom}(f) \subseteq \mathsf{Dom}(g)$, and for all $x \in \mathsf{Dom}(g)$, $g(x) = C_0 x[1] C_1 x[2] C_2 \cdots$ where:*

- *$C_0 \subseteq I$ and for all $i \geqslant 0$, $C_i, x[i{+}1], C_{i+1}$ is a pre-step;*
- *if $x \in \mathsf{Dom}(f)$ then $\forall i \geqslant 0$, $q^x_i \in C_i$, where $q^x_0 \xrightarrow{x[1]} q^x_1 \xrightarrow{x[2]} \cdots$ is the accepting run of $\mathcal{T}$.*

Given $x \in \mathsf{Dom}(f)$, we denote by $C^x_0, C^x_1, \ldots$ the sequence of compatible sets produced by $\mathcal{N}$ in Lemma 4.16. We now describe a 1-bounded dSST $\mathcal{S}$ which, when given as input $g(x) \in (\mathsf{Comp} \uplus A)^\omega$ for $x \in \mathsf{Dom}(f)$, outputs $f(x)$ (this description is continued in Section 5).

Tree of compatibles. Given $C \in \mathsf{Comp}$, we define $\mathsf{tree}(C)$ as a finite set of words over $\mathsf{Comp}(C)$, which describes the decreasing chains for $\subset$. It can be identified with the set of all root-to-node paths of a tree labelled by elements of $\mathsf{Comp}(C)$, as shown in Example 4.18.

▶ **Definition 4.17** (Tree of compatibles). *Given $C \in \mathsf{Comp}$, we denote by $\mathsf{tree}(C)$ the set of words $\pi = C_1 \cdots C_n \in (\mathsf{Comp}(C))^+$ such that $C_1 = C$ and for all $1 \leqslant i \leqslant n{-}1$, $C_i \supset C_{i+1}$.*

▶ **Example 4.18.** If $C = \{1,2,3\}$ and $\mathsf{Comp}(C) = \{\{1,2,3\},\{1,2\},\{2,3\},\{1\},\{2\},\{3\}\}$, then we have $\mathsf{tree}(C) = \{\{1,2,3\}\{1,2\}\{1\}, \{1,2,3\}\{1,2\}\{2\}, \{1,2,3\}\{2,3\}\{2\}, \{1,2,3\}\{2,3\}\{3\}, \{1,2,3\}\{1\}, \{1,2,3\}\{2\}, \{1,2,3\}\{3\}\}$. Its view as a tree is depicted in Figure 3.

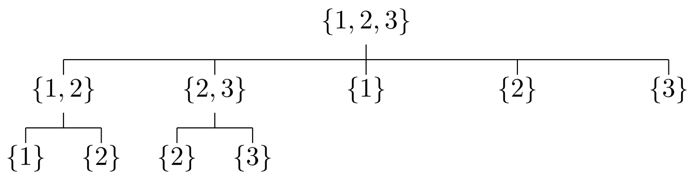


Figure 3 The tree of compatibles obtained from Example 4.18.

Information stored. The states of the dSST $\mathcal{S}$ are partitioned in two categories: the sets of the *separable mode* and the sets of of the *non-separable mode*. A configuration of the dSST $\mathcal{S}$ will always keep track of the following information:

- the content of a register $\mathfrak{out}$;
- two sets $\mathsf{J}, \mathsf{C} \in \mathsf{Comp}$ and a function $\mathsf{pre} : \mathsf{C} \to \mathsf{J}$ (stored in the state);
- a function $\mathsf{lag} : \mathsf{C} \to B^*$ such that $|\mathsf{lag}(q)| \leqslant 3\Omega$ for all $q \in \mathsf{C}$ (stored in the state).

Furthermore, when $\mathcal{S}$ is in a state of the separable mode, it will additionally store:

- a value $\theta \in B^*$ with $|\theta| = \Omega!$ (stored in the state);
- for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ (note that $C_1 = \mathsf{C}$ by definition of $\mathsf{tree}(\mathsf{C})$):
  - a function $\mathsf{nb}_\pi : C_n \to [0{:}4]$ (stored in the state);
  - the content of a register $\mathfrak{out}_\pi$. For $\pi = \mathsf{C}$, we identify the register $\mathfrak{out}_\mathsf{C}$ with $\mathfrak{out}$;
- a function $\mathsf{last} : \mathsf{C} \to B^*$ such that $|\mathsf{last}(q)| < \Omega!$ forall $q \in \mathsf{C}$ (stored in the state).

If a configuration of $\mathcal{S}$ is clearly fixed, we abuse notations and denote by $\mathfrak{out}_\pi$ (resp. nb_π, lag, etc.) the value contained in register $\mathfrak{out}_\pi$ (resp. stored in the state) in this configuration. In a given configuration of $\mathcal{S}$, we say that $\pi \in \mathsf{tree}(\mathsf{C})$ is *close* if for all $\pi \sqsubset \pi' \in \mathsf{tree}(\mathsf{C})$, we have $\mathsf{nb}_{\pi'} = 0$ and $\mathfrak{out}_{\pi'} = \varepsilon$ (intuitively, the subtree rooted in π stores empty informations).

Invariants. The main idea for building $\mathcal{S}$ it the following. If C_i^x is a non-separable set, then the productions along the initial runs which end in C_i^x are mutual prefixes (by Lemma 4.8) which only differ from a bounded information. Hence the common part com of these runs is stored to $\mathfrak{out}$, and the adv are stored in the lag. If C_i^x becomes separable, then these runs still produce mutual prefixes, but two of them can differ by a large information. However by Lemma 4.13, they iterate some value θ. Hence the only relevant information is the number of θ which were produced along these runs. Formally, our construction ensures that the following invariants hold when $\mathcal{S}$ has just read $C_0^x x[1] C_1^x \cdots x[i] C_i^x$ for $i \geqslant 0$:

1. $\mathsf{C} = C_i^x$;
2. $\mathsf{J}, x[1{:}i], \mathsf{C}$ is an initial step and $\mathsf{pre} = \mathsf{pre}_{\mathsf{J},\mathsf{C}}$
3. if C is not separable, then $\mathcal{S}$ is in non-separable mode and:
   a. $\mathfrak{out} = \mathsf{com}_{\mathsf{J},\mathsf{C}}^{x[1:i]}$;
   b. $\mathsf{lag}(q) = \mathsf{adv}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(q)$ for all $q \in \mathsf{C}$.
4. if C is separable, then $\mathcal{S}$ is in separable mode and:
   a. the $\mathsf{lag}(q)$ for $q \in Q$ are mutual prefixes, and so $\mathsf{max\text{-}lag} := \bigvee_{q \in C} \mathsf{lag}(q)$ is defined. Furthermore, there exists $q \in \mathsf{C}$ such that $\mathsf{lag}(q) = \varepsilon$. We say that some $q \in \mathsf{C}$ is *lagging* if and only if $\mathsf{lag}(q) \sqsubset \mathsf{max\text{-}lag}$ (strict prefix), otherwise it is *not lagging*;
   b. if $\pi \in \mathsf{tree}(\mathsf{C})$ is such that $\pi \neq \mathsf{C}$ (i.e. $\mathfrak{out}_\pi \neq \mathfrak{out}$), then $\mathfrak{out}_\pi \in \theta^*$;
   c. for all $q \in \mathsf{C}$, $\mathsf{last}(q) \sqsubseteq \theta^\omega$ (if furthermore $|\mathsf{last}(q)| < \Omega!$, then $\mathsf{last}(q) \sqsubset \theta$);
   d. if q is lagging, then $\mathsf{last}(q) = \varepsilon$ and for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ such that $q \in C_n$, we have $\mathsf{nb}_\pi(q) = 0$ and, if $\pi \neq \mathsf{C}$, $\mathfrak{out}_\pi = \varepsilon$;
   e. for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, for $1 \leqslant i \leqslant n$ define $\pi_i := C_1 \cdots C_i$. If $C_n = \{q\}$, then:
      - $\mathsf{prod}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(q) = \mathfrak{out}\ \mathsf{lag}(q)$ if q is lagging;
      - $\mathsf{prod}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(q) = \mathfrak{out}\ \mathsf{max\text{-}lag}\ \theta^{\mathsf{nb}_{\pi_1}(q)} \left(\prod_{i=2}^{n} \mathfrak{out}_{\pi_i}\, \theta^{\mathsf{nb}_{\pi_i}(q)}\right) \mathsf{last}(q)$ if q is not lagging.
   f. for all future steps C, u, D and for all $q \in D$, $\mathsf{prod}_{\mathsf{J},D}^{x[1:i]u}(q) \sqsubseteq \mathfrak{out}\ \mathsf{max\text{-}lag}\ \theta^\omega$;
   g. for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ not close, let $J_n := \mathsf{pre}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(C_n) \subseteq \mathsf{J}$. Then $J_n, x[1{:}i], C_n$ is an initial step, which can be decomposed as an initial step $J_n, x[1{:}j], E$ and a step $E, x[j{+}1{:}i], C_n$ such that $|\mathsf{max\text{-}adv}_{J_n,E}^{x[1:j]}| \geqslant 4\Omega!$.

5 Description of the 1-bounded dSST for Subsection 4.2

In this section, we finally describe how the dSST $\mathcal{S}$ can preserve the invariants of Subsection 4.2, while being 1-bounded and outputting $f(x)$ when $x \in \mathsf{Dom}(f)$.

Let us first deal with the initialization of $\mathcal{S}$. When reading the first letter C_0^x of $g(x)$, $\mathcal{S}$ stores $\mathsf{J} \leftarrow C_0^x$, $\mathsf{C} \leftarrow C_0^x$ and $\mathsf{lag}(q) \leftarrow \varepsilon$ for all $q \in C_0^x$. This is enough if C_0^x is not separable. Otherwise, we let θ be given by Lemma 4.13 (applied to the initial simulation $C_0^x, \varepsilon, C_0^x$), $\mathsf{nb}_\pi(q) \leftarrow 0$ and $\mathfrak{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(C_0^x)$ and all $q \in C_n$.

▷ Claim 5.1. Invariants 1 to 4 (with $i = 0$) hold after this operation.

Assume now that the invariants hold for some $x \in \mathsf{Dom}(f)$ and $i \geqslant 0$. We describe how $\mathcal{S}$ updates its information when it reads $x[i{+}1]C_{i+1}^x$. Let $a := x[i{+}1]$.

5.1 If C_i^x was not separable

In this case $\mathcal{S}$ was in the non-separable mode. We update $\mathsf{pre} \leftarrow \mathsf{pre} \circ \mathsf{pre}^a_{C_i^x, C_{i+1}^x}$, $\mathsf{C} \leftarrow C_{i+1}^x$ and $\mathsf{J} \leftarrow \mathsf{pre}(\mathsf{C})$. Since C_i^x, a, C_{i+1}^x was a pre-step, then $\mathsf{J}, x[1{:}i{+}1], \mathsf{C}$ is an initial step. For all $q \in C_{i+1}^x$, let $\delta_q := \mathsf{lag}(\mathsf{pre}^a_{C_i^x, C_{i+1}^x}(q))\, \mathsf{prod}^a_{C_i^x, C_{i+1}^x}(q)$. Now let $c := \bigwedge_{q \in Q} \delta_q$, we update $\mathfrak{out} \leftarrow \mathfrak{out}\, c$ and define $\alpha_q := c^{-1}\delta_q$ for all $q \in \mathsf{C}$. It is easy to see that:

▷ Claim 5.2. $\mathfrak{out} = \mathsf{com}_{\mathsf{J},\mathsf{C}}^{x[1:i+1]}$ and $\alpha_q = \mathsf{adv}_{\mathsf{J},\mathsf{C}}^{x[1:i+1]}(q)$ for all $q \in \mathsf{C}$.

Finally we discuss two cases depending on the separability of $\mathsf{C} = C_{i+1}^x$:

- if C is not separable, then $\mathcal{S}$ stays in non-separable mode and it updates $\mathsf{lag}(q) \leftarrow \alpha_q$ for all $q \in \mathsf{C}$ (note that $|\alpha_q| \leqslant \Omega \leqslant 3\Omega$). We easily see that invariants 1, 2 and 3 hold.
- if C is separable, $\mathcal{S}$ goes to separable mode. By applying Lemma 4.13 to $\mathsf{J}, x[1{:}i{+}1], \mathsf{C}$ we get $\tau \in B^*$ with $k := |\tau| \leqslant 3\Omega$. We update $\mathsf{lag}(q) \leftarrow \alpha_q[1{:}k]$ and $\mathsf{last}(q) \leftarrow \alpha_q[k{+}1{:}]$ for all $q \in \mathsf{C}$. The θ is given by Lemma 4.13, and we let $\mathsf{nb}_\pi(q) \leftarrow 0$ and $\mathfrak{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ (except for $\mathfrak{out}_\pi = \mathfrak{out}$ when $\pi = \mathsf{C} = C_{i+1}^x$) and all $q \in C_n$.

  ▶ **Lemma 5.3.** *Invariants 1, 2 and 4 hold in $i{+}1$ after this operation. Furthermore $|\theta| = \Omega!$, $|\mathsf{lag}(q)| \leqslant 3\Omega$ for all $q \in \mathsf{C}$, and for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, $\mathsf{nb}_\pi = 0$.*

  Note that we may have $|\mathsf{last}(q)| \geqslant \Omega!$. In order to reduce their sizes, we apply the tool detailed in Subsection 5.2 (it will push the $\mathsf{last}(q)$ into the $\mathsf{nb}_\pi(q)$ and $\mathfrak{out}_\pi$).

5.2 Toolbox: reducing the size of $\mathsf{last}(q)$

In this subsection, we assume that $\mathcal{S}$ is in its separable mode and that invariants 2 and 4 hold in some $i \geqslant 0$. Furthermore, we suppose that $|\theta| = \Omega!$, $|\mathsf{lag}(q)| \leqslant 3\Omega$ for all $q \in \mathsf{C}$, and for all $C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, $\mathsf{nb}_{C_1 \cdots C_n} : C_n \to [0{:}4]$. However last may be longer than it should.

From invariant 4c, there exists $n : \mathsf{C} \to \mathbb{N}$ such that $\mathsf{last}(q) = \theta^{n(q)}\delta_q$ with $\delta_q \sqsubset \theta$ for all $q \in \mathsf{C}$. We update $\mathsf{last}(q) \leftarrow \delta_q$ and $\mathsf{nb}_\mathsf{C}(q) \leftarrow \mathsf{nb}_\mathsf{C}(q) + n(q)$ for all $q \in \mathsf{C}$. Now, we have $|\mathsf{last}(q)| < \Omega!$ and $\mathsf{nb}_\pi(q) \leqslant 4$ when $\pi \neq \mathsf{C}$.

In order to reduce the value nb_C, we apply the function **down**(C) of Algorithm 1 which adds some θ in the $\mathfrak{out}_\pi$. Let us describe its base case informally. If $\mathsf{nb}_\mathsf{C}(q) > 0$ for all $q \in \mathsf{C}$, then no state is lagging by invariant 4d. Thus $\mathsf{lag}(q) = \mathsf{max\text{-}lag}$ for all $q \in \mathsf{C}$, and so $\mathsf{max\text{-}lag} = \varepsilon$ by invariant 4a. With the notations of invariant 4e (note that $\pi_1 = \mathsf{C}$), we get $\mathsf{prod}^{\mathsf{J},\mathsf{C}}_{x[1:i]}(q) = \mathfrak{out}\ \theta^{\mathsf{nb}_{\pi_1}(q)} \left(\prod_{i=2}^n \mathfrak{out}_{\pi_i}\, \theta^{\mathsf{nb}_{\pi_i}(q)}\right) \mathsf{last}(q)$ for all $q \in \mathsf{C}$. Thus we can produce in $\mathfrak{out}$ the value $\theta^m := \bigwedge_{q \in \mathsf{C}} \theta^{\mathsf{nb}_\mathsf{C}(q)}$ (i.e. $m \leftarrow \min_{q \in \mathsf{C}} \mathsf{nb}_\mathsf{C}(q)$) and remove m to each $\mathsf{nb}_\mathsf{C}(q)$.

Algorithm 1 Sending down values in $\mathsf{tree}(\mathsf{C})$.

```
Function down($\pi$)
    $C_1 \cdots C_n \leftarrow \pi$;
    /* 1.  Add the common part of the buffers to the local output      */
    $m \leftarrow \min_{q \in C_n} \mathsf{nb}_\pi(q)$;
    $\mathfrak{out}_\pi \leftarrow \mathfrak{out}_\pi\, \theta^m$;
    $\mathsf{nb}_\pi(q) \leftarrow \mathsf{nb}_\pi(q) - m$  for all  $q \in C'$;
    /* 2.  Check if some buffers $\mathsf{nb}_\pi(q)$ are still more than $4$      */
    for $q \in C_n$ do
        if $\mathsf{nb}_\pi(q) > 4$ then
            for $C' \in \mathsf{Comp}(C_n)$ such that $C' \neq C_n$ and $q \in C'$ do
                $\mathsf{nb}_{\pi C'}(q) \leftarrow \mathsf{nb}_{\pi C'}(q) + (\mathsf{nb}_\pi(q) - 4)$;
            end
            $\mathsf{nb}_\pi(q) \leftarrow 4$;
        end
    end
    /* 3.  Recursive calls                                              */
    for $C' \in \mathsf{Comp}(C_n)$ with $C' \neq C_n$ do
        down($\pi C'$);
    end
```

▶ **Lemma 5.4.** *Algorithm 1 is well defined. After the operation described in this subsection, invariants 2 and 4 hold, and furthermore we have* $|\theta| = \Omega!$, $|\mathsf{lag}(q)| \leqslant 3\Omega$ *and* $|\mathsf{last}(q)| < \Omega!$ *for all* $q \in \mathsf{C}$, *and for all* $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, $\mathsf{nb}_\pi : C_n \to [0{:}4]$.

5.3 If C_i^x was separable

If C_i^x is separable, then $\mathcal{S}$ was in the separable mode by invariant 4. We first explain in Subsubsection 5.3.1 how to perform the update when C, a, C_{i+1}^x is a step (it corresponds to the "easy case" thanks to invariant 4f which deals with future steps). Then, we explain in Subsubsection 5.3.2 how the other case can be reduced to the first one, after a preprocessing which selects a subset $C' \subseteq \mathsf{C}$ such that C', a, C_{i+1}^x is a step.

5.3.1 Updating when C, a, C_{i+1}^x is a step

In the current subsubsection we assume that invariants 2 and 4 hold, that $\mathsf{C} \subsetneq C_i^x$ is separable, and that C, a, C_{i+1}^x is a step. We show how to update the information stored by $\mathcal{S}$ in accordance with this step. Note that C_{i+1}^x is necessarily separable.

Since C will be modified, so will be $\mathsf{tree}(\mathsf{C})$, hence we begin with several register updates. For $\pi = D_1 \cdots D_n \in \mathsf{tree}(C_{i+1}^x)$, we define $C_i := \mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(D_i)$ for $1 \leqslant i \leqslant n$. Since we had a step then $C_1 = \mathsf{C}$, $C_i \in \mathsf{Comp}(\mathsf{C})$ and $C_1 \supseteq \cdots \supseteq C_n$. But we may not have $C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ due to possible equalities. Let $1 = i_1 < \cdots < i_m \leqslant n$ be such that $C_{i_1} = \cdots = C_{i_2 - 1} \supset C_{i_2}$ and so on until $C_{i_m - 1} \supset C_{i_m} = \cdots = C_n$. Then $\rho := C_{i_1} \cdots C_{i_m} \in \mathsf{tree}(\mathsf{C})$ and:

- if $i_m = n$, we let $\mathsf{nb}_\pi \leftarrow \mathsf{nb}_\rho \circ \mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}$ and $\mathfrak{out}_\pi \leftarrow \mathfrak{out}_\rho$;
- if $i_m < n$, we let $\mathsf{nb}_\pi \leftarrow 0$ and $\mathfrak{out}_\pi \leftarrow \varepsilon$.

For all $q \in C_{i+1}^x$, let $k_q := |\mathsf{lag}(\mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(q))^{-1}\, \mathsf{max\text{-}lag}\,|$ and:

- $\mathsf{lag}(q) \leftarrow \mathsf{lag}(\mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(q))(\mathsf{prod}^a_{\mathsf{C}, C_{i+1}^x}(q)[1{:}k_q])$ (note that $\mathsf{max\text{-}lag}$ remains unchanged);
- $\mathsf{last}(q) \leftarrow \mathsf{last}(\mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(q))(\mathsf{prod}^a_{\mathsf{C}, C_{i+1}^x}(q)[k_q{+}1{:}])$.

Now let $c := \bigwedge_{q \in \mathsf{C}} \mathsf{lag}(q)$. We update $\mathsf{lag}(q) \leftarrow c^{-1}\,\mathsf{lag}(q)$ for all $q \in \mathsf{C}$ (therefore $\mathsf{max\text{-}lag}$ becomes $c^{-1}\,\mathsf{max\text{-}lag}$), $\mathfrak{out} \leftarrow \mathfrak{out}\ \ c$, $\mathsf{C} \leftarrow C^x_{i+1}$ and finally $\mathsf{pre} \leftarrow \mathsf{pre} \circ \mathsf{pre}^a_{\mathsf{C},C^x_{i+1}}$.

▶ **Lemma 5.5.** *After the operation described in this subsection, invariants 1, 2 and 4 hold, and* $|\theta| = \Omega!$, $|\,\mathsf{lag}(q)| \leqslant \Omega$ *for all* $q \in \mathsf{C}$, *and* $\mathsf{nb}_\pi : C_n \to [0{:}4]$ *for all* $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$.

However, we may have $|\,\mathsf{last}(q)| \geqslant \Omega!$. Thus we finally apply Subsection 5.2 once more.

5.3.2 Preprocessing when C, a, C^x_{i+1} is not a step

In the current subsubsection we assume that invariants 1, 2 and 4 hold in $i \geqslant 0$, that $\mathsf{C} = C^x_i$ is separable, and that C^x_i, a, C^x_{i+1} is *not* a step. Then let $C' := \mathsf{pre}^a_{C^x_i,C^x_{i+1}}(C^x_{i+1}) \subset \mathsf{C}$ (an equality would give a step) and $\pi := \mathsf{C}\,C' \in \mathsf{tree}(C)$. Two cases can occur.

If π is close. In this case, we have for all $\pi \sqsubset \pi' \in \mathsf{tree}(\mathsf{C})$ that $\mathsf{nb}_{\pi'} = 0$ and $\mathfrak{out}_{\pi'} = \varepsilon$. Therefore by invariant 4e we can describe the productions for all $q \in C'$ as follows:

- $\mathsf{prod}^{\mathsf{J},\mathsf{C}}_{x[1:i]}(q) = \mathfrak{out}\ \ \mathsf{lag}(q)$ if q is lagging;
- $\mathsf{prod}^{\mathsf{J},\mathsf{C}}_{x[1:i]}(q) = \mathfrak{out}\ \ \mathsf{max\text{-}lag}\ \ \mathfrak{out}_{\mathsf{C}\,C'}\ \ \theta^{\mathsf{nb}_\mathsf{C}(q)+\mathsf{nb}_{\mathsf{C}\,C'}(q)}\ \ \mathsf{last}(q)$ if q is not lagging.

Now two cases are possible, depending on whether there is a lagging state in C' or not:

- if there exists $q' \in C'$ which is lagging, then we must have $\mathfrak{out}_{\mathsf{C}\,C'} = \varepsilon$ by invariant 4d. For all $q \in C'$ let $\delta_q := \mathsf{lag}(q)\theta^{\mathsf{nb}_\mathsf{C}(q)+\mathsf{nb}_{\mathsf{C}\,C'}(q)}\ \ \mathsf{last}(q)$ and let $c := \bigwedge_{q \in C'} \delta_q$. Then we update $\mathfrak{out} \leftarrow \mathfrak{out}\ \ c$ and define $\alpha_q := c^{-1}\delta_q$ for all $q \in C'$;
- if each $q \in C'$ is not lagging, we define $\delta_q := \theta^{\mathsf{nb}_\mathsf{C}(q)+\mathsf{nb}_{\mathsf{C}\,C'}(q)}\,\mathsf{last}(q)$ and $c := \bigwedge_{q \in C'} \delta_q$. Then we update $\mathfrak{out} \leftarrow \mathfrak{out}\ \ \mathsf{max\text{-}lag}\ \ \mathfrak{out}_{\mathsf{C}\,C'}\ \ c$ and define $\alpha_q := c^{-1}\delta_q$ for all $q \in C'$;

We finally update $\mathsf{J} \leftarrow \mathsf{pre}(C')$, $\mathsf{C} \leftarrow C'$ and $\mathsf{pre} \leftarrow \mathsf{pre}|_{C'}$. It is easy to see that $\mathsf{J}, x[1{:}i], \mathsf{C}$ is a step and furthermore that we have computed com and adv.

▷ Claim 5.6. $\mathfrak{out} = \mathsf{com}^{x[1:i+1]}_{\mathsf{J},\mathsf{C}}$ and $\alpha_q = \mathsf{adv}^{x[1:i+1]}_{\mathsf{J},\mathsf{C}}(q)$ for all $q \in \mathsf{C}$.

This result exactly corresponds to Claim 5.2 from Subsection 5.1 (replace $i{+}1$ by i). Thus, to conclude, we just need to apply the operations described after Claim 5.2.

If π is not close. Let $c := \bigwedge_{q \in C'} \mathsf{lag}(q)$, we update $\mathfrak{out} \leftarrow \mathfrak{out}\ \ c\ \ \mathfrak{out}_{\mathsf{C}\,C'}$ and for all $q \in C'$, $\mathsf{lag}(q) \leftarrow c^{-1}\,\mathsf{lag}(q)$ and $\mathsf{last}(q) \leftarrow \theta^{\mathsf{nb}_\mathsf{C}(q)}\,\mathsf{last}(q)$. Then, we update $\mathsf{nb}_{C'\pi} \leftarrow \mathsf{nb}_{\mathsf{C}\,C'\pi}$ and $\mathfrak{out}_{C'\pi} \leftarrow \mathfrak{out}_{\mathsf{C}\,C'\pi}$ for all $\pi \in (C')^{-1}\,\mathsf{tree}(C')$ (except for $\pi = \varepsilon$, in which case we have already updated $\mathfrak{out}_{C'} = \mathfrak{out}$ before). We finally update $\mathsf{J} \leftarrow \mathsf{pre}(C')$, $\mathsf{C} \leftarrow C'$ and $\mathsf{pre} \leftarrow \mathsf{pre}|_{C'}$.

▶ **Lemma 5.7.** *After the operation described in this subsection, invariants 2 and 4 hold, and* $|\theta| = \Omega!$, $|\,\mathsf{lag}(q)| \leqslant \Omega$ *for all* $q \in \mathsf{C}$, *and* $\mathsf{nb}_\pi : C_n \to [0{:}4]$ *for all* $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$. *Furthermore* C *is separable.*

▶ Remark 5.8. Contrary to the former cases, the main difficulty here is to show the preservation of invariant 4f. For this we essentially rely on invariant 4g and show that θ is still suitable.

Again, we may have $|\,\mathsf{last}(q)| \geqslant \Omega!$. Thus we finally apply Subsection 5.2 once more.

5.4 Boundedness and productivity of the construction

We first claim that $\mathcal{S}$ is a 1-bounded dSST, by construction.

▶ **Lemma 5.9.** *The* dSST $\mathcal{S}$ *is* 1*-bounded.*

It follows from invariants 1, 2 and 4e that for all $x \in \mathsf{Dom}(f)$, $\mathfrak{out}$ is always a prefix of $f(x)$ when $\mathcal{S}$ reads $g(x)$. To conclude the construction of $\mathcal{S}$, it remains to see that $\mathfrak{out}$ tends to an infinite word. The key ideas for showing Lemma 5.10 is to use the fact that $\mathcal{T}$ is productive, and that Algorithm 1 can only empty a buffer $\mathsf{nb}_{\mathsf{C}}(q)$ if it outputs a word.

▶ **Lemma 5.10.** *If $x \in \mathsf{Dom}(f)$, then $|\mathfrak{out}| \to \infty$ when $\mathcal{S}$ reads $g(x)$.*

6 Outlook

This paper provides a solution to an open problem. From a practical point of view, it allows to build a copyless streaming algorithm from a rational specification whenever it is possible (it is impossible when the rational function is not continuous). We conjecture that the techniques introduced in this paper can be extended to show that any continuous *regular* function is deterministic regular. Furthermore, they may also be used to study the rational or regular functions which are uniformly continuous for the Cantor topology, and capture them with a specific transducer model (another open problem of [5]).

References

1 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl, 2010.

2 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 65–74. IEEE Computer Society, 2012.

3 Christian Choffrut and Serge Grigorieff. Uniformization of rational relations. In *Jewels are Forever*, pages 59–71. Springer, 1999.

4 Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic string transducers. *Int. J. Found. Comput. Sci.*, 29(5):801–824, 2018.

5 Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Synthesis of computable regular functions of infinite words. In *31st International Conference on Concurrency Theory (CONCUR 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

6 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, 2020.

7 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.

8 Emmanuel Filiot and Sarah Winter. Synthesizing computable functions from rational specifications over infinite words. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

9 Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games.* Academic Press, 2004.

10 Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theoretical Computer Science*, 250(1-2):71–82, 2001.

On Kernels for d-Path Vertex Cover

Radovan Červený
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic

Pratibha Choudhary
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic

Ondřej Suchý
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic

Abstract

In this paper we study the kernelization of the d-Path Vertex Cover (d-PVC) problem. Given a graph G, the problem requires finding whether there exists a set of at most k vertices whose removal from G results in a graph that does not contain a path (not necessarily induced) with d vertices. It is known that d-PVC is NP-complete for $d \geq 2$. Since the problem generalizes to d-Hitting Set, it is known to admit a kernel with $\mathcal{O}(dk^d)$ edges. We improve on this by giving better kernels. Specifically, we give kernels with $\mathcal{O}(k^2)$ vertices and edges for the cases when $d = 4$ and $d = 5$. Further, we give a kernel with $\mathcal{O}(k^4 d^{2d+9})$ vertices and edges for general d.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized complexity, Kernelization, d-Hitting Set, d-Path Vertex Cover, Expansion Lemma

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.29

Related Version *Full Version*: https://arxiv.org/abs/2107.12245

Funding *Radovan Červený*: The author acknowledges the support of the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/208/OHK3/3T/18.
Pratibha Choudhary: The author acknowledges the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".
Ondřej Suchý: The author acknowledges the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

1 Introduction

Vertex deletion problems have been studied extensively in graph theory. These problems require finding a subset of vertices whose deletion results in a graph that belongs to some desired class of graphs. One such problem is path covering. Given a graph $G = (V, E)$, the d-Path Vertex Cover problem (d-PVC) asks to compute a subset $S \subseteq V$ of vertices such that the graph resulting from removal of S does not contain a path on d vertices. Here the path need not necessarily be induced. The problem was first introduced by Brešar et al. [1]. It is known to be NP-complete for any $d \geq 2$ due to the meta-theorem of Lewis and Yannakakis [21]. The 2-PVC problem is the same as the well known Vertex Cover problem. The 3-PVC problem is also known as Maximum Dissociation Set or Bounded Degree-One Deletion. The d-PVC problem is motivated by the field of designing secure wireless communication protocols [22] or in route planning and speeding up of shortest path queries [18].

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 29; pp. 29:1–29:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

With respect to exact algorithms, several efficient (better than brute force enumeration) algorithms are known for 2-PVC and 3-PVC. In particular, 2-PVC (Vertex Cover) can be solved in $\mathcal{O}(1.1996^n)$ time and polynomial space due to Xiao and Nagamochi [33], while 3-PVC can be solved in $\mathcal{O}(1.4613^n)$ time and polynomial space due to Chang et al. [6] or in $\mathcal{O}(1.3659^n)$ time and exponential space due to Xiao and Kou [31].

From the approximation point of view, it is known due to Brešar et al. [1] that d-PVC, for $d > 2$, cannot be r-approximated within a factor of $r = 1.3606$ in polynomial time, unless P=NP. A greedy d-approximation algorithm for d-PVC can be employed by repeatedly finding a d-path and putting its vertices into the solution. Due to Fomin et al. [17], we can find an arbitrary d-path in $\mathcal{O}(2.619^d dn \log n)$ time, and therefore the approximation algorithm runs in $\mathcal{O}(n^2 \log n)$ time in the size of the input. While the algorithms of Zehavi [34] and Tsur [25], with running times $\mathcal{O}^*(2.597^d)$ and $\mathcal{O}^*(2.554^d)$,[1] respectively, can be faster for large d, their running time factor polynomial in input size is much worse than $\mathcal{O}(n \log n)$. Lee [20] gave a $\mathcal{O}(\log d)$-approximation algorithm which runs in $\mathcal{O}^*(2^{\mathcal{O}(d^3 \log d)})$ time. For 3-PVC a 2-approximation algorithm was given by Tu and Zhou [30] and for 4-PVC a 3-approximation algorithm is known due to Camby et al. [2].

When parameterized by the size of the solution k, d-PVC is directly solvable by a trivial FPT algorithm for d-Hitting Set, that runs in $\mathcal{O}^*(d^k)$ time. However, since d-PVC is a restricted case of d-Hitting Set, it is known due to Fomin et al. [15] that for $d \geq 4$ d-PVC can be solved in $\mathcal{O}^*((d - 0.9245)^k)$ time and for $d \geq 6$ algorithms with even better running times are known due to Fernau [14]. Namely the running times are $\mathcal{O}^*((d - 1 + c_d)^k)$, where c_d is a small positive constant which monotonically approaches 0 as d goes to ∞. There has been considerable study for the case when d is a small constant. For the 2-PVC (Vertex Cover) problem, current best known algorithm due to Chen, Kanj, and Xia [7] runs in time $\mathcal{O}^*(1.2738^k)$. For 3-PVC, the current best known algorithm due to Tsur [27] runs in $\mathcal{O}^*(1.713^k)$ time. For the 4-PVC problem, Tsur [28] gave the current best algorithm that runs in $\mathcal{O}^*(2.619^k)$ time. In previous work [3, 4], a subset of authors developed an $\mathcal{O}^*(4^k)$ algorithm for 5-PVC. For $d = 5$, 6, and 7 Tsur [26] claimed algorithms for d-PVC with running times $\mathcal{O}^*(3.945^k)$, $\mathcal{O}^*(4.947^k)$, and $\mathcal{O}^*(5.951^k)$, respectively. A subset of authors used a computer to generate even faster algorithms for $3 \leq d \leq 8$ [5].

In this paper, we are interested in kernels for the d-PVC problem. Since an instance of d-PVC can be formulated as an instance of d-Hitting Set, by using the results of Fafianie and Kratsch [13] we immediately get a kernel for d-PVC with at most $d(k+1)^d$ vertices and at most $(d-1)(k+1)^d$ edges by keeping only the vertices and edges that are contained in the corresponding sets of the reduced d-Hitting Set instance.

Regarding the lower bounds for kernels of d-PVC, Dell and Melkebeek [10] have shown that for Vertex Cover it is not possible to achieve a kernel with $\mathcal{O}(k^{2-\varepsilon})$ edges unless coNP is in NP/poly (which would imply a collapse of the polynomial hierarchy). This result extends to d-PVC for any $d \geq 2$. Therefore, kernels with $\mathcal{O}(k^2)$ edges for d-PVC are the best we can hope for.

The current best kernels known are a kernel for Vertex Cover with $2k - c \log k$ vertices for any fixed constant c due to Lampis [19] and a kernel with $5k$ vertices for 3-PVC due to Xiao and Kou [32]. No specific kernels are known for d-PVC with $d \geq 4$, except for those inherited from d-Hitting Set.

Dell and Marx [9] recently studied kernels for the related d-Path Packing problem, which also inspired our work.

[1] The $\mathcal{O}^*()$ notation suppresses all factors polynomial in the input size.

Our contribution

We give kernels with $\mathcal{O}(k^2)$ edges for 4-PVC and 5-PVC (asymptotically optimal, unless coNP $\subseteq$ NP/poly). Furthermore, for the general case, we give a kernel for d-PVC for any $d \geq 6$ with $\mathcal{O}(k^4 d^{2d+9})$ edges.

2 Preliminaries

We use the notations related to parameterized complexity as described by Cygan et al. [8]. We consider simple and undirected graphs unless otherwise stated. For a graph G, we use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G. By $G[X]$ we denote the subgraph of G induced by vertices of $X \subseteq V(G)$. By $N(v)$ we denote the set of neighbors of $v \in V(G)$ in G. Analogically, $N(X) = \bigcup_{x \in X} N(x) \setminus X$ denotes the set of neighbors of vertices in $X \subseteq V(G)$. The degree of vertex v is denoted by $\deg(v) = |N(v)|$. For simplicity, we write $G \setminus v$ for $v \in V(G)$ and $G \setminus X$ for $X \subseteq V(G)$ as shorthands for $G[V(G) \setminus \{v\}]$ and $G[V(G) \setminus X]$, respectively.

A *d-path* (also denoted by P_d), denoted as an ordered d-tuple $(p_1, p_2, \ldots, p_d)$, is a path on d vertices $\{p_1, p_2, \ldots, p_d\}$. A *$d$-path free* graph is a graph that does not contain a d-path as a subgraph (the d-path needs not to be induced). The *length* of a path P is the number of edges in P, in particular, the length of a d-path P_d is $d-1$.

The d-PATH VERTEX COVER problem is formally defined as follows:

d-PATH VERTEX COVER, d-PVC	
INPUT:	A graph $G = (V, E)$, a non-negative integer k.
OUTPUT:	A set $S \subseteq V$, such that $\lvert S \rvert \leq k$ and $G \setminus S$ is a P_d-free graph.

A d-path packing $\mathcal{P}$ of size l in a graph G is a collection of l vertex disjoint d-paths in the graph G. We use $V(\mathcal{P})$ to denote the union of the vertex sets of the d-paths in the packing $\mathcal{P}$. For rest of the graph theory notations we refer to Diestel [11].

For a positive integer i, we will use $[i]$ to denote the set $\{1, 2, \ldots, i\}$.

▶ **Proposition 1** ($\star$). *For a given graph G and an integer k, there is an algorithm which either correctly answers whether G has a d-path vertex cover of size at most k, or finds an inclusion-wise maximal d-path packing $\mathcal{P}$ of size at most k in $\mathcal{O}(2.619^d dkn \log n)$ time.*

3 General Reduction Rules

Let us start with reduction rules that apply to d-PVC for most values of d. Assume that we are working with an instance $(G = (V, E), k)$ of d-PVC for some $d \geq 4$. We start with a reduction rule whose correctness is immediate.

▶ **Reduction Rule 1.** *If there is a connected component C in G which does not contain a P_d, then remove C.*

The next rule allows us to get rid of multiple degree-one vertices adjacent to a single vertex.

▶ **Reduction Rule 2** ($\star$). *Let there be three distinct vertices $v, x, y \in V$ such that $N(x) = N(y) = \{v\}$. We reduce the instance by deleting the vertex x.*

4 High Degree Reduction Rule for 4-PVC and 5-PVC

In this section, we are going to introduce the reduction rules which are applicable to both 4-PVC and 5-PVC instances. We assume that we are working with a d-PVC instance $(G = (V, E), k)$ for $d \in \{4, 5\}$ which is reduced by exhaustively employing Reduction Rule 2.

Our aim is to show that the degree of each vertex can be reduced to linear in the parameter. First assume that there is a large matching in the neighborhood of some vertex v. We call a matching $\mathcal{M}$ in G *adjacent* to vertex v, if it is a matching in $G \setminus v$ and for each edge $\{a_i, b_i\} \in \mathcal{M}$ at least one of its vertices, say a_i, is adjacent to v in G.

▶ **Reduction Rule 3** (⋆). *If v is a vertex and $\mathcal{M}$ a matching adjacent to v of size $|\mathcal{M}| \geq k+2$, then delete v and decrease k by* 1.

To exhaustively apply Reduction Rule 3, we need to find for each $v \in V$ a largest matching adjacent to v. This can be done as follows. Let $A = N(v)$ and $B = N(A) \setminus \{v\}$. Let G_v be the graph obtained from $G[A \cup B]$ by removing edges with both endpoints in B. It is easy to observe, that each matching adjacent to v is also a matching in G_v and vice-versa. Hence, it suffices to find a largest matching in G_v, which can done in polynomial time [12].

Therefore, we further assume that the instance is reduced with respect to Reduction Rule 3. We fix a vertex v and find a largest matching $\mathcal{M}$ adjacent to it by the above algorithm. Let M be the set of vertices covered by matching $\mathcal{M}$ and $m = |\mathcal{M}|$. Since the instance is reduced, we know that $m \leq k + 1$. Let $X = N(v) \setminus M$. We refer the reader to the Figure 1 for overview of our setting.

▶ **Observation 2** (⋆). *For each $x \in X$ we have $N(x) \setminus \{v\} \subseteq M$.*

▶ **Observation 3** (⋆). *No two distinct vertices $x, y \in X$ are connected to the opposite endpoints of a single edge $\{a_i, b_i\}$ in $\mathcal{M}$.*

▶ **Observation 4** (⋆). *If there is a vertex $x \in X$ such that for some edge $\{a_i, b_i\}$ in the matching $\mathcal{M}$ we have that $\{a_i, b_i\} \subseteq N(x)$, then $N(\{a_i, b_i\}) \cap X = \{x\}$.*

We now partition the set X into three sets. Let X_2 be the set of vertices such that for each $x \in X_2$ we have some edge $\{a_i, b_i\}$ in the matching $\mathcal{M}$ such that $\{a_i, b_i\} \subseteq N(x)$. Let X_0 be the vertices such that for each $x \in X_0$ we have that $N(x) = \{v\}$. Note, that X_0 contains at most one vertex due to Reduction Rule 2 being exhaustively applied. Lastly, let $X_1 = X \setminus (X_2 \cup X_0)$ be the rest of the vertices in X. See Figure 1 for an illustration of the sets X_2, X_0, and X_1.

▶ **Observation 5** (⋆). *If the vertex v has degree at least $(d + 2)(k + 1) + 1$, then $|X_1| \geq (d - 1)(k + 1)$.*

Now we focus on the edges between X_1 and M. By Observation 3, for each edge $\{a_i, b_i\}$ in $\mathcal{M}$ we have that the vertices in X_1 may be adjacent to at most one vertex of such edge, i.e. $|\{a_i, b_i\} \cap N(X_1)| \leq 1$. Letting $M_1 = M \cap N(X_1)$ we have $|M_1| \leq k + 1$.

We are now ready to employ the Expansion Lemma. We use the version of Fomin et al. [16], which is a generalization of the original results by Prieto [23, Corollary 8.1] and Thomassé [24, Theorem 2.3].

▶ **Definition 6.** *Let G be a bipartite graph with vertex bipartition (A, B). A set of edges $Q \subseteq E(G)$ is called a q-expansion, $q \geq 1$, of A into B if every vertex of A is incident with exactly q edges of Q, and Q saturates exactly $q|A|$ vertices in B.*

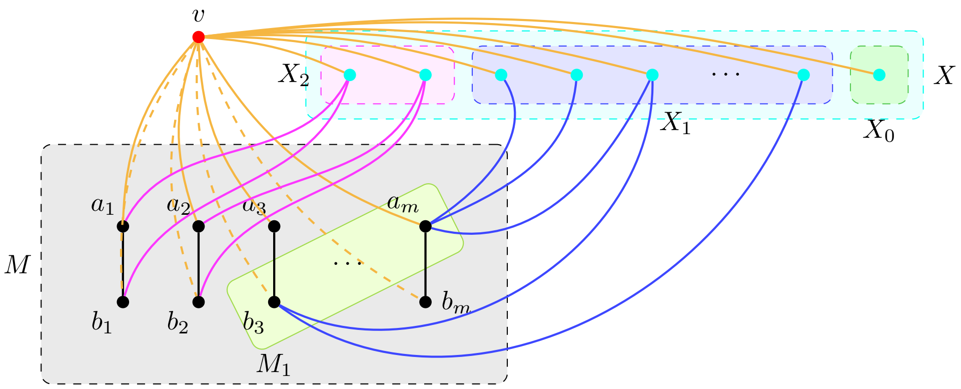


Figure 1 An overview of the definitions of sets X_0, X_1, X_2, and M_1.

▶ **Lemma 7** (Expansion Lemma; Fomin et al. [16]). *Let q be a positive integer, and G be a bipartite graph with bipartition (A, B) such that $|B| \geq q|A|$, and there are no isolated vertices in B. Then, there exists nonempty $A' \subseteq A$ and $B' \subseteq B$ such that A' has a q-expansion into B' and $N(B') \subseteq A'$. Moreover, the sets A', B' and the q-expansion can be found in polynomial time.*

▶ **Observation 8** ($\star$). *There exist non-empty subsets $M' \subseteq M_1$ and $X' \subseteq X_1$ such that there is a $(d-1)$-expansion Q' from M' into X' and $N(X') \subseteq M' \cup \{v\}$.*

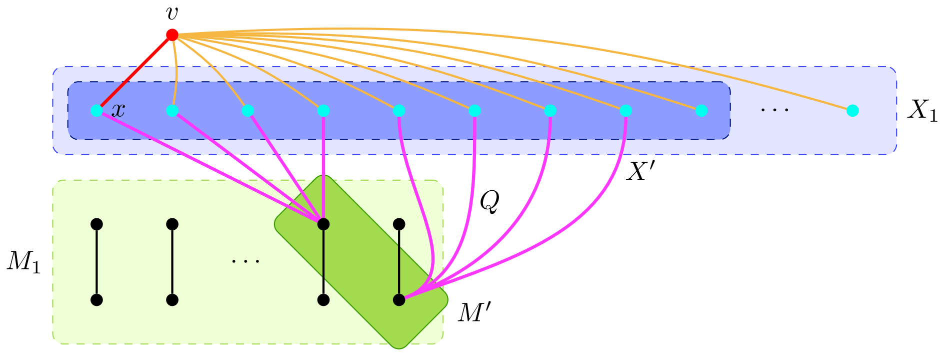


Figure 2 A graphical interpretation of applying the Expansion Lemma to our setting.

We refer the reader to Figure 2 for a graphical interpretation of the situation guaranteed by Observation 8. Now, let us focus on the sets M' and X' and the way they are connected with vertex v. We are going to show that some edge between v and X' is now redundant.

▶ **Reduction Rule 4.** *Let v be a vertex of degree at least $(d+2)(k+1)+1$. Let $\mathcal{M}$ be a largest matching adjacent to v and M be the set of vertices covered by $\mathcal{M}$. Let $X_1 \subseteq N(v) \setminus M$ be the set of vertices x with $N(x) \cap M \neq \emptyset$ and $|N(x) \cap \{a_i, b_i\}| \leq 1$ for each $\{a_i, b_i\} \in \mathcal{M}$. Let $M_1 = M \cap N(X_1)$. Let the non-empty subsets $M' \subseteq M_1$ and $X' \subseteq X_1$ be the sets with the $(d-1)$-expansion Q from M' into X' and such that $N(X') \subseteq M' \cup \{v\}$. Let $x \in X'$. Reduce the instance by deleting the edge $\{x, v\}$.*

Proof of Correctness. Let $(G = (V, E), k)$ be the original instance and $(G' = (V, E'), k)$ the reduced one. For each vertex $m \in M'$ let Q_m be the set of vertices of X' incident to m in the $(d-1)$-expansion Q. Since G' is a subgraph of G, if S is a solution for G, then S is also a solution for G'. Hence we will concetrate on the other direction.

Suppose that S' is a solution for the reduced instance. If it is also a solution for the original one, then we are done. Suppose it is not, i.e., there is a P_d in $G \setminus S'$. This P_d contains the edge $\{x, v\}$, otherwise it would be also present in $G' \setminus S'$. Therefore $v \notin S'$ and $x \notin S'$.

There are three ways how solution S' can interact with M' and X' that we need to address.

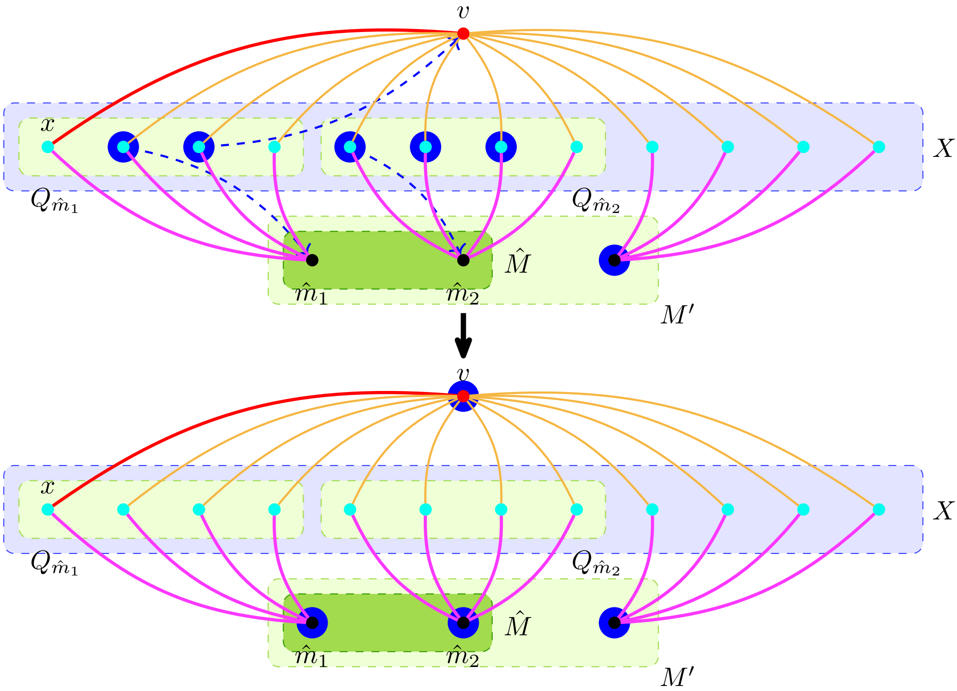


Figure 3 Illustration of the first part of the proof of correctness of Reduction Rule 4.

Firstly, suppose that $M' \nsubseteq S'$ and let $\widehat{M} = M' \setminus S'$. See Figure 3 for an illustration. We have that for each vertex $\widehat{m} \in \widehat{M}$ it must be that $|Q_{\widehat{m}} \cap S'| \geq 2$. Indeed, if this is not the case, there would be a P_d in $G' \setminus S'$ which uses the $(d-2)$ vertices of $Q_{\widehat{m}}$ not in S' and the vertices $\widehat{m}$ and v. Consider a set $S = (S' \cup \widehat{M} \cup \{v\}) \setminus \bigcup_{\widehat{m} \in \widehat{M}} Q_{\widehat{m}}$. Observe, that any P_d which uses some vertex from X' must contain at least one of the vertices in M' or v, because $N(X') \subseteq M' \cup \{v\}$. Therefore the set S is a solution for the reduced graph G' as it contains both M' and v. The set S is also a solution for G as it contains v and therefore covers any P_d which might use the deleted edge $\{x, v\}$. Finally, $|S| \leq |S'|$, because for each $\widehat{m}$ that we add into S, we remove at least two vertices of $Q_{\widehat{m}}$ from S and therefore we have that $|\widehat{M} \cup \{v\}| \leq 2|\widehat{M}| \leq |\bigcup_{\widehat{m} \in \widehat{M}} Q_{\widehat{m}}|$.

Secondly, assume that $M' \subseteq S'$ and there is some $x' \in X'$ such that $x' \in S'$. We construct the set $S = (S' \setminus X') \cup \{v\}$. Again, observe that S' is a solution for G' because any P_d which uses some vertex from X' must contain at least one of the vertices in M' or v and both are fully contained in S. We also have that S is a solution for G, again, as it contains v and therefore covers any P_d which might use the deleted edge $\{x, v\}$. Finally, $|S| \leq |S'|$, because we have the assumption that there is some $x' \in X'$ and $x' \in S'$.

Lastly, assume that $M' \subseteq S'$ and $X' \cap S' = \emptyset$. In this case, the P_d that we found in $G \setminus S'$ must be of the form $P = (x, v, u_3, \ldots, u_d)$ and $u_3, \ldots, u_d \notin (M' \cup X')$. The existence of such P_d also gives us that $v, u_3, \ldots, u_d \notin S'$. Let x' be an arbitrary vertex of $X' \setminus \{x\}$ (note that $|X'| \geq (d-1)|M'| \geq 3$). Then the P_d of the form $P' = (x', v, u_3, \ldots, u_d)$ can be found in both G and G', which contradicts the fact that S' is a solution in G'.

To sum up, we have shown that when we delete the edge $\{x, v\}$ from G, then for any solution S' for G' which is not also a solution for G, we can always find a new solution S, $|S| \leq |S'|$ which is a solution for both G' and G. ◀

As the application of the rule only requires finding a largest matching adjacent to v, classifying the vertices of $N(v)$, and finding a $(d-1)$-expansion and these tasks can be done in polynomial time, the rule can be applied in polynomial time.

5 4-PVC Kernel with Quadratic Number of Edges

Let $(G = (V, E), k)$ be an instance reduced by exhaustively employing Reduction Rules 1–4. Then the maximum degree in G is at most $(d + 2)(k + 1) = 6k + 6$. Furthermore, assume that the algorithm of Proposition 1 actually returned an inclusion-wise maximal packing $\mathcal{P}$ in G with at most k 4-paths instead of answering immediately. Let $P = V(\mathcal{P})$, and let $A = V \setminus P$. There are at most $4k$ vertices in P. Each connected component in $G[A]$ is a 4-path free graph, otherwise we would be able to increase the size of the packing $\mathcal{P}$. Since the instance is reduced with respect to Reduction Rule 1, each connected component in $G[A]$ is connected to P by at least one edge.

To show that an instance reduced with respect to all the above rules has a quadratic number of edges, it suffices to count separately the number of edges incident on P and the number of edges in $G[A]$. Since the maximum degree in G is at most $6k + 6$ and there are at most $4k$ vertices in P, there are at most $4k \cdot (6k + 6) = 24k^2 + 24k$ edges incident on P.

To count the edges in $G[A]$, we first observe that a connected 4-path free graph is either a triangle, or a star (possibly degenerate, i.e., with at most 3 vertices). Here a *q-star* is a graph with vertices $\{c, l_1, \ldots, l_q\}$, $q \geq 0$ and edges $\{\{c, l_i\} \mid i \in \{1, \ldots, q\}\}$. Vertex c is called *a center*, vertices $\{l_1, \ldots, l_q\}$ are called *leaves*. The term *star* will be used for a q-star with an arbitrary number of leaves. Note that, a graph with a single vertex is a 0-star, a graph with two vertices and a single edge is a 1-star, and a 3-path is a 2-star. A *triangle* is a cycle on three vertices.

Secondly, as the instance is reduced with respect to Reduction Rule 1, each connected component in $G[A]$ is connected to P by at least one edge. Therefore, there are at most $24k^2 + 24k$ connected components in $G[A]$, as there are only that many edges going from P to A. Next, we provide an observation about stars in $G[A]$.

▶ **Observation 9** (⋆). *For each q-star $(c, l_1, l_2, \ldots, l_q)$ in $G[A]$, there are at most two vertices in the q-star which are not connected to P by any edge in G, one possibly being the center c and the other possibly being a leaf l_i of the star.*

▶ **Observation 10** (⋆). *There are at most $72k^2 + 72k$ edges in $G[A]$.*

We conclude this section with the final statement about our kernel.

▶ **Theorem 11** (⋆). *4-Path Vertex Cover admits a kernel with $96k^2 + 96k$ edges, where k is the size of the solution.*

6 5-PVC Kernel with Quadratic Number of Edges

The idea is completely analogous to the previous section. We employ the following characterization.

A *star with a triangle* is formed by connecting two leaves of a star with an edge. A *bi-star* is formed by connecting the centers of two stars with an edge.

▶ **Lemma 12** (Červený and Suchý [3, Lemma 4]). *A connected 5-path free graph is either a graph on at most 4 vertices, a star with a triangle, or a bi-star.*

We conclude this section with the final statement about our kernel.

▶ **Theorem 13** ($\star$). *5-Path Vertex Cover admits a kernel with* $245k^2 + 245k$ *edges, where* k *is the size of the solution.*

7 d-PVC Kernel with $\mathcal{O}(k^4 d^{2d+9})$ Edges

In this section, we give a kernelization algorithm for d-PVC with $d \geq 6$.

An intuition behind the approach. The kernelization algorithm marks some vertices and edges, which it wants to keep, and throws away the rest. Essentially, the kernelization creates a subgraph $\widehat{G}$ of the input graph G. For correctness of the algorithm, we want to show that if there is a d-path P in G which misses some set of vertices S (a prospective solution), then we will also find some d-path P' in $\widehat{G}$, which also misses the set S.

We begin by finding a maximal packing $\mathcal{M}$ in G and we keep in $\widehat{G}$ all vertices M of the packing and all edges between them. Now, on one hand, if the path P would be completely contained in M, then trivially the path appears also in $\widehat{G}$. On the other hand, the path P cannot be completely outside of M. Thus, the path P crosses between M and outside of M at least once. This corresponds to vertices of M being connected by a path of prescribed length outside of M. We later formalize this as a "request".

To get more structure, we leverage the behavior of DFS trees of the connected components outside of M. With the DFS trees we identify vertices, which are "crucial" for the requests, and we further split the requests into "sub-requests" according to the "crucial" vertices.

The algorithm is inspired by Dell and Marx [9]. However, while the considered problems have similarities, many ideas are not translatable. In particular, they could afford to consider all "sub-requests" and keep $\Omega(k)$ vertices for each without affecting their bound (cf. [9, p. 23]). We had to be more careful in which "sub-requests" we consider and we need to employ Lemma 15 (below) to only keep $d^{\mathcal{O}(d)}$ vertices and edges for each such "sub-request" to achieve our precise bound. Also, to achieve the edge bound, we need to keep track of the purpose for which the individual vertices were marked, which makes it hard to split the algorithm into small self-contained steps.

Formal definitions. More formally, assume that we are given an instance $(G = (V, E), k)$ of d-PVC. We start by running the algorithm of Proposition 1 on the instance. If it answers directly, then we are done. Otherwise it returns a maximal packing $\mathcal{M}$ in G. Let M be the vertices of the packing $\mathcal{M}$. Recall that $|M| \leq dk$.

Let $G' = G \setminus M$, i.e., G' is the graph outside the packing $\mathcal{M}$. Label the connected components of G' as $G'_1, G'_2, \ldots, G'_t$. For each component G'_i pick an arbitrary vertex $r_i \in G'_i$ and compute a *depth-first search* tree T_i of G'_i rooted at r_i. Note that $V(T_i) = V(G'_i)$. Let $\mathcal{F}$ denote the forest consisting of all the trees $T_i, i \in [t]$. Note that $V(\mathcal{F}) = V(G')$.

For a rooted forest F and its vertex $v \in V(F)$, $\mathit{sub}(v)$ denotes the set of vertices of a maximal subtree of F rooted at v and $\mathit{anc}(v)$ the set of ancestors of v in F, i.e., $\mathit{anc}(v) = \{u \mid u \in V(F), v \in \mathit{sub}(u)\}$. Note that $v \in \mathit{anc}(v)$.

We provide the following observations regarding the DFS trees T_i and the forest $\mathcal{F}$.

▶ **Observation 14** ($\star$).

(a) *Each $y \in V(\mathcal{F})$ has at most $d-1$ ancestors.*

(b) *For two vertices $u, v \in V(\mathcal{F})$ which are not in ancestor-descendant relation, we have $sub(u) \cap sub(v) = \emptyset$ and $\{u, v\} \notin E(G)$.*

(c) *If C is a connected subgraph of G', then there is a vertex $w \in V(C)$ such that $V(C) \subseteq sub(w)$.*

A triple (f, l, X) is *an X-request*, if $l \in \{1, \dots, d-1\}$, $X \subseteq V(G)$, and either $f = \{u, v\} \subseteq X$, $u \neq v$, or $f = \{x\} \subseteq X$. For an X-request (f, l, X) and $H \subseteq V(G)$, we use $\mathcal{P}^H_{f,l}$ to denote the set of paths P of length l in $G[H \cup f]$ such that each $x \in f$ is an endpoint of P. In particular, if $f = \{u, v\}$, $u \neq v$, then u and v are the endpoints of P and if $f = \{x\}$, then one endpoint of P is x and the other can be any vertex of H. A set $H \subseteq V(G)$ is said to *satisfy an X-request* (f, l, X) if $\mathcal{P}^H_{f,l} \neq \emptyset$.

An M-request (f, l, M) will be simply denoted as $\varrho(f, l)$ and called *a request*.

In the next paragraphs we cover the notion of the "crucial" vertices mentioned in the earlier intuition. Roughly speaking, a vertex of F is "crucial" if the set of its descendants (vertices of its subtree) satisfies some request.

For each request $\varrho(f, l)$ we define the set $Y_{f,l} \subseteq V(\mathcal{F})$ as the set of all vertices $v \in V(\mathcal{F})$ such that $sub(v)$ satisfies the request $\varrho(f, l)$. Note that if $v \in Y_{f,l}$, then also $w \in Y_{f,l}$ for every $w \in anc(v)$, since $sub(w) \supseteq sub(v)$. Thus, if $Y_{f,l} \cap V(G'_i) \neq \emptyset$ for some i, then in particular $r_i \in Y_{f,l}$ and $Y_{f,l}$ induces a connected subtree of T_i. The request $\varrho(f, l)$ is *resolved* if the subforest $\mathcal{F}[Y_{f,l}]$ has at least $k + d + 1$ leaves.

Recall, that in the intuition we examined some d-path P in G. The resolved request basically ensures that there are at least $k + d + 1$ disjoint paths in G which satisfy said request. The idea is that the prospective solution may compromise at most k of these paths and the other parts of P may compromise at most d of these paths. As we will keep exactly $k + d + 1$ of these disjoint paths in $\widehat{G}$, we can be sure, that at least one of them will always be usable to reroute some part of P which will help us to find the desired path P' in $\widehat{G}$.

Now, we focus on the unresolved requests. Let $\mathcal{R}^*$ be the set of all requests $\varrho(f, l)$ which are *not* resolved and let $\mathcal{Y} = \bigcup_{\varrho(f,l) \in \mathcal{R}^*} Y_{f,l}$.

We are now getting to the notion of sub-requests. These have either one endpoint in M and the other in $\mathcal{Y}$, or both endpoints in $\mathcal{Y}$, or only one prescribed endpoint, which is in $\mathcal{Y}$. Note that if the two endpoint are in $\mathcal{Y}$, then, by Observation 14, either one of the endpoints is an ancestor of the other, or there is no path connecting them outside $(M \cup \mathcal{Y})$.

An $(M \cup \mathcal{Y})$-request $(g, j, M \cup \mathcal{Y})$ will be simply denoted as $\sigma(g, j)$ and called *a sub-request* if there exists $y \in \mathcal{Y}$ such that $y \in g$ and $g \subseteq M \cup anc(y)$. In particular, either $g = \{y\}$ or one of the vertices in g is y and the other vertex is in $M \cup anc(y)$.

Even though we will not formally define a *resolved sub-request*, later we will actually show that the sub-request is "resolved" if there are at least $2d$ paths satisfying it.

Description of the algorithm. We are now ready to describe the kernelization algorithm. As we mentioned earlier, the algorithm first marks some vertices and edges, which it wants to keep, and it deletes the rest of the graph. Therefore, the core of the algorithm is the marking procedure. In our case, the main procedure is called **Mark** which in turn uses a procedure called **Mark 2**. These procedures are described in Algorithm 1 and Algorithm 2, respectively.

Let us now give an insight into how the procedures **Mark** and **Mark 2** were constructed. We will start with **Mark**.

Algorithm 1 Marking procedure **Mark**.

```
1  Let $M$, $\mathcal{F}$, and $\mathcal{Y}$ be as in the text.
2  Mark all the vertices and edges in $G[M \cup \mathcal{Y}]$.
3  foreach resolved request $\varrho(f, l)$ do
4      Pick arbitrary $k + d + 1$ leaves $h_1, h_2, \ldots, h_{k+d+1}$ of $\mathcal{F}[Y_{f,l}]$.
5      foreach leaf $h_i$ do
6          Pick an arbitrary path $P$ from $\mathcal{P}_{f,l}^{sub(h_i)}$.
7          Mark the vertices and edges of $P$.
8  foreach $y \in \mathcal{Y}$ do
9      foreach sub-request $\sigma(g, j)$ such that $g \subseteq M \cup anc(y)$ and $g \not\subseteq M$ do
10         Let $C_1, C_2, \ldots, C_{q'}$ be the vertex sets of the connected components of
           $G \setminus (M \cup \mathcal{Y})$ such that for all $i \in [q']$, $N(C_i) \cap \mathcal{Y} \subseteq anc(y)$ and $C_i$ satisfies
           $\sigma(g, j)$.
11         if $q' \geq 2d$ then
12             foreach $i \in [2d]$ do
13                 Pick an arbitrary path $P \in \mathcal{P}_{g,j}^{C_i}$.
14                 Mark the vertices and edges of $P$.
15         else
16             foreach $i \in [q']$ do
17                 Run the marking procedure Mark 2 on $\Big(\sigma(g, j), C_i, \emptyset\Big)$.
```

Algorithm 2 Marking procedure **Mark 2**$\Big(\sigma(g, j), C_i, W\Big)$.

```
1  if $|W| \leq 2d$ and $\mathcal{P}_{g,j}^{C_i \setminus W} \neq \emptyset$ then
2      Let $P \in \mathcal{P}_{g,j}^{C_i \setminus W}$.
3      Mark all the edges and vertices of $P$.
4      foreach $v \in V(P) \setminus g$ do
5          $W' = W \cup \{v\}$
6          Call Mark 2 on $\Big(\sigma(g, j), C_i, W'\Big)$
```

The lines 3–7 deal with the resolved requests. Essentially, by preserving $k + d + 1$ corresponding paths for the request $\varrho(f, l)$, we retain all the necessary structure such that we do not create any new solutions in the reduced instance.

In the two following `for`-cycles, we first pick a vertex y of $\mathcal{Y}$. This fixes the set of ancestors $anc(y)$, i.e., it fixes the set of vertices on the path from y to the root of its tree in $\mathcal{F}$. And, for this particular y, we then pick a sub-request $\sigma(g, j)$ which lives on this fixed set $anc(y)$ and M. This allows us to look only at some components of $G \setminus (M \cup \mathcal{Y})$ and actually makes it possible for us to bound their number. The bounding happens on lines 11–14 and we can also say that the sub-request $\sigma(g, j)$ is resolved when the number of components is at least $2d$. The bound $2d$ follows from Lemma 15, which will be stated later. For a resolved sub-request we proceed similarly to resolved request. Namely, we preserve one corresponding path in each of some $2d$ of the components.

If the number of components is not large, the lines 15–17 run the second marking procedure **Mark 2** on each of these components and the aim is to bound their size.

Now, recall again, that in the intuition we examined some d-path P in G and some prospective solution S. The purpose of the marking procedure **Mark 2** is to brute-force all the possible ways of how the path P and solution S may compromise the paths which satisfy the sub-request $\sigma(g,j)$ and which are contained in the component C_i. The procedure works recursively, starts with the empty set of "compromising" vertices W and it always picks a path which was not yet compromised, marks it (so that it remains in $\widehat{G}$), and tries to compromise its vertices one by one. By doing it like this, we ensure, that all the important parts of C_i remain in $\widehat{G}$ no matter what parts of C_i were compromised.

And the main trick is that we can stop the recursion of **Mark 2** once the number of "compromising" vertices reaches $2d$. This number $2d$ again follows from Lemma 15.

Now, the kernelization can be formally summarized as follows. Run the marking procedure **Mark** on the instance (G,k). The marking results in two subsets $\widehat{V} \subseteq V(G)$ and $\widehat{E} \subseteq E(G)$ corresponding to marked vertices and edges by **Mark**. Reduce the instance (G,k) to the instance $(\widehat{G},k)$ where $\widehat{G} = (\widehat{V},\widehat{E})$.

With that we conclude the intuition and we continue with the formal proof of correctness. First, we state the crucial Lemma 15, then the main body of the proof follows in Lemma 16, and we finish with the proof of the size of the kernel in Lemma 17.

Lemma 15 roughly states that any reasonable solution only contains at most d vertices among each set of components considered on line 10 of Algorithm 1.

▶ **Lemma 15.** *Let (G,k) be an instance of d-PVC. Let $\widehat{G} = (\widehat{V},\widehat{E})$ be a subgraph of G such that $\widehat{V}$ and $\widehat{E}$ are the result of running the marking procedure **Mark** on (G,k). Let S' be a solution for the instance $(\widehat{G},k)$ of d-PVC. Let $y \in \mathcal{Y}$ and let $C_1, C_2, \dots, C_c$ be the vertex sets of the connected components of $\widehat{G} \setminus (M \cup \mathcal{Y})$ such that $N(C_i) \cap \mathcal{Y} \subseteq anc(y)$ for $i \in [c]$. Then $\widehat{S} = (S' \setminus \bigcup_{i\in[c]} C_i) \cup anc(y)$ is a solution for $\widehat{G}$.*

Proof. If $\widehat{S}$ is a solution for $(\widehat{G},k)$, we are done. Suppose to the contrary that it is not. Then there is a d-path P in $\widehat{G} \setminus \widehat{S}$. Assume that P is selected such that it contains the least number of vertices which are in S' i.e., $|V(P) \cap S'|$ is minimized among all d-paths in $\widehat{G} \setminus \widehat{S}$. As $S' \setminus \widehat{S} \subseteq \bigcup_{i\in[c]} C_i$, path P must contain at least one vertex from at least one set $C_i \cap S'$, because otherwise P would also be in $\widehat{G} \setminus S'$, which is a contradiction with S' being a solution for $(\widehat{G},k)$. Further, since $N(C_i) \subseteq (M \cup \mathcal{Y})$, $N(C_i) \cap \mathcal{Y} \subseteq anc(y)$, and $anc(y) \subseteq \widehat{S}$ by assumption, we have $N(C_i) \setminus \widehat{S} \subseteq M$. Therefore $P \cap M \neq \emptyset$, as otherwise the path P would be contained in C_i, which is a contradiction with $\mathcal{M}$ being a maximal packing of d-paths.

We split the path P into segments according to the vertices of M, i.e., a *segment* of P is a sub-path $(v_1, v_2, \dots, v_s)$ of P such that $v_2, v_3, \dots, v_{s-1} \in V(G) \setminus M$ and either $\{v_1, v_s\} \subseteq M$ (an inner segment), or one of v_1, v_s is in M, while the other is an endpoint of P (an outer segment). The argument is the same in both cases.

Let $P' = (v_1, v_2, \dots, v_s)$ be the segment of P which uses some vertex from $C_i \cap S'$. Observe, that the segment P' corresponds to request $\varrho(f,l) = \varrho(V(P') \cap M, s-1)$ as $2 \leq s \leq d$ and $|V(P') \cap M| \in \{1,2\}$. In particular, $V(P') \setminus M$ satisfies $\varrho(f,l)$.

We also know that $V(P') \cap \mathcal{Y} = \emptyset$, because $N(C_i) \setminus \widehat{S} \subseteq M$. With that we argue that the request $\varrho(f,l)$ must be resolved. Indeed, suppose it is not. By Observation 14(c) there is a vertex v in $V(P') \cap C_i$ such that that $V(P') \cap C_i \subseteq sub(v)$. But that implies that $sub(v)$ satisfies the request $\varrho(f,l)$ and, therefore, vertex v should have been included in $Y_{f,l}$ and, consequently, v should have been included in $\mathcal{Y}$, which is a contradiction with $V(P') \cap \mathcal{Y} = \emptyset$.

Now, as the request $\varrho(f,l)$ is resolved, the marking procedure **Mark** picked $k+d+1$ leaves $h_1, h_2, \dots, h_{k+d+1}$ from $\mathcal{F}[Y_{f,l}]$ and for each such leaf h_i it marked the vertices and edges of some path $P_i \in \mathcal{P}_{f,l}^{sub(h_i)}$. Therefore, these paths $P_1, P_2, \dots, P_{k+d+1}$ remained in $\widehat{G}$. Further, at least one of these paths is untouched by the vertices of S' and the vertices of P as $|S'| \leq k$ and $|V(P)| \leq d$, respectively. Let this one untouched path be P_i. Observe, that we can swap the segment P' with the path P_i in P to obtain a d-path P^*. But then the path P^* contains strictly fewer vertices which are in S' than P, which is a contradiction with the choice of P. ◀

Now we can prove the correctness of the algorithm.

▶ **Lemma 16** (⋆). *Let (G,k) be an instance of d-PVC. Let $\widehat{G} = (\widehat{V}, \widehat{E})$ be a subgraph of G such that $\widehat{V}$ and $\widehat{E}$ are obtained by running the marking procedure* ***Mark*** *on (G,k). Then, (G,k) is a YES instance if and only if $(\widehat{G},k)$ is a YES instance.*

Proof sketch. For the "if" direction we pick a solution S' to $\widehat{G}$ such that any application of Lemma 15 would increase its size. If S' was not a solution to G, then as in Lemma 15, we pick a special d-path P witnessing that, but this time with the least number of unmarked edges in G. Then we again split P into segments according to M and, in case the corresponding request was not resolved, further into sub-segments according to $\mathcal{Y}$. We always pick a (sub-)segment with at least one unmarked edge and we show that we can swap the (sub-)segment with some other suitable fully marked sub-path to obtain a contradiction with the choice of P. ◀

The following lemma shows the bound on the size of the kernel.

▶ **Lemma 17** (⋆). *Let (G,k) be an instance of d-PVC. Let $\widehat{G} = (\widehat{V}, \widehat{E})$ be a subgraph of G such that $\widehat{V}$ and $\widehat{E}$ are obtained by running the marking procedure* ***Mark*** *on (G,k). Then, $|\widehat{V}| = \mathcal{O}(k^4 d^{2d+9})$ and $|\widehat{E}| = \mathcal{O}(k^4 d^{2d+9})$.*

We summarize the result in the following theorem.

▶ **Theorem 18.** *d-Path Vertex Cover admits a kernel with $\mathcal{O}(k^4 d^{2d+9})$ vertices and edges, where k is the size of the solution.*

Proof. As the marking procedures **Mark** and **Mark 2** can by implemented in polynomial time, the theorem directly follows from Lemmas 16 and 17. ◀

8 Conclusion

We presented kernels with $\mathcal{O}(k^2)$ edges for 4-PVC and 5-PVC and with $\mathcal{O}(k^4 d^{2d+9})$ edges for d-PVC for any $d \geq 6$. An obvious open question is whether there is a kernel with $\mathcal{O}(k^2)$ edges for every $d \geq 6$.

Furthermore, the size of our kernel depends on d by a factor of $d^{\mathcal{O}(d)}$. We believe that this could be improved to $2^{\mathcal{O}(d)}$ with the use of representative sets. However, improving this to a factor polynomial in d would imply coNP $\subseteq$ NP/poly. As observed by Dell and Marx [9], running such a kernel with $k = 0$ would give a polynomial kernel for the d-Path problem, which would have the above mentioned implications.

Next, for 2-PVC and 3-PVC, there are kernels with linear number of vertices [19, 32]. Hence, another open question is whether such a kernel can be obtained also for say 4-PVC. Further interesting open questions can be found in the recent survey of Tu [29].

References

1 Boštjan Brešar, František Kardoš, Ján Katrenič, and Gabriel Semanišin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189–1195, 2011. doi:10.1016/j.dam.2011.04.008.

2 Eglantine Camby, Jean Cardinal, Mathieu Chapelle, Samuel Fiorini, and Gwenaël Joret. A primal-dual 3-approximation algorithm for hitting 4-vertex paths. In *9th International colloquium on graph theory and combinatorics*, 2014.

3 Radovan Červený and Ondřej 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. doi:10.4230/LIPIcs.MFCS.2019.32.

4 Radovan Červený and Ondřej Suchý. Faster FPT algorithm for 5-path vertex cover. *CoRR*, abs/1906.09213, 2019. arXiv:1906.09213.

5 Radovan Červený and Ondřej Suchý. Generating faster algorithms for d-path vertex cover. *CoRR*, abs/2111.05896, 2021. arXiv:2111.05896.

6 Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Yi-Zhi Liu, Peter Rossmanith, and Somnath Sikdar. Moderately exponential time algorithms for the maximum bounded-degree-1 set problem. *Discret. Appl. Math.*, 251:114–125, 2018. doi:10.1016/j.dam.2018.05.032.

7 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

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 Holger Dell and Dániel Marx. Kernelization of packing problems. *CoRR*, abs/1812.03155, 2018. arXiv:1812.03155.

10 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 251–260. ACM, 2010. doi:10.1145/1806689.1806725.

11 Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2016.

12 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

13 Stefan Fafianie and Stefan Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-48054-0_25.

14 Henning Fernau. Parameterized algorithmics for *d*-hitting set. *Int. J. Comput. Math.*, 87(14):3157–3174, 2010. doi:10.1080/00207160903176868.

15 Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7-9):1045–1053, 2010. doi:10.1016/j.tcs.2009.11.012.

16 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discret. Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.

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), September 2016. doi:10.1145/2886094.

18 Stefan Funke, André Nusser, and Sabine Storandt. On k-path covers and their applications. *VLDB J.*, 25(1):103–123, 2016. doi:10.1007/s00778-015-0392-3.

19 Michael Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Inf. Process. Lett.*, 111(23-24):1089–1091, 2011. doi:10.1016/j.ipl.2011.09.003.

20 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.

21 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

22 Marián Novotný. Design and analysis of a generalized canvas protocol. In *Proc. 4th IFIP WG 11.2 International Workshop on Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, WISTP 2010*, pages 106–121, 2010. doi:10.1007/978-3-642-12368-9_8.

23 Elena Prieto-Rodríguez. *Systematic kernelization in FPT algorithm design*. PhD thesis, University of Newcastle, 2005. URL: http://hdl.handle.net/1959.13/1418337.

24 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

25 Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theor. Comput. Sci.*, 790:96–104, 2019. doi:10.1016/j.tcs.2019.04.024.

26 Dekel Tsur. l-path vertex cover is easier than l-hitting set for small l. *CoRR*, abs/1906.10523, 2019. arXiv:1906.10523.

27 Dekel Tsur. Parameterized algorithm for 3-path vertex cover. *Theor. Comput. Sci.*, 783:1–8, 2019. doi:10.1016/j.tcs.2019.03.013.

28 Dekel Tsur. An $O^*(2.619^k)$ algorithm for 4-path vertex cover. *Discret. Appl. Math.*, 291:1–14, 2021. doi:10.1016/j.dam.2020.11.019.

29 Jianhua Tu. A survey on the k-path vertex cover problem. *CoRR*, abs/2201.03397, 2022. arXiv:2201.03397.

30 Jianhua Tu and Wenli Zhou. A primal-dual approximation algorithm for the vertex cover P_3 problem. *Theor. Comput. Sci.*, 412(50):7044–7048, 2011. doi:10.1016/j.tcs.2011.09.013.

31 Mingyu Xiao and Shaowei Kou. Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theor. Comput. Sci.*, 657:86–97, 2017. doi:10.1016/j.tcs.2016.04.043.

32 Mingyu Xiao and Shaowei Kou. Kernelization and parameterized algorithms for 3-path vertex cover. In T. V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Theory and Applications of Models of Computation – 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, volume 10185 of *Lecture Notes in Computer Science*, pages 654–668, 2017. doi:10.1007/978-3-319-55911-7_47.

33 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

34 Meirav Zehavi. Mixing color coding-related techniques. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 1037–1049. Springer, 2015. doi:10.1007/978-3-662-48350-3_86.

On Synthesizing Computable Skolem Functions for First Order Logic

Supratik Chakraborty
Department of Computer Science and Engineering, IIT Bombay, India

S. Akshay
Department of Computer Science and Engineering, IIT Bombay, India

Abstract

Skolem functions play a central role in the study of first order logic, both from theoretical and practical perspectives. While every Skolemized formula in first-order logic makes use of Skolem constants and/or functions, not all such Skolem constants and/or functions admit effectively computable interpretations. Indeed, the question of whether there exists an effectively computable interpretation of a Skolem function, and if so, how to automatically synthesize it, is fundamental to their use in several applications, such as planning, strategy synthesis, program synthesis etc.

In this paper, we investigate the computability of Skolem functions and their automated synthesis in the full generality of first order logic. We first show a strong negative result, that even under mild assumptions on the vocabulary, it is impossible to obtain computable interpretations of Skolem functions. We then show a positive result, providing a precise characterization of first-order theories that admit effective interpretations of Skolem functions, and also present algorithms to automatically synthesize such interpretations. We discuss applications of our characterization as well as complexity bounds for Skolem functions (interpreted as Turing machines).

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Skolem functions, Automated, Synthesis, First order logic, Computability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.30

Related Version *Full Version*: https://arxiv.org/abs/2102.07463

1 Introduction

The history of Skolem functions can be traced back to 1920, when the Norwegian mathematician, Thoralf Albert Skolem, gave a simplified proof of a landmark result in logic, now known as the *Löwenheim-Skolem* theorem. Skolem's proof made use of a key observation: *For every first order logic formula $\exists y\, \varphi(x, y)$, the choice of y that makes $\varphi(x, y)$ true (if at all) depends on x in general. This dependence can be thought of as implicitly defining a function that gives the "correct" value of y for every value of x. If F_y denotes a fresh unary function symbol, the second order sentence $\exists F_y\, \forall x\, \big(\exists y\, \varphi(x, y) \Rightarrow \varphi(x, F_y(x))\big)$ formalizes this idea.* Since the implication trivially holds in the other direction too, the second order sentence $\exists F_y\, \forall x\, \big(\exists y\, \varphi(x, y) \Leftrightarrow \varphi(x, F_y(x))\big)$ is valid.

Let $\xi_1 \equiv \exists y\, \varphi(x, y)$ and $\xi_2 \equiv \varphi(x, F_y(x))$. The fresh function symbol F_y introduced in transforming ξ_1 to ξ_2 is called a *Skolem function*. Skolem functions play an extremely important role in logic – both from theoretical and applied perspectives. While it suffices in some contexts to simply know that a Skolem function F_y exists, in other contexts, we require an effective procedure to compute $F_y(x)$ for every value of x. This motivates us to ask if Skolem functions are always computable, and whenever they are, can we algorithmically generate a halting Turing machine that computes the function? Note that we are concerned with computability at two levels here: (i) computability of the Skolem function itself, and (ii) computability of a halting Turing machine that computes the Skolem function. For clarity of

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 30; pp. 30:1–30:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

exposition, we call the Turing machine referred to in (ii) above a *computable interpretation of Skolem function*, and the problem of generating it algorithmically the *synthesis problem for Skolem functions.*

The synthesis problem for Skolem functions has been studied in detail in the propositional setting, specifically for quantified Boolean formulas (QBF) with a $\forall^*\exists^*$ quantifier prefix [17, 16, 12, 18, 13, 22, 27, 5, 23, 3, 1, 21, 4, 14, 24]. Surprisingly, a similar in-depth investigation in the context of general first order logic appears lacking in the literature, despite several potential applications, viz. automatic program synthesis and repair [26, 19, 28]. Some notable works in the context of specific theories include those of Kuncak et al [19] (for linear rational arithmetic), Spielman et al [25] (unbounded bit-vector theory), Preiner et al [20] (bit-vector theory) etc. in which terms that serve as interpretations of Skolem functions in specific theories are synthesized. In [17], Jiang presented a partial approach for quantifier elimination in general first-order theories by relying on the availability of functions that can be conditionally expressed by a finite set of terms. Unfortunately, such finite conditional decomposition may not always be possible (as acknowledged in [17]), even when a computable interpretation of the Skolem function exists. The problem of quantifier-free constraint solving, i.e. finding assignments of free variables that render a quantifier-free formula `true`, has been investigated in depth for several theories, viz. propositional logic, theory of arrays, linear rational arithmetic, real algebraic numbers, Presburger arithmetic, regular languages of finite strings etc. If the theory also admits effective quantifier elimination, this yields an algorithm for synthesizing computable interpretations of Skolem functions. However, not all first order theories admit effective quantifier elimination, e.g. Presburger arithmetic (without divisibility predicates) or the theory of evaluated trees [10] does not. We show that for some such theories, computable interpretations of Skolem functions can be synthesized algorithmically.

Our main contributions are to ask and answer the following questions:

- Does there always exist computable interpretations of Skolem functions for a first order formula interpreted over a structure? We answer this question strongly in the negative by showing that uncomputable interpretations cannot be avoided even with one binary predicate and one existential quantifier in the formula.
- We next ask if it is possible to algorithmically decide whether computable interpretations exist for all Skolem functions, given a formula and a structure over which it is interpreted. We answer this question in the negative.
- Next, we ask if it is possible to characterize the class of structures such that effectively computable interpretations of Skolem functions can be algorithmically synthesized for all formulas interpreted over a structure in the class. We answer this by showing that decidability of the elementary diagram of a structure serves as the required necessary and sufficient condition. Using this result, we show that several important first-order theories admit synthesis of effectively computable Skolem functions, while others do not.
- For structures satisfying the condition in the above characterization, we present lower and upper complexity bounds for effectively computable interpretations of Skolem functions.
- Finally, we distinguish between synthesizing Skolem functions as halting Turing machines vs terms in the underlying logic and show that the latter is a strictly weaker notion.

Our results reveal a highly a nuanced picture of the computability landscape for synthesizing interpretations of Skolem functions in first-order logic. We hope that this work will be a starting point towards further research into the design of practical algorithms (whenever possible) to synthesize Skolem functions for various first order theories. Proofs that are missing due to lack of space can be found in the full version at [2].

2 Preliminaries

Since every Turing machine with tape alphabet $\{0,1\}$ can be encoded as a natural number (we use $\mathbb{N}$ for naturals), and since every finite string over $\{0,1\}^*$ can be encoded as a natural number, we often speak of Turing machine i, denoted TM_i, running on input string j, where $i, j \in \mathbb{N}$.

We use x, y, z, etc., possibly with subscripts, to denote first order variables, $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, etc., possibly with subscripts, to denote sequences of first order variables. We use φ, ξ, α, possibly with subscripts, to denote formulas. For a sequence $\mathbf{X}_i$, $|\mathbf{X}_i|$ denotes the count of variables in $\mathbf{X}_i$, and $x_{i,1}, \ldots x_{i,|\mathbf{X}_i|}$ denotes the variables. A *vocabulary* $\mathcal{V}$, is a set of function and/or predicate symbols, along with their respective arities. Constants are function symbols with arity 0. We assume that $\mathcal{V}$ *has finitely many predicate and function symbols, except possibly for countably infinitely many constant symbols.* We also assume that *a special binary predicate "$=$" (equality) is present in every vocabulary.*

We consider first order logic formulas over vocabulary $\mathcal{V}$, also called $\mathcal{V}$*-formulas.* The notion of *bound* and *free* variables is standard, $\mathcal{V}$-formulas without free variables are $\mathcal{V}$-*sentences.* A $\mathcal{V}$*-term* is either a variable or $f(t_1, \ldots t_k)$, where f is a k-ary function symbol in $\mathcal{V}$ and $t_1, \ldots t_k$ are $\mathcal{V}$-terms. When $\mathcal{V}$ is implicit from the context, we omit it. A *ground term* (resp. *ground formula*) is a term (resp. formula) without any variables. For x, a free variable in ξ, t a term in which all variables (if any) are free in ξ, $\xi[x \mapsto t]$ denotes the formula obtained by *substituting t for x in ξ*, i.e., replacing every free occurrence of x in ξ with t.

A $\mathcal{V}$*-structure* $\mathfrak{M}$ consists of a *universe* $U^{\mathfrak{M}}$ of elements and an interpretation of every predicate and function symbol in $\mathcal{V}$ over $U^{\mathfrak{M}}$. The interpretation of the special predicate "$=$" is always the identity relation, and we write $t_1 = t_2$ instead of $=(t_1, t_2)$ for notational convenience. We denote the interpretation of a predicate symbol P (resp. function symbol f) in $\mathfrak{M}$ as $P^{\mathfrak{M}}$ (resp. $f^{\mathfrak{M}}$). In general, an interpretation of a predicate or function symbol may be well-defined but not computable. We say a $\mathcal{V}$-structure $\mathfrak{M}$ is *computable* if $U^{\mathfrak{M}}$ is countable and if $P^{\mathfrak{M}}$ (resp. $f^{\mathfrak{M}}$) is computable for all predicate symbol P (resp. function symbol f) in $\mathcal{V}$. In other words, there exists a halting Turing machine for computing the interpretations $P^{\mathfrak{M}}$ (resp. $f^{\mathfrak{M}}$). *Throughout this paper, we assume that all $\mathcal{V}$-structures are computable.* This is motivated by practical applications of Skolem functions; additionally, non-computable $\mathcal{V}$-structures may make it difficult (even impossible) to obtain computable interpretations of Skolem functions in most cases. A computable $\mathcal{V}$ structure can be finitely represented, e.g. by using a single bit to encode whether the universe is finite or countably infinite, and by giving a natural number encoding of each Turing machine that computes an interpretation of a predicate or function symbol. If there are countably infinite constant symbols, we assume that interpretations of all of them can be collectively encoded by a single Turing machine that computes a mapping from $\mathbb{N}$ (index of constant symbol) to $\mathbb{N}$ (index of element in universe). If a $\mathcal{V}$-formula $\xi(\mathbf{Z})$ evaluates to true when interpreted over $\mathfrak{M}$ and with $\mathbf{Z}$ set to $\sigma \in \left(U^{\mathfrak{M}}\right)^{|\mathbf{Z}|}$, we say that $\mathfrak{M}$ is a *model* of $\xi(\sigma)$ and denote it by $\mathfrak{M} \models \xi(\sigma)$. An *expansion* of a vocabulary $\mathcal{V}$ is a vocabulary $\mathcal{V}'$ such that $\mathcal{V} \subseteq \mathcal{V}'$. Given a $\mathcal{V}$-structure $\mathfrak{M}$ and a $\mathcal{V}'$-structure $\mathfrak{M}'$, where $\mathcal{V}'$ is an expansion of $\mathcal{V}$, $\mathfrak{M}'$ is an *expansion* of $\mathfrak{M}$ if (i) $U^{\mathfrak{M}'} = U^{\mathfrak{M}}$, and (ii) all predicate/function symbols in $\mathcal{V}$ are interpreted identically in $\mathfrak{M}$ and $\mathfrak{M}'$.

For a quantifier $Q \in \{\exists, \forall\}$ and sequence of variables $\mathbf{X}_i = (x_{i,1}, \ldots x_{i,|\mathbf{X}_i|})$, we use $Q\mathbf{X}_i$ to denote the block of quantifiers $Qx_{i,1} \ldots Qx_{i,|\mathbf{X}_i|}$. Every first order logic formula can be effectively transformed to a semantically equivalent *prenex normal form*, in which all quantifiers appear to the left of the quantifier-free part of the formula. Henceforth, we assume all first order formulas are in prenex normal form, unless stated otherwise. Let $\xi(\mathbf{Z}) \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \cdots \forall \mathbf{X}_q \exists \mathbf{Y}_q\, \varphi(\mathbf{Z}, \mathbf{X}_1, \mathbf{Y}_1, \ldots \mathbf{X}_q, \mathbf{Y}_q)$ be such a formula, where $\mathbf{Z}$ is a sequence of free variables, and φ is quantifier-free. We say that $\forall \mathbf{X}_1 \exists \mathbf{Y}_1 \cdots \forall \mathbf{X}_q \exists \mathbf{Y}_q$ is the *quantifier prefix* of the formula, and it has q $\forall^* \exists^*$ blocks. The quantifier-free part, i.e. φ, is called the *matrix* of the formula. Note that in case the leading (leftmost) quantifier in ξ is existential, $\mathbf{X}_1$ may be considered to be an empty sequence, and similarly, if the trailing (rightmost) quantifier in ξ is universal. Every variable $y_{i,j}$ that is existentially quantified in the quantifier prefix is called an an *existential variable* in ξ. The notion of *universal variables* is analogously defined. The quantifier prefix imposes a total order on the quantified variables in ξ. We say that a variable u is *to the left* (resp. *right*) of variable v in the quantifier prefix iff Qu appears to the left (resp. right) of $Q'v$ in the quantifier prefix, where $Q, Q' \in \{\exists, \forall\}$.

Skolemization. Given a formula ξ in prenex normal form, *Skolemization* refers to the process of transforming ξ to a new formula $\xi^\star$ via the following steps: (i) for every existential variable $y_{i,j}$, substitute $F_{y_{i,j}}(\mathbf{Z}, \mathbf{X}_1, \ldots \mathbf{X}_i)$ for $y_{i,j}$ in φ, where $F_{y_{i,j}}$ is a new function symbol of arity $|\mathbf{Z}| + \sum_{j=1}^{i} |\mathbf{X}_j|$, and (ii) remove all existential quantifiers from the quantifier prefix of ξ. The functions $F_{y_{i,j}}$ introduced above are called *Skolem functions.* In case ξ has no free variables and the leading quantifier is existential, the Skolem functions for variables in the leftmost existential quantifier block have no arguments (i.e. they are nullary functions). Such functions are also called *Skolem constants.* The sentence $\xi^\star$ is said to be in *Skolem normal form* if the matrix of $\xi^\star$ is in conjunctive normal form. The key guarantee of Skolemization is as follows: for every existential variable $y_{i,j}$, let $\xi^\star_{y_{i,j}}$ denote the formula obtained by Skolemizing all existential variables to the left of $y_{i,j}$ in the quantifier prefix. Formally, $\xi^\star_{y_{i,j}}$ is obtained by (i) substituting the Skolem function $F_{y_{k,l}}$ for every existential variable $y_{k,l}$ to the left of $y_{i,j}$ in the quantifier prefix, and (ii) removing all quantifiers to the left of and including $\exists y_{i,j}$ from the quantifier prefix. Note that $\xi^\star_{y_{i,j}}$ has free variables in $\mathbf{Z}, \mathbf{X}_1, \ldots \mathbf{X}_i, y_{i,j}$. Skolemization guarantees that for every $\mathcal{V}$-structure $\mathfrak{M}$ over which ξ is interpreted, there always exists an expansion $\mathfrak{M}^\star$ of $\mathfrak{M}$ that provides an interpretation of $F_{y_{i,j}}$ for all existential variables $y_{i,j}$ such that the following holds for every $i \in \{1, \ldots q\}$ and $j \in \{1, \ldots |\mathbf{Y}_i|\}$:

$$\forall \mathbf{Z} \forall \mathbf{X}_1 \ldots \forall \mathbf{X}_i \left(\exists y_{i,j}\, \xi^\star_{y_{i,j}} \;\Leftrightarrow\; \xi^\star_{y_{i,j}}[y_{i,j} \mapsto F_{y_{i,j}}(\mathbf{Z}, \mathbf{X}_1, \ldots \mathbf{X}_i)]\right) \tag{1}$$

▶ **Example 1.** Consider $\xi(z) \equiv \exists y \forall x \exists u \forall v \exists w\, \varphi(z, x, y, u, v, w)$. Skolemizing gives $\xi^\star \equiv \forall x \forall v\, \varphi(z, x, F_y(z), F_u(z, x), v, F_w(z, x, v))$, where $F_y(z), F_u(z, x)$ and $F_w(z, x, v)$ are Skolem functions for y, u and w respectively. Using the notation introduced above, we have

- $\xi^\star_y(z, y) \;\equiv\; \forall x \exists u \forall v \exists w\, \varphi(z, x, y, u, v, w)$
- $\xi^\star_u(z, x, u) \;\equiv\; \forall v \exists w\, \varphi(z, x, F_y(z), u, v, w)$
- $\xi^\star_w(z, x, v, w) \;\equiv\; \varphi(z, x, F_y(z), F_u(z, x), v, w)$

By virtue of Skolemization, we know that for every structure $\mathfrak{M}$ over which ξ is interpreted, there exists an expansion $\mathfrak{M}^\star$ that interprets F_y, F_u and F_w such that the following hold.

- $\forall z \big(\exists y\, \xi^\star_y(z, y) \Leftrightarrow \xi^\star_y[y \mapsto F_y(z)]\big)$
- $\forall z \forall x \left(\exists u\, \xi^\star_u(z, x, u) \Leftrightarrow \xi^\star_u[u \mapsto F_u(z, x)]\right)$
- $\forall z \forall x \forall v \left(\exists w \xi^\star_w(z, x, v, w) \Leftrightarrow \xi^\star_w[w \mapsto F_w(z, x, v)]\right)$

Let $\mathcal{V}^\star$ be the expansion of $\mathcal{V}$ obtained by adding all Skolem function and constant symbols in $\xi^\star$ to $\mathcal{V}$. In general, a $\mathcal{V}$-structure $\mathfrak{M}$ over which $\xi(\mathbf{Z})$ is interpreted can be expanded to a $\mathcal{V}^\star$-structure by adding interpretations of Skolem functions for all existential variables in $\xi(\mathbf{Z})$. However, not every such expansion of $\mathfrak{M}$ may model the sentence (1) above for every existential variable $y_{i,j}$. Skolemization guarantees that there exists at least one "correct" expansion $\mathfrak{M}^\star$ of $\mathfrak{M}$ that does so. We call the interpretation of Skolem functions in such a "correct" expansion as an *$\mathfrak{M}$-interpretation* of the Skolem functions. There may be multiple "correct" expansions of $\mathfrak{M}$, and hence multiple $\mathfrak{M}$-interpretations of Skolem functions. Skolemization guarantees the existence of at least one $\mathfrak{M}$-interpretation of all Skolem functions/constants; however, it doesn't tell us whether these are computable interpretations, and if so, can we algorithmically synthesize the interpretation as a halting Turing machine? These are two central questions that concern us in this paper.

Sometimes, given a $\mathcal{V}$-formulas ξ, we can find an $\mathfrak{M}$-interpretation of Skolem functions that works in the same way for all computable structures $\mathfrak{M}$ over which ξ is interpreted (modulo differences in interpreting predicates and functions). Formally, suppose there exists a halting Turing machine M_ξ with access to oracles that compute the interpretations of predicates and functions in $\mathfrak{M}$, and suppose M_ξ computes an $\mathfrak{M}$-interpretation of Skolem functions for all existential variables in ξ, and for all $\mathcal{V}$-structures $\mathfrak{M}$. Then, we say that ξ admits a *uniform representation* of $\mathfrak{M}$-interpretations of Skolem functions.

Model theory. We use $\mathcal{V}(\mathfrak{M})$ to denote the expansion of $\mathcal{V}$ obtained by adding a fresh constant symbol c_e for every element $e \in U^{\mathfrak{M}}$, if not already present in $\mathcal{V}$. Clearly, if $U^{\mathfrak{M}}$ and $\mathcal{V}$ are countable, so is $\mathcal{V}(\mathfrak{M})$. We use $\mathfrak{M}_C$ to denote the expansion of $\mathfrak{M}$ to a $\mathcal{V}(\mathfrak{M})$-structure that interprets the additional constants in $\mathcal{V}(\mathfrak{M})$ in the natural way, i.e. c_e is interpreted to have the value e, for all $e \in U^{\mathfrak{M}}$. The *elementary diagram* of $\mathfrak{M}$, denoted $\mathcal{ED}(\mathfrak{M})$, is the set of all $\mathcal{V}(\mathfrak{M})$-sentences ξ such that $\mathfrak{M}_C \models \xi$. The *diagram* of $\mathfrak{M}$, denoted $\mathcal{D}(\mathfrak{M})$, is the set of all literals in $\mathcal{ED}(\mathfrak{M})$, i.e. the set of all atomic ground formulas that hold in $\mathfrak{M}_C$. Clearly, $\mathcal{D}(\mathfrak{M}) \subseteq \mathcal{ED}(\mathfrak{M})$. A set Γ of $\mathcal{V}$-sentences is called a *$\mathcal{V}$-theory* if it is consistent, i.e. there exists a $\mathcal{V}$-structure that serves as a model for every sentence in Γ. Given a $\mathcal{V}$-structure $\mathfrak{M}$, the set of all first order $\mathcal{V}$-sentences ξ such that $\mathfrak{M} \models \xi$ is called the *theory of* $\mathfrak{M}$, denoted $Th(\mathfrak{M})$. Note that both $\mathcal{ED}(\mathfrak{M})$ and $\mathcal{D}(\mathfrak{M})$ are $\mathcal{V}$-theories, and $\mathcal{ED}(\mathfrak{M}) = Th(\mathfrak{M}_C)$, where $Th(\mathfrak{M}_C)$ is the $\mathcal{V}(\mathfrak{M})$-theory of $\mathfrak{M}_C$. We say that a $\mathcal{V}$-theory Γ is *decidable* iff there exists a Turing machine that takes as input an arbitrary $\mathcal{V}$-sentence ξ and always halts and correctly reports whether $\xi \in \Gamma$ or not. If $\mathfrak{M}$ is a computable structure, it follows immediately that $\mathcal{D}(\mathfrak{M})$ is a decidable theory, but $\mathcal{ED}(\mathfrak{M})$ is not necessarily so.

A $\mathcal{V}$-theory Γ is said to admit *quantifier elimination* if for every $\mathcal{V}$-formula $\xi(\mathbf{Z})$ with free variables $\mathbf{Z}$, there exists a semantically equivalent quantifier-free $\mathcal{V}$-formula $\xi^\#(\mathbf{Z})$ such that the sentence $\forall \mathbf{Z}\,\big(\xi(\mathbf{Z}) \Leftrightarrow \xi^\#(\mathbf{Z})\big)$ is in Γ. If, in addition, there exists a Turing machine that takes an arbitrary $\mathcal{V}$-formula (ξ) as input and computes its quantifier-eliminated form ($\xi^\#$) and halts, we say that Γ admits *effective quantifier elimination*[1]. For a $\mathcal{V}$-structure $\mathfrak{M}$, we say that $Th(\mathfrak{M})$ admits *effective constraint solving* if there exists a Turing machine that takes a $\mathcal{V}$-formula $\xi(\mathbf{Z})$ with free variables $\mathbf{Z}$ as input and halts after reporting one of two things: (i) a $|\mathbf{Z}|$-tuple σ of elements from $U^{\mathfrak{M}}$ such that $\mathfrak{M} \models \xi(\sigma)$, or (ii) no such $|\mathbf{Z}|$-tuple of elements from $U^{\mathfrak{M}}$ exists. Note that the formula $\xi(\mathbf{Z})$ may have quantifiers in general. In case the above Turing machine exists only if $\xi(\mathbf{Z})$ is quantifier-free, we say that $Th(\mathfrak{M})$ admits *effective quantifier-free constraint solving.* Clearly, if $Th(M)$ admits effective quantifier elimination and effective quantifier-free constraint solving, then it also admits effective constraint solving.

[1] There is a technique, popularly called "Morleyization", that trivially makes a theory admit effective quantifier elimination by expanding the vocabulary to include a separate predicate symbol for each $\mathcal{V}$-formula. For purposes of this paper, we disallow expansion of the vocabulary (and hence "Morleyization") during effective quantifier elimination.

3 An illustrative example

Consider the vocabulary $\mathcal{V} = \{P, c, d\}$, where P is a binary predicate symbol, and c and d are constants, and the first-order $\mathcal{V}$-sentence $\xi \equiv \forall x \exists y P(x,y) \wedge \big(P(x,c) \vee P(x,d)\big)$. We will use $\varphi(x,y)$ to denote the matrix of the above formula, i.e. $P(x,y) \wedge \big(P(x,c) \vee P(x,d)\big)$. On Skolemizing ξ we get $\xi^\star \equiv \forall x \varphi(x, F_y(x))$, where F_y is a fresh unary Skolem function symbol. Let $\mathfrak{M}$ be a computable $\mathcal{V}$-structure. We now ask if there exists an algorithm $\mathcal{A}^{[F]}$ that serves as a computable interpretation of $F_y : U^{\mathfrak{M}} \to U^{\mathfrak{M}}$. A careful examination of ξ and $\xi^\star$ reveals that such an algorithm indeed exists. Specifically, the algorithm (represented informally as an imperative "program" for ease of understanding) "**input**(x)**; if** $\mathtt{P}^{\mathfrak{M}}(\mathtt{x}, \mathtt{c}^{\mathfrak{M}})$ **then return** $c^{\mathfrak{M}}$ **else return** $d^{\mathfrak{M}}$" takes as input $x \in U^{\mathfrak{M}}$ and returns either $c^{\mathfrak{M}}$ or $d^{\mathfrak{M}}$ depending on whether $P^{\mathfrak{M}}(x, c^{\mathfrak{M}})$ evaluates to true or false. If we let this algorithm interpret F_y in the expansion $\mathfrak{M}^\star$ of $\mathfrak{M}$, then it is not hard to see that we indeed have $\mathfrak{M}^\star \models \forall x \big(\exists y \varphi(x,y) \Leftrightarrow \varphi(x, F_y(x)\big)$.

However, is this always possible? Consider the $\mathcal{V}$-formula $\alpha \equiv \forall x \exists y\, P(x,y)$ instead of ξ, whose Skolemized version is $\alpha^\star \equiv \forall x\, P(x, F_y(x))$. As we show in Section 5, it is impossible to obtain a computable $\mathfrak{M}$-interpretation of the Skolem function $F_y(x)$ in this case for all $\mathcal{V}$-structures $\mathfrak{M}$.

There are several observations that one can now make. Clearly, algorithm $\mathcal{A}^{[F]}$ described above is specific to the formula ξ; a different formula would have required a different algorithm to be designed for its Skolem function(s). Interestingly, algorithm $\mathcal{A}^{[F]}$ also requires access to the interpretations of c, d and P in the $\mathcal{V}$-structure $\mathfrak{M}$ on which ξ is interpreted. Since we are given an effectively computable interpretation of P in $\mathfrak{M}$, there exists an algorithm $\mathcal{A}^{[P]}$ to compute $P^{\mathfrak{M}}$. Algorithm $\mathcal{A}^{[F]}$ effectively uses $\mathcal{A}^{[P]}$ as a sub-routine to compute the value of $F_y(x)$ for every $x \in U^{\mathfrak{M}}$. Note that if the interpretation of P (in perhaps a different $\mathcal{V}$-structure $\mathfrak{M}'$) was not effectively computable, the "program" above would not serve as an effectively computable interpretation of F_y. This underlines the importance of effectively computable structures in the synthesis of Skolem functions.

It is easy to see that "**input(x); if** $\mathtt{P}^{\mathfrak{M}}(\mathtt{x}, \mathtt{c}^{\mathfrak{M}})$ **then return** $c^{\mathfrak{M}}$ **else return** $d^{\mathfrak{M}}$" *uniformly* serves as a computable interpretation of F_y in every computable $\mathcal{V}$-structure $\mathfrak{M}$ over which ξ is interpreted. Regardless of the actual structure $\mathfrak{M}$, a computable $\mathfrak{M}$-interpretation of F_y is obtained by invoking algorithms to compute interpretations of P, c, d in $\mathfrak{M}$ as sub-routines. Thus we get a uniform representation of an $\mathfrak{M}$-interpretation of F_y.

Finally, the interpretation of Skolem function F discussed above is represented as an algorithm, and not as a $\mathcal{V}$-term. Is it possible to obtain a $\mathcal{V}$-term that uniformly represents an $\mathfrak{M}$-interpretation of F_y in this case? To answer this, first observe that there are only two terms, viz. c and d, that can be formed using $\mathcal{V}$. If one of these terms serves as a uniform $\mathfrak{M}$-interpretation of F_y, choose a structure $\mathfrak{M}$ as follows: $U^{\mathfrak{M}} = \{a_0, a_1\}, c^{\mathfrak{M}} = a_0, d^{\mathfrak{M}} = a_1, P^{\mathfrak{M}}(a_0, a_0) = P^{\mathfrak{M}}(a_1, a_1) = \mathsf{false}$ and $P^{\mathfrak{M}}(a_0, a_1) = P^{\mathfrak{M}}(a_1, a_0) = \mathsf{true}$. Clearly $\mathfrak{M} \models \forall x \exists y \varphi(x,y)$. However, with $F_y(x) = c$ (or $F_y(x) = d$), we have $MM^\star \not\models \forall x \big(\exists y \varphi(x,y) \;\Leftrightarrow\; \varphi(x, F(x))\big)$. Thus even when an effectively computable interpretation of a Skolem function exists, it may not be representable as a term over $\mathcal{V}$.

4 Problem statement

We now formulate the primary questions that we wish to address in this paper.

1. Given a vocabulary $\mathcal{V}$, a $\mathcal{V}$-formula $\xi(\mathbf{Z})$ in prenex normal form and a computable $\mathcal{V}$-structure $\mathfrak{M}$, the SkolemExist problem asks if there exists a *computable* $\mathfrak{M}$-interpretation of Skolem functions for all existential variables in ξ. We have already seen in Section 3 that there are positive instances of SkolemExist. We ask if there are negative instances as well, i.e. there is no computable $\mathfrak{M}$-interpretation of Skolem functions.

2. Next, we ask if SkolemExist is decidable.
3. We then consider special cases where either the formula $\xi(\mathbf{Z})$ or structure $\mathfrak{M}$ is fixed, and ask if it is possible to characterize the class of problems where the SkolemExist problem has a positive answer.
4. In cases where the SkolemExist problem has a positive answer, we ask the following:
   a. Does there exist an algorithm to synthesize computable $\mathfrak{M}$-interpretations of Skolem functions? We call this problem SkolemSynthesis and consider two variants of it, where either (i) $\mathcal{V}$ and $\xi(\mathbf{Z})$ are fixed and $\mathfrak{M}$ is the input of SkolemSynthesis, or (ii) $\mathcal{V}$ and $\mathfrak{M}$ is fixed and ξ is the input of SkolemSynthesis.
   b. Is it possible to obtain finite uniform representations of $\mathfrak{M}$-interpretations of Skolem functions, and if so, can we obtain these as $\mathcal{V}$-terms?
   c. In case SkolemExist has a positive answer, can we give bounds on the worst-case running time of computable $\mathfrak{M}$-interpretations of Skolem functions?

Note that SkolemSynthesis is not meaningful in cases where SkolemExist has a negative answer. Hence, we don't try to answer SkolemSynthesis in negative instances of SkolemExist. Moreover, all the above problems except the last one is trivial if the universe $U^{\mathfrak{M}}$ is finite. Therefore, we focus mostly on structures with countably infinite universe.

5 Hardness of SkolemExist and SkolemSynthesis

We have already seen a positive instance (i.e. problem instance with positive answer) of SkolemExist in Section 3. The following lemma shows that SkolemExist always has a positive answer if all Skolem functions are Skolem constants. In the following, we use $(\mathcal{V}, \mathfrak{M}, \xi)$ to denote an instance of SkolemExist, where $\mathcal{V}$ is a vocabulary, $\mathfrak{M}$ is a computable $\mathcal{V}$-structure and ξ is a $\mathcal{V}$-formula.

▶ **Lemma 2.** *For every vocabulary $\mathcal{V}$, every computable $\mathcal{V}$-structure $\mathfrak{M}$ and every $\mathcal{V}$-sentence $\exists \mathbf{Y}\, \varphi(\mathbf{Y})$, where φ is a quantifier-free $\mathcal{V}$-formula with free variables in $\mathbf{Y}$, the instance $(\mathcal{V}, \mathfrak{M}, \xi)$ of SkolemExist has a positive answer.*

However, there are negative instances of SkolemExist, even with a restricted vocabulary.

▶ **Theorem 3.** *There exists a negative instance of SkolemExist where the vocabulary has a single binary predicate.*

Now that we know there are positive and negative instances of SkolemExist, we ask if SkolemExist is decidable. Unfortunately, we obtain a negative answer in general.

▶ **Theorem 4.** *SkolemExist is undecidable.*

Proof. We prove this theorem by contradiction. Suppose, if possible, there exists a halting Turing machine M that takes as inputs a vocabulary $\mathcal{V}$, a $\mathcal{V}$-formula $\xi(\mathbf{Z})$ and a computable $\mathcal{V}$-structure $\mathfrak{M}$, and decides if there exists a computable $\mathfrak{M}$-interpretation of Skolem functions for all existential variables in $\xi(\mathbf{Z})$. We show below that we can use M to effectively decide if an arbitrary Turing machine, say TM_i, halts on the empty tape.

Consider $\mathcal{V} = \{Q, a\}$, where Q is a binary predicate symbol and a is a constant symbol. For each $i \in \mathbb{N}$, define $\mathfrak{M}_i$ to be a $\mathcal{V}$-structure such that $\mathbf{U}^{\mathfrak{M}_i} = \mathbb{N}$, $a^{\mathfrak{M}_i} = i$ and $Q^{\mathfrak{M}_i}(u, v) = \mathsf{true}$ for $u, v \in \mathbb{N}$ iff the Turing machine TM_u halts on the empty tape within v steps. It is easy to see that each $\mathfrak{M}_j$ is a computable $\mathcal{V}$-structure. We also define the $\mathcal{V}$-sentence $\xi \equiv \exists s \forall u \exists x \left(\left(Q(a, s) \wedge (x = u)\right) \vee \left(\neg Q(a, u) \wedge Q(u, x)\right)\right)$. Skolemizing this formula gives $\xi^\star \equiv \forall u \left(\left(Q(a, c_s) \wedge (F_x(u) = u)\right) \vee \left(\neg Q(a, u) \wedge Q(u, F_x(u))\right)\right)$, where c_s is a Skolem constant for s, and F_x is a Skolem function for x. From the guarantee of Skolemization, the following must hold:

- $\exists s \forall u \exists x \left(\left(Q(a,s) \wedge (x = u)\right) \vee \left(\neg Q(a,u) \wedge Q(u,x)\right)\right) \Leftrightarrow \forall u \exists x \left(\left(Q(a,c_s) \wedge (x = u)\right) \vee \left(\neg Q(a,u) \wedge Q(u,x)\right)\right)$
- $\forall u \big(\exists x \left(\left(Q(a,c_s) \wedge (x = u)\right) \vee \left(\neg Q(a,u) \wedge Q(u,x)\right)\right) \Leftrightarrow \left(\left(Q(a,c_s) \wedge (F_x(u) = u)\right) \vee \left(\neg Q(a,u) \wedge Q(u,F_x(u))\right)\right)\big)$

We now consider two cases.

- Suppose TM_i halts on the empty tape after $p \in \mathbb{N}$ steps. Then $\mathfrak{M}_i \models Q(a,p)$. In this case, by choosing the $c_s^{\mathfrak{M}_i} = p$ and by choosing $F_x^{\mathfrak{M}_i}(u) = u$ for all $u \in \mathbb{N}$, both the above guarantees of Skolemization are easily seen to hold. Clearly, the Skolem functions have computable interpretations in this case.
- If TM_i doesn't halt on the empty tape, then $\mathfrak{M}_i \models \forall u\, \neg Q(a,u)$. In this case, we choose an arbitary value, say 0, for s. However, for the guarantee of Skolemization to hold, we must have the following: for every $u \in \mathbb{N}$, if $\exists x Q(u,x)$ holds (i.e. TM_u halts on the empty tape), then $Q(u, F_x^{\mathfrak{M}_i}(u))$ must also hold (i.e. TM_u must also halt in $F_x^{\mathfrak{M}_i}(u)$ steps). Clearly, such an interpretation $F_x^{\mathfrak{M}_i}$ is not computable, as otherwise it can be used to decide the halting problem.

The above reasoning shows that there exist computable $\mathfrak{M}_i$-interpretations of all Skolem functions of existential variables in ξ iff TM_i halts on empty tape. Thus, if we feed the instance $(\mathcal{V}, \mathfrak{M}_i, \xi)$ as input to the supposed Turing machine M that decides SkolemExist, we can decide if TM_i halts on the empty tape, for every $i \in \mathbb{N}$. This gives a decision procedure for the halting problem on the empty tape – an impossibility! ◀

It is easy to see that the proof of Theorem 4 can be repeated with $\xi \equiv \forall u \exists s \exists x \left(\left(Q(a,s) \wedge (x = u)\right) \vee \left(\neg Q(a,u) \wedge Q(u,x)\right)\right)$ as well. This gives the following interesting result.

▶ **Theorem 5.** *If the vocabulary contains a binary predicate and a constant, SkolemExist is undecidable for the quantifier prefix classes $\exists\forall\exists$ and $\forall\exists\exists$. However it is decidable for the class $\exists^+\forall^*$.*

The second part of the above Theorem follows from an easy generalization of the proof of Lemma 2. This leaves only the case of $\forall\exists$ quantifier prefix, for which the decidability of SkolemExist remains open. We consider the case of the vocabulary having only monadic predicates later in Theorem 7.

The above negative results motivate us to consider special cases of SkolemExist and SkolemSynthesis, where either the $\mathcal{V}$-formula $\xi(\mathbf{Z})$ or the $\mathcal{V}$-structure $\mathfrak{M}$ is fixed.

Fixing the formula. The proof of Theorem 4 is quite damning: even if we allow the possibility of a potentially different algorithm, say $A_{\mathcal{V},\xi}$, for deciding SkolemExist for each combination of $\mathcal{V}$ and ξ, we cannot hope to have an algorithm $A_{\mathcal{V},\xi}$ for every $(\mathcal{V}, \xi)$ pair. This is because in the proof of Theorem 4, we had indeed kept the vocabulary and formula fixed. This leaves only a few questions to be investigated if we fix the vocabulary and formula. If we consider $\mathcal{V}$ and ξ as fixed, the $\mathcal{V}$-structure $\mathfrak{M}$ is the only input to our problems of interest. The following theorem shows that SkolemSynthesis cannot be answered positively in this case even under fairly strong conditions.

Recall from Lemma 2 that SkolemExist has a positive answer if all Skolem functions are Skolem constants. Hence, by choosing ξ to be a $\mathcal{V}$-sentence with only existential quantifiers, we are guaranteed that all problem instances are positive instances of SkolemExist.

▶ **Theorem 6.** *There exists a vocabulary $\mathcal{V}$, a $\mathcal{V}$-sentence ξ and a family of $\mathcal{V}$-structures $\mathcal{F} = \{\mathfrak{M}_i \mid i \in \mathbb{N}\}$, such that $(\mathcal{V}, \mathfrak{M}_i, \xi)$ is a positive instance of SkolemExist for all $i \in \mathbb{N}$, yet there is no uniform representation of $\mathfrak{M}_i$-interpretations of the Skolem constants. Additionally, the SkolemSynthesis problem has a negative answer for the class of problem instances $\{(\mathcal{V}, \xi, \mathfrak{M}_i) \mid i \in \mathbb{N}\}$.*

It is interesting to ask now if there is a characterization of $\mathcal{V}$-formulas, such that for each $\mathcal{V}$-formula satisfying this characterization, the SkolemExist and SkolemSynthesis problems have positive answers for all $\mathcal{V}$-structures. The proof of Theorem 3 tells us that we must disallow binary predicates and $\forall\exists$ blocks in the quantifier prefix, which severely restricts the vocabulary and formulas. What happens if we allow a relational vocabulary with only monadic predicates (Löwenheim class with equality) [9]?

▶ **Theorem 7.** *Let the vocabulary $\mathcal{V}$ contain only monadic predicates and equality. Then SkolemExist has a positive answer, but not so for SkolemSynthesis.*

Proof. With k monadic predicates, the universe can be partitioned into 2^k equivalence classes based on predicate valuations. By an argument (based on Ehrenfeucht-Fraisse games) similar to that used to prove small-model property of Löwenheim class (see [9]), if a prenex formula ξ has quantifier rank r, the range of each Skolem function can be restricted to $\leq r.2^k$ elements. Using an argument similar to that in proof of Lemma 2, there exists a TM that enumerates the required set, say S, of $\leq r.2^k$ elements. Since elements of an equivalence class can only be distinguished using $=$, for each Skolem function of arity p, we must search for its correct interpretation over all $S^p \rightarrow S$ mappings. Since there are finitely many such mappings, we can enumerate the TMs computing these mappings, and one of them must effectively serve as the correct interpretation for the Skolem function under consideration. Since ξ has finitely many existential variables, it follows that there exists computable interpretations of all Skolem functions in ξ.

To see why SkolemSynthesis has a negative answer in general even with one monadic predicate P and one existential quantifier, consider $\mathcal{V} = \{P\}$, $\xi \equiv \exists x P(x)$, and a structure $\mathfrak{M}_i$ having universe $\mathbb{N}$ and $P^{\mathfrak{M}_i}(x) =$ true iff TM_i halts on empty tape within x steps. If there exists an algorithm to synthesize computable M_i-interpretations of the Skolem constant for x in ξ, we can use it to decide if TM_i halts on empty tape – an impossibility! Thus, we must disallow even monadic predicates if we want to characterize V-formulas that admit positive answer to SkolemSynthesis for all $\mathcal{V}$-structures. ◀

Fixing the structure. We now fix the structure $\mathfrak{M}$ (and vocabulary $\mathcal{V}$) and take the formula ξ as the only input of our problems of interest. Since the structure $\mathfrak{M}$ is fixed, we use the notation $\mathcal{U}$ for $U^{\mathfrak{M}}$ henceforth. Theorem 3 already shows that even when the structure is fixed, the SkolemExist problem has a negative instance. However, the $\mathcal{V}$-structure used in that proof may appear hand-crafted. This leads us to ask if there is a "natural" vocabulary $\mathcal{V}$ and $\mathcal{V}$-structure $\mathfrak{M}$, such that SkolemExist has a negative instance when considering $\mathcal{V}$-formulas. It turns out that this is indeed the case, and we show it by appealing to the classical Matiyasevich-Robinson-Davis-Putnam (MRDP) theorem [11].

▶ **Proposition 8.** *Skolem functions for the first order theory of natural numbers over the vocabulary $\{\times, +, 0, 1\}$ do not admit computable interpretations.*

Finally, in the setting of a fixed $\mathcal{V}$-structure $\mathfrak{M}$, even if SkolemExist is answered in the positive for all $\mathcal{V}$-formulas in a class Ξ, the SkolemSynthesis problem may have a negative answer for the class of problem instances $\{(\mathcal{V}, \mathfrak{M}, \xi) \mid \xi \in \Xi\}$.

▶ **Theorem 9.** *There exists a vocabulary $\mathcal{V}$, a $\mathcal{V}$-structure $\mathfrak{M}$ and a class of $\mathcal{V}$-sentences $\Xi = \{\xi_i \mid i \in \mathbb{N}\}$ s.t., $(\mathcal{V}, \mathfrak{M}, \xi)$ is a positive instance of SkolemExist for all $\xi_i \in \Xi$, yet SkolemSynthesis has a negative answer for the class of instances $\{(\mathcal{V}, \mathfrak{M}, \xi_i) \mid i \in \mathbb{N}\}$.*

6 Necessary & sufficient condition for synthesizing Skolem functions

Given these strong negative results is there hope for proving existence and synthesizability of computable interpretations for Skolem functions. Indeed, there do exist many natural theories where computable interpretations of Skolem functions exist and can indeed be synthesized, e.g., Boolean case, Presburger arithmetic etc. So, what determines when a $\mathcal{V}$-theory admits effective synthesis of computable interpretations of Skolem functions for all $\mathcal{V}$-formulas? Our first positive result is a surprising characterization of a *necessary and sufficient* condition for algorithmic synthesis of computable interpretations of Skolem functions.

▶ **Theorem 10.** *Let $\mathfrak{M}$ be a computable $\mathcal{V}$-structure for vocabulary $\mathcal{V}$. The* SkolemSynthesis *problem for $\mathcal{V}$-formulas, i.e. for problem instances $\{(\mathcal{V}, \mathfrak{M}, \xi) \mid \xi$ is a $\mathcal{V}$-formula$\}$, has a positive answer iff $\mathcal{ED}(\mathfrak{M})$ is decidable.*

Proof. ($\Longleftarrow$) Let $\xi(\mathbf{Z})$ be a $\mathcal{V}$-formula with free variables $\mathbf{Z}$, where $\xi(\mathbf{Z}) \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1 \ldots \forall \mathbf{X}_n \exists \mathbf{Y}_n$
$\xi_n(\mathbf{Z}, \mathbf{X}_1, \mathbf{Y}_1, \ldots, \mathbf{X}_n, \mathbf{Y}_n)$, where $\mathbf{X}_1, \ldots, \mathbf{X}_n$ are n sequences of universally quantified variables, $\mathbf{Y}_1, \ldots \mathbf{Y}_n$ are sequences of existentially quantified variables, $\mathbf{Z}$ is a sequence of free variables and ξ_n is quantifier-free. We will show that there is an algorithm, that for every $i \in \{1, \ldots, n\}$, takes as input a $(|\mathbf{Z}|+|\mathbf{X}_1|+\ldots+|\mathbf{X}_i|)$ tuple of values from the universe $\mathcal{U}$, say, $\mu \in \mathcal{U}^{|\mathbf{Z}|}, \sigma_1 \in \mathcal{U}^{|\mathbf{X}_1|}, \ldots, \sigma_i \in \mathcal{U}^{|\mathbf{X}_i|}$ and halts after computing a $(|\mathbf{Y}_1|+\ldots+|\mathbf{Y}_i|)$-dimensional vector of values, $\mathbf{F}_1(\mu, \sigma_1) \in \mathcal{U}^{|Y_1|}, \ldots \mathbf{F}_i(\mu, \sigma_1, \ldots \sigma_i) \in \mathcal{U}^{|\mathbf{Y}_i|}$ where for each $1 \leq j \leq i$, $\mathbf{F}_j$ is a $|\mathbf{Y}_j|$-dimensional vector of Skolem functions, each of arity $|\mathbf{Z}| + |\mathbf{X}_1| + \ldots + |\mathbf{X}_j|$.

The proof is by induction on i. For $i = 1$, let $\xi(\mathbf{Z}) \equiv \forall \mathbf{X}_1 \exists \mathbf{Y}_1\, \xi_1(\mathbf{Z}, \mathbf{X}_1, \mathbf{Y}_1)$, where ξ_1 has one less number of quantifier alternations than ξ. On Skolemizing, we get $\xi^\star(\mathbf{Z}) \equiv \forall \mathbf{X}_1\, \xi_1(\mathbf{Z}, \mathbf{X}_1, \mathbf{F}_1(\mathbf{Z}, \mathbf{X}_1))$, where $\mathbf{F}_1$ is a $|\mathbf{Y}_1|$-dimensional vector of Skolem functions each of arity $|\mathbf{Z}| + |\mathbf{X}_1|$. We now design a Turing machine (or algorithm) M_1 that takes any $|\mathbf{Z}|+|\mathbf{X}_1|$-tuple of elements from $\mathcal{U}$, say (μ, σ_1), as input and halts after computing $\mathbf{F}_1(\mu, \sigma_1)$:

(a) It first determines if $\exists \mathbf{Y}_1\, \xi_1(\mu, \sigma_1, \mathbf{Y}_1)$ holds, using the decision procedure for $\mathcal{ED}(\mathfrak{M})$.

(b) If the answer to the above question is "Yes", the machine M_1 recursively enumerates $|\mathbf{Y}_1|$-tuples of elements of $\mathcal{U}$, and for each tuple ν thus enumerated, it checks if $\xi_1(\mu, \sigma_1, \nu)$ evaluates to true. Again the decidability of $\mathcal{ED}(\mathfrak{M})$ ensures that this check can also be effectively done. The machine M_1 outputs the first (in recursive enumeration order) element of $\mathcal{U}^{|\mathbf{Y}_1|}$, for which $\xi_1(\mu, \sigma_1, \nu)$ is true as $\mathbf{F}_1(\mu, \sigma_1)$, and halts.

(c) If the answer is "No", i.e. there is no $\nu \in \mathcal{U}^{|\mathbf{Y}_1|}$ s.t. $\xi_1(\mu, \sigma_1, \nu)$ is true, M_1 outputs the first (in recursive enumeration order) tuple of $\mathcal{U}^{|\mathbf{Y}_1|}$ as $\mathbf{F}_1(\mu, \sigma_1)$, and halts.

It is easy to verify that the vector of functions $\mathbf{F}_1$ computed by M_1 satisfies $\forall \mathbf{X}_1 \big(\exists \mathbf{Y}_1\, \xi_1(\mathbf{Z}, \mathbf{X}_1, \mathbf{Y}_1) \Leftrightarrow \xi_1(\mathbf{Z}, \mathbf{X}_1, \mathbf{F}_1(\mathbf{Z}, \mathbf{X}_1))\big)$ for every valuation of the free variables $\mathbf{Z}$ in $\mathcal{U}^{|\mathbf{Z}|}$, i.e., we have a (correct) $\mathfrak{M}$-interpretation of Skolem function $\mathbf{F}_1$. This completes the base case for $i = 1$.

For the general case of $i \geq 1$, we write $\xi(\mathbf{Z})$ as $\forall \mathbf{X}_1 \exists \mathbf{Y}_1 \ldots \forall \mathbf{X}_i \exists \mathbf{Y}_i \xi_i(\mathbf{Z}, \mathbf{X}_1, \mathbf{Y}_1, \ldots \mathbf{X}_i, \mathbf{Y}_i)$, where ξ_i is a formula with i less $\forall^*\exists^*$ blocks than ξ. By induction hypothesis, we know that there exists a Turing machine M_i that takes as input any values for free variables $\mathbf{Z}$ and universally quantified variables $\mathbf{X}_1, \ldots \mathbf{X}_i$ and outputs values for $\mathbf{Y}_1, \ldots \mathbf{Y}_i$ so that they correspond to outputs of (correct) $\mathfrak{M}$-interpretations of vectors of Skolem functions $\mathbf{F}_1, \ldots \mathbf{F}_i$.

We need to show the existence of a computable interpretation of the vector of Skolem functions $\mathbf{F}_{i+1}$ for $\mathbf{Y}_{i+1}$. Thus, we are given a $(|\mathbf{Z}| + |\mathbf{X}_1| + \ldots + |\mathbf{X}_i| + |\mathbf{X}_{i+1}|)$-tuple of values from $\mathcal{U}$, and we need to show how to define a Turing machine M_{i+1} that takes this vector as input and halts after computing values of vectors of Skolem functions $\mathbf{F}_1(\mu, \sigma_1), \ldots \mathbf{F}_{i+1}(\mu, \sigma_1, \ldots \sigma_i, \sigma_{i+1})$. Let $(\mu, \sigma_1, \ldots, \sigma_i, \sigma_{i+1})$ be the given set of input values. The Turing machine M_{i+1} first simulates M_i on input $(\mu, \sigma_1, \ldots, \sigma_i)$. This returns i vectors of values $\nu_1 = \mathbf{F}_1(\mu, \sigma_1) \in \mathcal{U}^{|Y_1|}, \ldots \nu_i = \mathbf{F}_i(\mu, \sigma_1, \ldots \sigma_i) \in \mathcal{U}^{|Y_i|}$ such that each of $\mathbf{F}_1, \ldots, \mathbf{F}_i$ is a Skolem function vector. Plugging in all these values in ξ_i gives a sentence $\hat{\xi}_i = \xi_i[\mathbf{Z} \mapsto \mu, \mathbf{X}_1 \mapsto \sigma_1, \mathbf{Y}_1 \mapsto \nu_1, \ldots, \mathbf{X}_i \mapsto \sigma_i, \mathbf{Y}_i \mapsto \nu_i]$. Observe that $\hat{\xi}_i \equiv \forall \mathbf{X}_{i+1} \exists \mathbf{Y}_{i+1} \hat{\xi}_{i+1}(\mathbf{X}_{i+1}, \mathbf{Y}_{i+1})$ for some formula $\hat{\xi}_{i+1}(\mathbf{X}_{i+1}, \mathbf{Y}_{i+1})$. We can then apply the same argument as in the base case above and conclude.

Note that the values of $\mathbf{Z}, \mathbf{X}_1, \ldots \mathbf{X}_n$ used above were arbitrary tuples from $\mathcal{U}^{|\mathbf{Z}|}$, $\mathcal{U}^{|\mathbf{X}_1|}, \ldots \mathcal{U}^{|\mathbf{X}_n|}$. Hence lifting the notation introduced in sentence (1) of Section 2 to talk about vector of variables $\mathbf{Y}_{i+1}$ instead of single variables $y_{i,j}$, we conclude that the vector of functions $\mathbf{F}_{i+1}$ computed by Turing machine M_{i+1} satisfies $\forall \mathbf{Z} \forall \mathbf{X}_1 \ldots \forall \mathbf{X}_{i+1}$ $\big(\exists \mathbf{Y}_{i+1} \xi^{\star}_{\mathbf{Y}_{i+1}} \Leftrightarrow \xi^{\star}_{\mathbf{Y}_{i+1}}[\mathbf{Y}_{i+1} \mapsto \mathbf{F}_{i+1}(\mathbf{Z}, \mathbf{X}_1, \ldots \mathbf{X}_{i+1})]\big)$. Thus, $\mathbf{F}_{i+1}$ gives a (correct) $\mathfrak{M}$-interpretation of a vector of Skolem functions for $\mathbf{Y}_{i+1}$, which completes the proof.

($\Rightarrow$) In the other direction, we will show that if there exists a halting Turing machine, say M, that synthesizes computable $\mathfrak{M}$-interpretations of Skolem functions for all existential variables in all $\mathcal{V}$-formulas, then $\mathcal{ED}(\mathfrak{M})$ must be decidable. We assume that there are at least two elements in $\mathcal{U}$; let's call them d_1, d_2. To show that $\mathcal{ED}(\mathfrak{M})$ is decidable, we need to show a decision procedure for the $\mathcal{V}(\mathfrak{M})$-theory of $\mathfrak{M}_C$. Consider any $\mathcal{V}(\mathfrak{M})$-sentence φ. This sentence may have finitely many constants that are not in $\mathcal{V}$. Let us say $c_1, \ldots c_k$ are these constants, for some non-negative integer k. We introduce k fresh variables $y_1, \ldots y_k$ and define φ' to be the formula obtained by taking φ and replacing each occurrence of the constant c_i by variable y_i respectively. Note that φ' is a $\mathcal{V}$-formula. We introduce 3 more fresh variables x, z_1, z_2 and consider the sentence $\psi \equiv \forall y_1 \ldots \forall y_k \forall z_1 \forall z_2 \exists x \big(((x = z_1) \wedge \varphi') \vee ((x = z_2) \wedge \neg \varphi')\big)$. We can easily rewrite this formula in prenex normal form, but doing so still leaves x as the leftmost existentially quantified variable. We now feed this prenex normal form of ψ as the input to Turing machine M (that synthesizes computable interpretations of Skolem functions for all existential variables in all $\mathcal{V}$-formulas). Let the computable interpretation of the Skolem function for x, as output by M, be the Turing machine M_x. Therefore, M_x takes as inputs values of $y_1, \ldots y_k, z_1, z_2$ (as these are the only universally quantified variables to the left of x in the quantifier prefix) and it always halts after computing a value for x. Now we run the Turing machine M_x with the inputs: c_i as the value for y_i for $i \in \{1, \ldots k\}$, d_1 as the value for z_1 and d_2 as the value for z_2. Let the value computed by M_x with these inputs be t. We then check if $t = d_1$. If yes, we conclude that $\mathfrak{M}_C \models \varphi$, else $\mathfrak{M}_C \not\models \varphi$. Thus, we have a decision procedure for $\mathcal{V}(\mathfrak{M})$-theory of $\mathfrak{M}_C$, i.e., $\mathcal{ED}(\mathfrak{M})$ is decidable. ◀

The construction in the first part of the above proof shows that there exists a Turing machine that runs in time polynomial in the length of ξ and in the length of a decision procedure for $\mathcal{ED}(\mathfrak{M})$, and generates a computable interpretation (i.e. code for a halting Turing machine) that computes $\mathfrak{M}$-interpretations of all Skolem functions in ξ. Further, since a positive answer to SkolemSynthesis implies a positive answer to SkolemExist, Theorem 10 also gives a sufficient condition for SkolemExist to have a positive answer.

7 Applications and complexity

We now look at some consequences of the above characterization. We first ask if we can algorithmically synthesize computable interpretations of Skolem functions in some well-known theories in first-order logic. We start with a lemma.

▶ **Lemma 11.** *Let $\mathfrak{M}$ be a computable $\mathcal{V}$-structure with universe $\mathcal{U}$. Suppose for every element $e \in \mathcal{U}$, there exists an effectively computable uni-variate $\mathcal{V}$-formula $\alpha_e(x)$ such that $\alpha_e(x)$ is* true *iff $x = e$. Then $Th(\mathfrak{M})$ is decidable iff $\mathcal{ED}(\mathfrak{M})$ is decidable.*

One may wonder if decidability of $Th(\mathfrak{M})$ automatically implies decidability of $\mathcal{ED}(\mathfrak{M})$. However, this is not true in general (see [2] for more details), emphasizing the need for Lemma 11. From Lemma 11 and Theorem 10 we now have,

▶ **Corollary 12.** *For the following theories, both SkolemExist and SkolemSynthesis have positive answers, and we can effectively synthesize computable $\mathfrak{M}$-interpretations for Skolem functions for a $\mathcal{V}$-formula: 1. Presburger arithmetic; 2. Linear rational arithmetic (LRA); 3. Theory of real algebraic numbers; 4. Theory of dense linear orders without endpoints.*

For the theory of natural numbers with addition, multiplication and order, we have seen in Proposition 8 that SkolemExist has a negative answer, which of course implies that SkolemSynthesis cannot have a positive answer. Using Lemma 11 and Theorem 10 we get a direct proof for the latter fact. To see this note that the premise of Lemma 11 holds for this theory as for Presburger arithmetic. Hence, $\mathcal{ED}(\mathfrak{M})$ is decidable iff $Th(\mathfrak{M})$ is decidable. But we know from the MRDP theorem [11] that the latter is indeed undecidable. Thus, from Theorem 10, we obtain that SkolemSynthesis has negative instances in this theory.

We remark that the above discussion can also be seen as an alternate proof of the fact that the elementary diagram is undecidable, since Theorem 10 is a characterization.

Complexity bounds on $\mathfrak{M}$-interpretations. When it applies, the proof of Theorem 10 gives us a construction of a Turing machine M that takes a formula ξ as input and outputs a computable $\mathfrak{M}$-interpretation (i.e. code for another Turing machine, say M') of Skolem functions for all existential variables in ξ. What bounds can we give on the worst case running time of M' (Problem 4.c in Section 4)? We start with a lower bound that follows from the second part of the proof of Theorem 10.

▶ **Theorem 13.** *Let $\mathfrak{M}$ be a computable $\mathcal{V}$-structure with a decidable $\mathcal{ED}(\mathfrak{M})$. The worst case running time of any computable $\mathfrak{M}$-interpretation of Skolem functions for a $\mathcal{V}$-formula is at least as much as that of a decision procedure for $\mathcal{ED}(\mathfrak{M})$.*

This shows for instance that for Presburger arithmetic, there exists formulas for which any computable $\mathfrak{M}$-interpretation of Skolem functions will take at least (alternating) double exponential time [8, 15] Next, for upper bounds, the computable $\mathfrak{M}$-interpretation of Skolem functions, as detailed in the proof of Theorem 10, relies on enumeration. Hence, it does not help in giving complexity upper bounds. However, if a theory admits effective constraint solving (see Sec. 2 for a definition), then we can do better.

▶ **Theorem 14.** *Let $\mathfrak{M}$ be a $\mathcal{V}$-structure such that $\mathcal{ED}(\mathfrak{M})$ is decidable. Suppose $\mathcal{ED}(\mathfrak{M})$ admits effective constraint solving with worst-case time complexity $T(n)$ and the solution is represented as a tuple of domain elements requiring at most $S(n)$ bits. Then we can synthesize $\mathfrak{M}$-interpretations of Skolem functions for $\mathcal{V}$-formulas of size n, such that the running time and output size of the $\mathfrak{M}$-interpretations are bounded by recursive functions of $T(n)$ and $S(n)$.*

As an example, if $S(n)$ is linear, i.e., $S(n) \leq C.n$ for a constant $C > 0$, then we get $time(n) \leq k.T(k.(\max\{C, 1\})^k.n)$ and $size(n) \leq k.(\max\{C, 1\})^k.n$. Finally, one way to obtain an algorithm for effective constraint solving is by using effective quantifier elimination repeatedly and then using quantifier-free constraint solving. Thus, we could further bound the complexity as functions of the complexity for effective quantifier elimination and that of quantifier-free constraint solving. This can be applied, for example, for LRA, theory of reals etc. Significantly, there are first-order theories that do not admit effective quantifier elimination but admit effective constraint solving, e.g., theory of evaluated trees [10]. In such cases, we can still use our approach to synthesize Skolem functions.

8 Expressing Skolem functions as terms

Whenever Skolem functions are computable, one can further ask: *Can Skolem functions be represented as terms?* Notice that in the Boolean setting, the notions of terms, functions and formulas are often conflated (as noted by Jiang [17] as well). Note that there are theories without any terms, for which Skolem functions can still be synthesized as halting Turing machines. For instance, the theory of (countable) dense linear order without endpoints does not admit any terms. Yet, from Corollary 12, we know that we can effectively synthesize computable Skolem functions for this theory. In fact, we can show a stronger result, viz, even when the theory admits terms, we may not be able to interpret a Skolem function as a term. To see this, consider the Presburger formula: $\forall y \forall z \exists x(((x = y) \vee (x = z)) \wedge ((x \geq y) \wedge (x \geq z)))$. The unique Skolem function for x is $\max(y, z)$, which can be written as an imperative program as: "**input(y,z); if** y $\geq$ z **then return** y **else return** z". This is a uniform representation (see Sec. 2) of a computable $\mathfrak{M}$-interpretation of the Skolem function for x. However, this function cannot be written as a term in Presburger arithmetic. Indeed, any term of y, z that uses only $+$, 0, 1 must be linear, while max is a non-linear function. Thus, we have

▶ **Proposition 15.** *There exist first order theories for which Skolem functions can be effectively computed, but they cannot be expressed as terms.*

As described in [17], if Skolem functions in a first order theory can be represented using a *finite set of conditional terms* (like in the case of $\max(y, z)$ above), the theory admits effective quantifier elimination. However, we already have first order theories, e.g. the theory of evaluated trees, that don't admit quantifier elimination, but admit effective synthesis of computable interpretations of Skolem functions. In such cases, Skolem functions can't be represented as a finite set of conditional terms either.

Note that there is a related notion of deskolemization in proof theory (see e.g., [6], [7]) in which proofs of Skolemized formulas are related to the proofs of corresponding formulas without Skolem functions. However, this does not necessarily yield computable interpretations of Skolem functions as terms.

9 Conclusion

The study of algorithmic computation of Skolem functions is highly nuanced. We explored what it means for Skolem functions for first order logic to be computable and synthesizable. Defining computable interpretations of Skolem functions as Turing machines, we showed that they may not always exist and checking if they exist is undecidable in general. However, when we fix a computable structure, we gave a precise characterization of when they exist and show several applications for specific theories. While we have made some preliminary progress regarding complexity issues, the question of synthesizing succinct interpretations is still open as is the question of when Skolem functions can be represented as terms in the logic. We hope that the theoretical framework set up here will lead to research towards implementable synthesis of Skolem functions for first order logic.

References

1 S. Akshay, J. Arora, S. Chakraborty, S. Krishna, D. Raghunathan, and S. Shah. Knowledge compilation for Boolean functional synthesis. In *Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA*, pages 161–169. IEEE, 2019.

2 S. Akshay and S. Chakraborty. On synthesizing Skolem functions for first order logic formulae. *CoRR*, abs/2102.07463, 2021. arXiv:2102.07463.

3 S. Akshay, S. Chakraborty, S. Goel, S. Kulal, and S. Shah. What's hard about Boolean functional synthesis? In *Computer Aided Verification – 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 251–269. Springer, 2018.

4 S. Akshay, S. Chakraborty, S. Goel, S. Kulal, and S. Shah. Boolean functional synthesis: hardness and practical algorithms. *Form Methods Syst Des.*, 57(1):53–86, 2021.

5 S. Akshay, S. Chakraborty, A. K. John, and S. Shah. Towards parallel boolean functional synthesis. In *TACAS 2017 Proceedings, Part I*, pages 337–353, 2017. doi:10.1007/978-3-662-54577-5_19.

6 J. Avigad. Eliminating definitions and Skolem functions in first-order logic. *ACM Trans. Comput. Log.*, 4(3):402–415, 2003. doi:10.1145/772062.772068.

7 M. Baaz, S. Hetzl, and D. Weller. On the complexity of proof deskolemization. *J. Symb. Log.*, 77(2):669–686, 2012. doi:10.2178/jsl/1333566645.

8 L. Berman. The complexity of logical theories. *Theoretical Computer Science*, 11(1):71–77, 1980. doi:10.1016/0304-3975(80)90037-7.

9 E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

10 T. Dao and K. Djelloul. Solving first-order constraints in the theory of the evaluated trees. In *Proceedings of the Constraint Solving and Contraint Logic Programming 11th Annual ERCIM International Conference on Recent Advances in Constraints*, CSCLP'06, pages 108–123. Springer-Verlag, 2006.

11 M. Davis, Y. Matijasevic, and J. Robinson. Hilbert's tenth problem. Diophantine equations: positive aspects of a negative solution. In *Proceedings of symposia in pure mathematics*, volume 28, pages 323–378, 1976.

12 K. Fazekas, M. J. H. Heule, M. Seidl, and A. Biere. Skolem function continuation for quantified Boolean formulas. In *International Conference on Tests and Proofs (TAP)*, volume 10375 of *Lecture Notes in Computer Science*, pages 129–138. Springer, 2017.

13 D. Fried, L. M. Tabajara, and M. Y. Vardi. BDD-based boolean functional synthesis. In *Computer Aided Verification – 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 402–421, 2016.

14 P. Golia, S. Roy, and K. S. Meel. Manthan: A data-driven approach for boolean function synthesis. In *Computer Aided Verification – 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 611–633. Springer, 2020.

15 C. Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.

16 M. Heule, M. Seidl, and A. Biere. Efficient Extraction of Skolem Functions from QRAT Proofs. In *Formal Methods in Computer-Aided Design , FMCAD 2014, Lausanne, Switzerland*, pages 107–114, 2014.

17 J.-H. R. Jiang. Quantifier elimination via functional composition. In *Proc. of CAV*, pages 383–397. Springer, 2009.

18 A. John, S. Shah, S. Chakraborty, A. Trivedi, and S. Akshay. Skolem functions for factored formulas. In *Formal Methods in Computer-Aided Design, FMCAD 2015*, pages 73–80, 2015.

19 V. Kuncak, M. Mayer, R. Piskac, and P. Suter. Complete functional synthesis. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 316–329. ACM, 2010.

20 M. Preiner, A. Niemetz, and A. Biere. Counterexample-guided model synthesis. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 264–280, 2017.

21 M. N. Rabe. Incremental determinization for quantifier elimination and functional synthesis. In *Computer Aided Verification – 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, pages 84–94, 2019.

22 M. N. Rabe and S. A. Seshia. Incremental determinization. In *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 375–392, 2016. doi:10.1007/978-3-319-40970-2_23.

23 M. N. Rabe, L. Tentrup, C. Rasmussen, and S. A. Seshia. Understanding and extending incremental determinization for 2QBF. In *Computer Aided Verification – 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, pages 256–274, 2018.

24 Preey Shah, Aman Bansal, S. Akshay, and Supratik Chakraborty. A normal form characterization for efficient boolean skolem function synthesis. 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.

25 A. Spielmann and V. Kuncak. Synthesis for unbounded bit-vector arithmetic. In *Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2012.

26 S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. *SIGPLAN Not.*, 45(1):313–326, 2010. doi:10.1145/1707801.1706337.

27 L. M. Tabajara and M. Y. Vardi. Factored boolean functional synthesis. In *Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 124–131, 2017.

28 S. Verma and S. Roy. Debug-localize-repair: a symbiotic construction for heap manipulations. *Formal Methods Syst. Des.*, 58(3):399–439, 2021. doi:10.1007/s10703-021-00387-z.

Sample Compression Schemes for Balls in Graphs

Jérémie Chalopin ✉
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France

Victor Chepoi ✉
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France

Fionn Mc Inerney ✉
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Sébastien Ratel ✉
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France

Yann Vaxès ✉
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France

Abstract

One of the open problems in machine learning is whether any set-family of VC-dimension d admits a sample compression scheme of size $O(d)$. In this paper, we study this problem for balls in graphs. For balls of arbitrary radius r, we design proper sample compression schemes of size 4 for interval graphs, of size 6 for trees of cycles, and of size 22 for cube-free median graphs. We also design approximate sample compression schemes of size 2 for balls of δ-hyperbolic graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Machine learning theory

Keywords and phrases Proper Sample Compression Schemes, Balls, Graphs, VC-dimension

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.31

Related Version *Full Version*: https://arxiv.org/abs/2206.13254

Funding This work was supported by the ANR project DISTANCIA (ANR-17-CE40-0015) and the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

1 Introduction

Sample compression schemes were introduced by Littlestone and Warmuth [22], and have been vastly studied in the literature due to their importance in computational machine learning. Roughly, a sample compression scheme consists of a compressor α and a reconstructor β, and the aim is to compress data as much as possible, such that data coherent with the original data can be reconstructed from the compressed data. For balls in graphs, sample compression schemes of size k can be defined as follows. Given a ball $B = B_r(x)$ of a graph $G = (V, E)$, a realizable sample for B is a signed subset $X = (X^+, X^-)$ of V such that X^+ is included in B, and X^- is disjoint from B. Given a realizable sample X, X is compressed to a subsample $\alpha(X) \subseteq X$ of size at most k. The reconstructor β takes $\alpha(X)$ as an input and returns $\beta(\alpha(X))$, a subset B' of vertices of G that is consistent with X, *i.e.*, X^+ is included in B', and X^- is disjoint from B'. If B' is always a ball of G, then the compression scheme is proper. If $X^+ = B$ and $X^- = V \setminus B$, then $\beta(\alpha(X))$ must coincide with B. Note that a proper sample compression scheme of size k for the family of all balls of G yields a sample compression scheme of size k for any subfamily of balls (*e.g.*, for balls of a fixed radius r), but this scheme is no longer proper. Sample compression schemes are labeled if β knows the labels of the elements of $\alpha(X)$, and are unlabeled otherwise (abbreviated LSCS and USCS, resp.). The Vapnik-Chervonenkis dimension (VC-dimension) of a set system was introduced by Vapnik and Chervonenkis [27] as a complexity measure of set systems. VC-dimension is central in PAC-learning, and is important in combinatorics and discrete geometry. Floyd and Warmuth [17] asked whether any set-family of VC-dimension d has a sample compression scheme of size $O(d)$. This remains one of the oldest open problems in machine learning.

In this paper, we consider the family of balls in graphs, which is as general as the sample compression conjecture. Indeed, the sample compression conjecture for set families in general is equivalent to the same conjecture restricted to the family of balls of radius 1 on split graphs in which samples only contain vertices in the clique, and the centers of the unit balls are in the stable set. Balls in graphs also constitute an important topic in graph theory, and moreover, their VC-dimension has often been considered in the literature (see, *e.g.*, [4, 7, 10, 16, 26]).

The VC-dimension of the balls of radius r of a graph not containing K_{n+1} as a minor is at most n [10]. This result was extended to arbitrary balls in [7]. Hence, the VC-dimension of balls of planar graphs is at most 4 (2 for trees and 3 for trees of cycles), and the VC-dimension of balls of a chordal graph G is at most its clique number $\omega(G)$. The VC-dimension of balls of interval graphs was shown to be at most 2 in [16]. Finally, the VC-dimension of balls of cube-free median graphs is unknown, but we can prove that it is at least 4.

Our results. In this paper, we design proper sample compression schemes of small size for the family of balls of a graph G. We investigate this problem for different graph classes. For trees of cycles, we exhibit proper LSCS of size 6 for all balls. Then, we design proper LSCS of size 22 for all balls of cube-free median graphs. We also construct proper LSCS of size 4 for all balls of interval graphs. Finally, we define (ρ, μ)-approximate proper sample compression schemes, and design $(2\delta, 3\delta)$-approximate LSCS of size 2 for δ-hyperbolic graphs.

Related work. Floyd and Warmuth [17] proved that, for any concept class of VC-dimension d, any LSCS has size at least $\frac{d}{5}$, and that, for some maximum classes of VC-dimension d, they have size at least d. Pálvölgyi and Tardos [25] proved that some concept classes of VC-dimension 2 do not admit USCS of size at most 2. On the positive side, it was shown by Moran and Yehudayoff [24] that LSCS of size $O(2^d)$ exist (their schemes are not proper). For particular concept classes, better results are known. Floyd and Warmuth [17] designed LSCS of size d for regions in arrangements of central hyperplanes in $\mathbb{R}^d$. Ben-David and Litman [5] obtained USCS of size d for regions in arrangements of affine hyperplanes in $\mathbb{R}^d$. Helmbold, Sloan, and Warmuth [19] (implicitly) constructed USCS of size d for intersection-closed concept classes. Moran and Warmuth [23] designed proper LSCS of size d for ample classes. Chalopin et al. [9] designed USCS of size d for maximum families. They also combinatorially characterized USCS for ample classes via the existence of *unique sink orientations* of their graphs. However, the existence of such orientations is open. Chepoi, Knauer, and Philibert [12] extended the result of [23], and designed proper LSCS of size d for concept classes defined by Complexes of Oriented Matroids (COMs). COMs were introduced in [3] as a natural common generalization of ample classes and Oriented Matroids [6].

2 Definitions

Concept classes and sample compression schemes. Let V be a non-empty finite set. Let $\mathcal{C} \subseteq 2^V$ be a family of subsets (also called a *concept class*) of V. The *VC-dimension* VC-dim$(\mathcal{C})$ of $\mathcal{C}$ is the size of a largest set $Y \subseteq V$ *shattered* by $\mathcal{C}$, *i.e.*, such that $\{C \cap Y : C \in \mathcal{C}\} = 2^Y$. In machine learning, a *(labeled) sample* is a set $X = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, where $x_i \in V$ and $y_i \in \{-1, +1\}$. To X is associated the unlabeled sample $\underline{X} = \{x_1, \ldots, x_m\}$. A sample X is *realizable by a concept* C if $y_i = +1$ if $x_i \in C$, and $y_i = -1$ if $x_i \notin C$. A sample X is *realizable by a concept class* $\mathcal{C}$ if X is realizable by some $C \in \mathcal{C}$.

We adopt the language of sign maps and sign vectors from [6]. Let $\mathcal{L}$ be a *set of sign vectors*, *i.e.*, maps from V to $\{\pm1, 0\} := \{-1, 0, +1\}$. The elements of $\mathcal{L}$ are also called *covectors*. For $X \in \mathcal{L}$, let $X^+ := \{v \in V : X_v = +1\}$ and $X^- := \{v \in V : X_v = -1\}$. $\underline{X} = X^- \cup X^+$ is called the *support* of X, and its complement $X^0 := V \setminus \underline{X} = \{v \in V : X_v = 0\}$ the *zero set* of X. Since $X^0 = V \setminus (X^- \cup X^+)$, we will view any sample X as $X^- \cup X^+$. Let $\preceq$ be the product ordering on $\{\pm1, 0\}^V$ relative to the ordering of signs with $0 \preceq -1$ and $0 \preceq +1$. Any concept class $\mathcal{C} \subseteq 2^V$ can be viewed as a set of sign vectors of $\{\pm1\}^V$: for any $C \in \mathcal{C}$ we consider the sign vector $X(C)$, where $X_v(C) = +1$ if $v \in C$ and $X_v(C) = -1$ if $v \notin C$. For simplicity, we will consider $\mathcal{C}$ as a family of sets and as a set of $\{\pm1\}$-vectors. We now define sample compression schemes. This way of presenting them seems novel. From the definition, it follows that a sample X is just a $\{\pm1, 0\}$-sign vector. Given a concept class $\mathcal{C} \subseteq 2^V$ and $C \in \mathcal{C}$, the set of samples realizable by C consists of all covectors $X \in \{\pm1, 0\}^V$ such that $X \preceq C$. We denote by $\downarrow \mathcal{C}$ the set of all samples realizable by $\mathcal{C}$.

A *proper labeled sample compression scheme* (*proper LSCS*) of size k for a concept class $\mathcal{C} \subseteq \{\pm1\}^V$ is defined by a *compressor* $\alpha : \{\pm1, 0\}^V \to \{\pm1, 0\}^V$ and a *reconstructor* $\beta : \{\pm1, 0\}^V \to \mathcal{C}$ such that, for any realizable sample $X \in \downarrow \mathcal{C}$, $\alpha(X) \preceq X \preceq \beta(\alpha(X))$ and $|\underline{\alpha}(X)| \leq k$, where $\preceq$ is the order between sign vectors defined above, and $\underline{\alpha}(X)$ is the support of the subsample of the sign vector X. Hence, $\alpha(X)$ is a signed vector with a support of size at most k such that $\alpha(X) \preceq X$, and $\beta(\alpha(X))$ is a concept C of $\mathcal{C}$ viewed as a sign vector. It suffices to define the map α only on $\downarrow \mathcal{C}$, and the map β only on $\mathrm{Im}(\alpha) := \alpha(\downarrow \mathcal{C})$. The condition $X \preceq \beta(\alpha(X))$ is equivalent to the condition $\beta(\alpha(X))|\underline{X} = X$, which means that the restriction of the concept $\beta(\alpha(X))$ to the support of X coincides with the sign vector X. *Proper unlabeled sample compression schemes* (*proper USCS*) are defined analogously, only that $\alpha(X)$ is not a signed vector, but a subset of size at most k of the support of X. For graphs, any preprocessing on the input graph G, such as a labeling or an embedding of G, is permitted and known to both the compressor and the reconstructor. As in, *e.g.*, [22, 24], information, like representing the support as a vector with coordinates, is also permitted, and when we use such information, we refer to α and β as vectors rather than maps. Lastly, in our schemes, the reconstructor returns the empty set when $X^+ = \varnothing$, and thus, one may consider that our schemes are not proper. We note that in all of the LSCS for the family of balls of arbitrary radius we exhibit in this paper, we could simply choose an ordering on the vertices of the graph $G = (V, E)$, and put into $\alpha(X)$ a single vertex $z \in X^-$ such that its successor z' in the ordering does not belong to X^-. Then, the reconstructor returns a ball $B_0(z')$ that does not intersect $X = X^-$ by the choice of z. However, to avoid additional complications for such degenerate cases, we make use of the empty set.

Graphs. Every graph $G = (V, E)$ in this paper is simple and connected. The *distance* $d(u, v) := d_G(u, v)$ between two vertices u and v of a graph G is the length of a (u, v)-shortest path. The *interval* $I(u, v)$ is the set of vertices contained in (u, v)-shortest paths. A set S is *gated* if, for any vertex $x \in V$, there is a vertex $x' \in S$ (the *gate* of x, with $x' = x$ if $x \in S$) such that $x' \in I(x, y)$ for any $y \in S$. A *median* of a triplet u, v, w is any vertex in $I(u, v) \cap I(v, w) \cap I(w, u)$. A graph G is *median* [1] if any triplet of vertices u, v, w has a unique median. For any vertex $x \in V$ and any integer $r \geq 0$, the *ball of radius r centered at x* is the set $B_r(x) := \{v \in V : d(v, x) \leq r\}$. The unit ball $B_1(x)$ is usually denoted by $N[x]$ and called the *closed neighborhood of x*. The *sphere* of radius r centered at x is the set $S_r(x) = \{z \in V : d(z, x) = r\}$. Let also $\mathrm{cB}_r(u) = V \setminus B_r(u)$. Two balls $B_{r_1}(x)$ and $B_{r_2}(y)$ are *distinct* if $B_{r_1}(x)$ and $B_{r_2}(y)$ are distinct as sets. We denote by $\mathcal{B}(G)$ the set of all distinct balls of G, and by $\mathcal{B}_r(G)$ the set of all distinct balls of radius r of G. For a subset $Y \subseteq V$, we call $\mathrm{diam}(Y) = \max\{d(u, v) : u, v \in Y\}$ the *diameter* of Y, and we call any pair $u, v \in Y$ such that $d(u, v) = \mathrm{diam}(Y)$ a *diametral pair* of Y.

3 Trees of cycles

A *tree of cycles* (or *cactus*) is a graph in which each *block* (2-connected component) is a cycle or an edge. We can design proper labeled (unlabeled, resp.) sample compression schemes of size 2 for balls of (metric, resp.) trees, and balls of radius r of trees [8]. Indeed, for balls in metric trees, $\alpha(X)$ is generally a diametral pair u, v of X^+ and we return the ball of radius $d(u, v)/2$ centered at the middle point of the (u, v)-shortest path. This does not work for balls of radius r in trees, for which we cleverly encode a center.

We now describe the main result of this section: a proper labeled sample compression scheme of size 6 for balls of trees of cycles. Let G be a tree of cycles. For a vertex v of G that is not a cut vertex, let $C(v)$ be the unique cycle containing v. If v is a cut vertex or a degree-one vertex, then set $C(v) = \{v\}$. Let $T(G)$ be the tree whose vertices are the cut vertices and the blocks of G, and where a cut vertex v is adjacent to a block B of G if and only if $v \in B$. For any two vertices u, v of G, let $C(u, v)$ denote the union of all cycles and/or edges on the unique path of $T(G)$ between $C(u)$ and $C(v)$. Note that $C(u, v)$ is a path of cycles, and that $C(u, v)$ is gated. Let X be a realizable sample for $\mathcal{B}(G)$, and $\{u^+, v^+\}$ a diametral pair of X^+. The next lemma shows that the center of a ball realizing X can always be found in $C(u^+, v^+)$.

▶ **Lemma 1.** *Let $B_r(x)$ be a ball realizing X, x' be the gate of x in $C(u^+, v^+)$, and $r' = r - d(x, x')$. Then, the ball $B_{r'}(x')$ also realizes X.*

In what follows, let $B_r(x)$ be a ball realizing X with x in $C(u^+, v^+)$ (it exists by Lemma 1). Let C be a cycle of $C(u^+, v^+)$ containing x. The main idea is to encode a region of $C(u^+, v^+)$ where the center x of $B_r(x)$ is located (this region may be C), the center, and the radius of $B_r(x)$ by a few vertices of X. The diametral pair $\{u^+, v^+\}$ is in $\alpha(X)$. If X contains a vertex $w \neq u^+, v^+$ whose gate in $C(u^+, v^+)$ is in C, then C is easily detected by including w in $\alpha(X)$. In this case, it remains to find the position of x in C and to compute the radius r. This is done by using 2 or 3 vertices of X. Otherwise, if the gates in $C(u^+, v^+)$ of all vertices $w \in X \setminus \{u^+, v^+\}$ are outside C, then we show that $B_r(x)$ is determined by 4 vertices in X.

The partitioning of X. For a vertex $y \in C(u^+, v^+)$, set $r_y := \max\{d(y, u^+), d(y, v^+)\}$ and $r^*_y := \max\{d(y, w) : w \in X^+\}$. Clearly, $B_{r^*_y}(y)$ is the smallest ball centered at y containing X^+. For any vertex z of G, we denote by z' its gate in $C(u^+, v^+)$. Let u^* and v^* be the gates of u^+ and v^+ in C. We partition X and X^- as follows. Let X_u (X^-_u, resp.) consist of all $w \in X$ ($w \in X^-$, resp.) whose gate w' in $C(u^+, v^+)$ belongs to $C(u^+, u^*)$. The sets X_v and X^-_v are defined analogously. Let X_C (X^-_C, resp.) consist of all the vertices $w \in X$ ($w \in X^-$, resp.) whose gates w' in $C(u^+, v^+)$ belong to the cycle C. Note that some of these sets can be empty and that $X^-_u \subseteq X_u$, $X^-_v \subseteq X_v$, and $X^-_C \subseteq X_C$. Let u_0 be the cut vertex of $C(u^+, v^+)$ farthest from u^*, and such that, for any vertex $w \in X_u$, its gate $w' \in C(u^+, v^+)$ is not in $C(u_0, u^*)$. Analogously, we define the cut vertex v_0 with respect to v^* and X_v. If $u^* = u^+$ ($v^* = v^+$, resp.), then set $u_0 = u^* = u^+$ ($v_0 = v^* = v^+$, resp.).

First, suppose that $X_C = \varnothing$. Let w_1 be a vertex of X_u closest to u_0, and z_1 a vertex of X^-_u closest to x. Note that w_1 always exists as u^+ is in X_u, and that z_1 exists if and only if X^-_u is non-empty. Similarly, we define the vertices w_2 and z_2 with respect to X_v and X^-_v. See Fig. 1 for an illustration. The next lemmas show how to compute $B_r(x)$ in this case.

▶ **Lemma 2.** *For $y \in C(u_0, v_0)$, if there exists a vertex $w \in X^+ \setminus B_{r_y}(y)$, then $w' \in C(y)$. Consequently, if $X_C = \varnothing$, then, for any $y \in C(u_0, v_0)$, we have $X^+ \subset B_{r_y}(y)$ and $r_y = r^*_y$.*

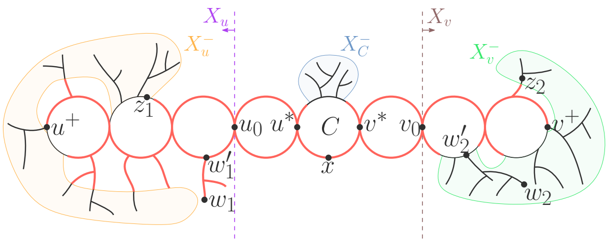


Figure 1 The vertices and sets used in the proper labeled sample compression scheme for trees of cycles. The ball $B_r(x)$ is represented in red. The cycles outside $C(u^+, v^+)$ are represented as paths.

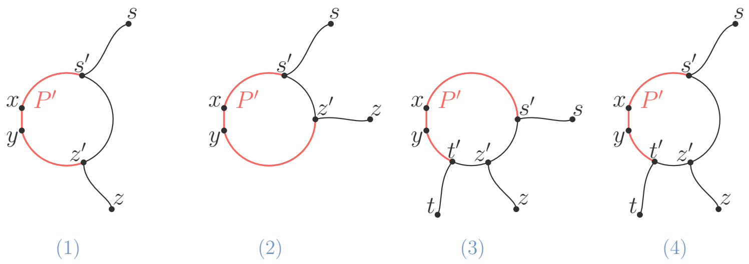
▶ **Lemma 3.** *If $X^- \neq \varnothing$ and $X_C = \varnothing$, then $B_{r_y^*}(y) \cap X^- = \varnothing$ for any vertex $y \in C(u_0, v_0)$ such that $B_{r_y^*}(y) \cap \{z_1, z_2\} = \varnothing$.*

Now, suppose that $X_C \neq \varnothing$. By the definition of r_x^*, $B_{r_x^*}(x)$ also realizes X. Let w be a vertex of X whose gate w' in $C(u^+, v^+)$ is in C. If, for every $y \in C$, $B_{r_y^*}(y)$ realizes X, then $B_{r_{w'}^*}(w')$ realizes X, and, in this case, let $s \in X^+$ be such that $d(w', s) = r_{w'}^*$. Otherwise, we can find two adjacent vertices x and y of C such that $B_{r_x^*}(x)$ realizes X, but $B_{r_y^*}(y)$ does not. This implies that there is a vertex $z \in X^-$ with $z \in B_{r_y^*}(y) \setminus B_{r_x^*}(x)$. In this case, let $s, t \in X^+$ be such that $r_y^* = d(y, s)$ and $r_x^* = d(x, t)$, with $t = s$ whenever $r_y^* = r_x^* + 1$. Let s', t', and z' be the respective gates of s, t, and z in C. If $s = t$ ($s \neq t$, resp.), then let P' be the path of C between s' and z' (t', resp.) containing the edge xy. See Fig. 2 for an illustration.

▶ **Lemma 4.** *For adjacent vertices $x, y \in C$, and the corresponding vertices $z \in X^-$ and $s \in X^+$, one of the following conditions holds:*
(1) *$r_y^* = r_x^* + 1$, $d(x, z) = d(y, z) + 1$, and $d(x, s) = d(y, s) - 1$;*
(2) *$r_y^* = r_x^* + 1$, $d(x, z) = d(y, z)$, and $d(x, s) = d(y, s) - 1$;*
(3) *$r_y^* = r_x^*$, $d(x, z) = d(y, z) + 1$, and $d(x, s) = d(y, s)$;*
(4) *$r_y^* = r_x^*$, $d(x, z) = d(y, z) + 1$, and $d(x, s) = d(y, s) - 1$.*

Without the knowledge of r_x^* and r_y^*, the relationships between $d(x, z)$ and $d(y, z)$, and between $d(x, s)$ and $d(y, s)$ do not allow us to distinguish between the cases (1) and (4). This can be done by additionally using the vertex $t \in X^+$ defined above. Indeed, in Case (1) we have $t = s$, while in Case (4) we have $t \neq s$ and $d(x, t) = d(y, t) + 1$. We continue with the following simple lemma for paths (where, for each edge xy in the path, x is to the left of y):

▶ **Lemma 5.** *Let Q be a graph which is a path with end-vertices $a \neq b$, and let d' be its distance function. Then, Q contains a unique edge $x_0 y_0$ such that $d'(x_0, b) - d'(x_0, a) \in \{1, 2\}$.*



(1) (2) (3) (4)

Figure 2 Definition and positioning of s, t, and z in the four cases of Lemma 4.

We use Lemma 5 to find adjacent vertices x_0 and y_0 of C and an integer r_x^* that satisfy a condition of Lemma 4. Let P' be the path between z' and s' (or between s' and t') containing the edge xy as defined above. Let P be the path of G obtained by joining the shortest (s', s)- and (z, z')-paths ((s, s')- and (t', t)-paths, resp.) to P'. Let d' be the distance function on P.

▶ **Lemma 6.** *Let P be the (s,z)-path or (s,t)-path of G defined above. Let x_0y_0 be the unique edge of P satisfying the conclusion of Lemma 5. Then, $x_0 = x$ and $y_0 = y$. Moreover,*

(1) *if P is an (s,z)-path, $d'(x_0,z) = d(x_0,z)$, and $d'(y_0,s) = d(y_0,s)$, then $r^*_x = d(y_0,s) - 1$;*

(2) *if P is an (s,z)-path, $d'(x_0,z) = d(x_0,z) + 1$, and $d'(y_0,s) = d(y_0,s)$, then $r^*_x = d(y_0,s) - 1$;*

(3) *if P is an (s,z)-path, $d'(x_0,z) = d(x_0,z)$, and $d'(y_0,s) = d(y_0,s) + 1$, then $r^*_x = d(y_0,s)$;*

(4) *if P is an (s,t)-path, then $r^*_x = d(x_0,t)$.*

The compressor $\alpha(X)$. The compressor $\alpha(X)$ is a vector with six coordinates, which are grouped into three pairs: $\alpha(X) := (\alpha_1(X), \alpha_2(X), \alpha_3(X))$. The pair $\alpha_1(X) \subseteq X^+$ is a diametral pair (u^+, v^+) of X^+, $\alpha_2(X)$ is used to specify the region of $C(u^+, v^+)$ where the center of the target ball is located, and the pair $\alpha_3(X)$ is used to compute the radius of this ball. We use the symbol $*$ to indicate that the respective coordinate of $\alpha(X)$ is empty.

We continue with the definitions of $\alpha_2(X)$ and $\alpha_3(X)$. First, suppose that $X_C = \varnothing$, *i.e.*, $X_u \cup X_v = X$ and $X^-_u \cup X^-_v = X^-$. Then, set $\alpha_2(X) := (w_1, w_2)$ and $\alpha_3(X) := (z_1, z_2)$. Now, suppose that $X_C \neq \varnothing$. Let w be a vertex of X whose gate w' in $C(u^+, v^+)$ belongs to C. If $B_{r^*_x}(x)$ realizes X for any vertex x of C, then set $\alpha_2(X) := (w, *)$ and $\alpha_3(X) := (s, *)$, where $s \in X^+$ is such that $d(w', s) = r^*_{w'}$. Otherwise, we pick an edge xy of C such that $B_{r^*_x}(x)$ realizes X and $B_{r^*_y}(y)$ does not realize X. Let s', t', and z' be the respective gates in C of the vertices s, t, and z as defined previously. If $s = t$, then the path P is defined by the vertices s and z, and set $\alpha_3(X) := (s, z)$. Otherwise, the path P is defined by the vertices s and t, and set $\alpha_3(X) := (s, t)$. Moreover, set $\alpha_2(X) := (*, w)$ if the edge xy belongs to the path from s' to z' (from s' to t', resp.) in the clockwise traversal of C, and $\alpha_2(X) := (w, *)$ otherwise. Formally, the compressor function α is defined in the following way:

(C1) if $X^- = \varnothing$, set $\alpha_1(X) = \alpha_2(X) = \alpha_3(X) := (*, *)$;

(C2) otherwise, if $|X^+| = 0$, set $\alpha_1(X) = \alpha_2(X) := (*, *)$ and $\alpha_3(X) := (z, *)$, where z is an arbitrary vertex of X^-;

(C3) otherwise, if $X^+ = \{u\}$, set $\alpha_1(X) := (u, *)$, $\alpha_2(X) := (*, *)$, and $\alpha_3(X) := (z, *)$, where z is an arbitrary vertex of X^-;

(C4) otherwise, if $|X^+| \geq 2$ and $X_C = \varnothing$, set $\alpha_1(X) := (u^+, v^+)$, $\alpha_2(X) := (w_1, w_2)$, and

- **(C4i)** if the vertex z_2 does not exist, then set $\alpha_3(X) := (z_1, *)$;
- **(C4ii)** if the vertex z_1 does not exist, then set $\alpha_3(X) := (*, z_2)$;
- **(C4iii)** if the vertices z_1 and z_2 exist, set $\alpha_3(X) := (z_1, z_2)$;

(C5) otherwise ($|X^+| \geq 2$ and $X_C \neq \varnothing$), and

- **(C5i)** if, for any vertex $y \in C$, the ball $B_{r^*_y}(y)$ realizes X, then set $\alpha_1(X) := (u^+, v^+)$, $\alpha_2(X) := (w, *)$, and $\alpha_3(X) := (s, *)$, where $s \in X^+$ is such that $d(w', s) = r^*_{w'}$;
- **(C5ii)** otherwise, if s and z are given, and the edge xy belongs to the clockwise (s', z')-path of C, then set $\alpha_2(X) := (*, w)$ and $\alpha_3(X) := (s, z)$;
- **(C5iii)** otherwise, if s and z are given, and the edge xy belongs to the counterclockwise (s', z')-path of C, then set $\alpha_2(X) := (w, *)$ and $\alpha_3(X) := (s, z)$;
- **(C5iv)** otherwise, if s and t are given, and the edge xy belongs to the clockwise (s', t')-path of C, then set $\alpha_2(X) := (*, w)$ and $\alpha_3(X) := (s, t)$;
- **(C5v)** otherwise, if s and t are given, and the edge xy belongs to the counterclockwise (s', t')-path of C, then set $\alpha_2(X) := (w, *)$ and $\alpha_3(X) := (s, t)$.

The reconstructor $\beta(X)$. Let Y be a vector on six coordinates grouped into three pairs Y_1, Y_2, and Y_3. If $Y_1 = (y_1, y_2)$, then, for any vertex t of G, we denote by t' its gate in $C(y_1, y_2)$. For any vertex y of $C(y_1, y_2)$, we also set $r_y := \max\{d(y, y_1), d(y, y_2)\}$. The reconstructor β takes Y and returns a ball $B_r(y)$ of G defined in the following way:

(R1) if $Y = ((*, *), (*, *), (*, *))$, then $\beta(Y)$ is any ball that contains the vertex set of G;

(R2) if $Y = ((*, *), (*, *), (y_5, *))$, then $\beta(Y)$ is the empty set;

(R3) if $Y = ((y_1, *), (*, *), (y_5, *))$, then $\beta(Y)$ is the ball $B_0(y_1)$;

(R4) if $Y_1 = (y_1, y_2)$ and $Y_2 = (y_3, y_4)$, then let u_0 be the cut vertex of $C(y'_3)$ between y'_3 and y_2, and v_0 be the cut vertex of $C(y'_4)$ between y'_4 and y_1. Then, $\beta(Y)$ is any ball $B_{r_y}(y)$ centered at $y \in C(u_0, v_0)$ such that $B_{r_y}(y)$ contains no vertex of Y_3.

(R5i) if $Y = ((y_1, y_2), (y_3, *), (y_5, *))$, then $\beta(Y)$ is the ball $B_r(y'_3)$ of radius $r = d(y'_3, y_5)$;

(R5ii) if $Y = ((y_1, y_2), (*, y_4), (y_5, y_6))$ and $(y_5, y_6) \in X^+ \times X^-$, let xy be the edge of the (y'_5, y'_6)-path in the clockwise traversal of the cycle $C(y'_4)$ such that $|d'(x, y_6) - d'(x, y_5)| \in \{1, 2\}$ and y is closer to y_6 than x is. Let $\beta(Y)$ be the ball $B_r(x)$, where $r = d(y, y_5)$ if $d'(y, y_5) = d(y, y_5) + 1$, and $r = d(y, y_5) - 1$ otherwise;

(R5iii) if $Y = ((y_1, y_2), (y_3, *), (y_5, y_6))$ and $(y_5, y_6) \in X^+ \times X^-$, let xy be the edge of the (y'_5, y'_6)-path in the counterclockwise traversal of $C(y'_3)$ such that $|d'(x, y_6) - d'(x, y_5)| \in \{1, 2\}$ and y is closer to y_6 than x is. Let $\beta(Y)$ be the ball $B_r(x)$, where $r = d(y, y_5)$ if $d'(y, y_5) = d(y, y_5) + 1$, and $r = d(y, y_5) - 1$ otherwise;

(R5iv) if $Y = ((y_1, y_2), (*, y_4), (y_5, y_6))$ and $(y_5, y_6) \in X^+ \times X^+$, let xy be the edge of the (y'_5, y'_6)-path in the clockwise traversal of the cycle $C(y'_4)$ such that $|d'(x, y_6) - d'(x, y_5)| \in \{1, 2\}$ and y is closer to y_6 than x is. Let $\beta(Y)$ be the ball $B_r(x)$, where $r = d(x, y_6)$;

(R5v) if $Y = ((y_1, y_2), (y_3, *), (y_5, y_6))$ and $(y_5, y_6) \in X^+ \times X^+$, let xy be the edge of the (y'_5, y'_6)-path in the counterclockwise traversal of the cycle $C(y'_3)$ such that $|d'(x, y_6) - d'(x, y_5)| \in \{1, 2\}$ and y is closer to y_6 than x is. Let $\beta(Y)$ be the ball $B_r(x)$, where $r = d(x, y_6)$;

▶ **Proposition 7.** *For any tree of cycles G, the pair (α, β) of vectors defines a proper labeled sample compression scheme of size 6 for $\mathcal{B}(G)$.*

Proof. Let X be a realizable sample for $\mathcal{B}$, $Y = \alpha(X)$, and $B_r(x^*) = \beta(Y)$. We prove case by case that the ball $B_r(x^*)$ realizes the sample X. One can easily see that the cases (Rk) and their subcases in the definition of β correspond to the cases (Ck) and their subcases in the definition of α: namely, the vector Y in Case (Rk) has the same specified coordinates as the vector $\alpha(X)$ in Case (Ck). We consider only the Case (R4), the other cases being similar. Then, $Y_1 = (y_1, y_2)$ and $Y_2 = (y_3, y_4)$. Since $Y = \alpha(X)$, the sample X satisfies the conditions of Case (C4), *i.e.*, $|X^+| \geq 2$ and $X_C = \varnothing$. Therefore, $Y_1 = (y_1, y_2) = (u^+, v^+) = \alpha_1(X), Y_2 = (y_3, y_4) = (w_1, w_2) = \alpha_2(X)$, and Y_3 (containing one or two vertices) coincides with $\alpha_3(X)$ (containing one or two vertices z_1, z_2 as in subcases (C4i)–(C4iii)). The ball $B_{r_y}(y)$ returned by Case (R4) is centered at $y \in C(u_0, v_0)$, contains Y_1, and is disjoint from Y_3. Since the target ball $B_r(x)$ has its center on $C(u_0, v_0)$ and is compatible with $X \supseteq Y_1 \cup Y_3$, the ball $B_{r_y}(y)$ is well-defined. By Lemma 2, $B_{r_y}(y)$ contains X^+. By Lemma 3, $B_{r_y}(y)$ is disjoint from X^-. Thus, $B_{r_y}(y)$ is compatible with X. ◀

▶ Remark 8. The most technically involved case of the previous result is the case $X_C \neq \varnothing$. In fact, this case corresponds to proper labeled sample compression schemes in *spiders*, *i.e.*, in graphs consisting of a single cycle C and paths of different lengths emanating from this cycle. Due to this case, $\alpha(X)$ in our result is not a signed map but a signed vector of size 6. Thus, in this case, we need extra information compared to the initial definition of proper labeled sample compression schemes. The VC-dimension of the family of balls in a spider and in a tree of cycles is 3. We wonder *whether the family of balls in spiders admits a proper labeled sample compression scheme without any information that is of (a) size* 3 *or (b) constant size.*

4 Cube-free median graphs

The *dimension* $\dim(G)$ of a median graph G is the largest dimension of a hypercube of G. A *cube-free median graph* is a median graph of dimension 2, *i.e.*, a median graph not containing 3-cubes as isometric subgraphs. For references about median graphs, see [1]. For cube-free median graphs, see [2, 11, 13, 14]. We use the fact that intervals of median graphs are gated. We describe a proper LSCS of size 22 for balls of cube-free median graphs.

Let G be a cube-free median graph. Let X be a realizable sample for $\mathcal{B}(G)$, and $\{u^+, v^+\}$ a diametral pair of X^+. The next lemma shows that the center of a ball realizing X can always be found in $I(u^+, v^+)$ (this result does not hold for all median graphs):

▶ **Lemma 9.** *If x' is the gate of x in the interval $I(u^+, v^+)$, and $r' = r - d(x, x')$, then X is a realizable sample for $B_{r'}(x')$, i.e., $X^+ \subseteq B_{r'}(x')$ and $X^- \cap B_{r'}(x') = \varnothing$.*

By [14], $I(u^+, v^+)$ of a cube-free median graph has an isometric embedding in the square grid $\mathbb{Z}^2$. We denote by (z_a, z_b) the coordinates in $\mathbb{Z}^2$ of a vertex $z \in I(u, v)$. We consider isometric embeddings of $I(u, v)$ in $\mathbb{Z}^2$ for which $u = (0, 0)$ and $v = (v_a, v_b)$ with $v_a \geq 0$ and $v_b \geq 0$. We fix a canonical isometric embedding, which can be used both by the compressor and the reconstructor. Finally, we use the same notation for the vertices and their images under the embedding, and we denote by $\mathbf{I}$ the interval $I(u^+, v^+)$ embedded in $\mathbb{Z}^2$. As usual, for a vertex $z \in V$, we denote by z' its gate in the interval $I(u^+, v^+)$.

The compressor $\alpha(X)$. The compressor $\alpha(X)$ is a vector with 22 coordinates grouped into four parts $\alpha(X) := (\alpha_1(X), \alpha_2(X), \alpha_3(X), \alpha_4(X))$. The part $\alpha_1(X) \subseteq X^+$ consists of a diametral pair (u^+, v^+) of X^+. The part $\alpha_2(X) \subseteq X$ has size 4, and is used to specify a region $\mathbf{R} \subseteq \mathbf{I} = I(u^+, v^+)$ such that the gates in $I(u^+, v^+)$ of all the vertices of X are located outside or on the boundary of $\mathbf{R}$. Moreover, $\mathbf{R}$ contains the center x of the target ball $B_r(x)$. The parts $\alpha_3(X) \subseteq X^+$ and $\alpha_4(X) \subseteq X^-$ each have size 8 and are used to locate the center and the radius of a ball $B_{r''}(y)$ realizing X. Now, we formally define $\alpha_i(X)$, $i = 1, ..., 4$. Let $X_1 := \{w \in X : w'_b \geq x_b\}$, $X_2 := \{w \in X : w'_a \geq x_a\}$, $X_3 := \{w \in X : w'_b \leq x_b\}$, and $X_4 := \{w \in X : w'_a \leq x_a\}$. Since $I(u^+, v^+)$ is gated, $X = \cup_{i=1}^4 X_i$. Denote by X'_i, $i = 1, ..., 4$, the gates of the vertices of X_i in $I(u^+, v^+)$. Set $\alpha_2(X) := (w_1, w_2, w_3, w_4) \in X^4$, where:

- w_1 is a vertex of X_1 whose gate w'_1 has the smallest ordinate among the vertices of X'_1;
- w_2 is a vertex of X_2 whose gate w'_2 has the smallest abscissa among the vertices of X'_2;
- w_3 is a vertex of X_3 whose gate w'_3 has the largest ordinate among the vertices of X'_3;
- w_4 is a vertex of X_4 whose gate w'_4 has the largest abscissa among the vertices of X'_4;

For a vertex $w = (w_a, w_b) \in \mathbb{Z}^2$, we consider the four coordinate halfplanes $\mathbf{H}_{\leq w_a} := \{t : t_a \leq w_a\}$, $\mathbf{H}_{\geq w_a}$,$\mathbf{H}_{\leq w_b}$, and $\mathbf{H}_{\geq w_b}$. Let $\mathbf{R}$ be the set of vertices of $\mathbf{I}$ that belong to the intersection of the halfplanes $\mathbf{H}_1 := \mathbf{H}_{\leq w_{1b}}$, $\mathbf{H}_2 := \mathbf{H}_{\leq w_{2a}}$, $\mathbf{H}_3 := \mathbf{H}_{\geq w_{3b}}$, and $\mathbf{H}_4 := \mathbf{H}_{\geq w_{4a}}$. If a vertex w_i does not exist, then the corresponding halfplane $\mathbf{H}_i$ is not defined. From the definition, the inside of $\mathbf{R}$ does not contain gates of vertices of X. We denote by $\mathbf{S}_i$, $i = 1, ..., 4$, the intersection of $\mathbf{I}$ with the closure of the complementary halfspace of $\mathbf{H}_i$. We call $\mathbf{S}_i$, $i = 1, ...4$, a *strip of* $\mathbf{I}$. Consequently, the interval $\mathbf{I}$ is covered by the region $\mathbf{R}$, two *horizontal* strips $\mathbf{S}_1$ and $\mathbf{S}_3$, and two *vertical* strips $\mathbf{S}_2$ and $\mathbf{S}_4$. Using this notation, we can redefine X_i as the

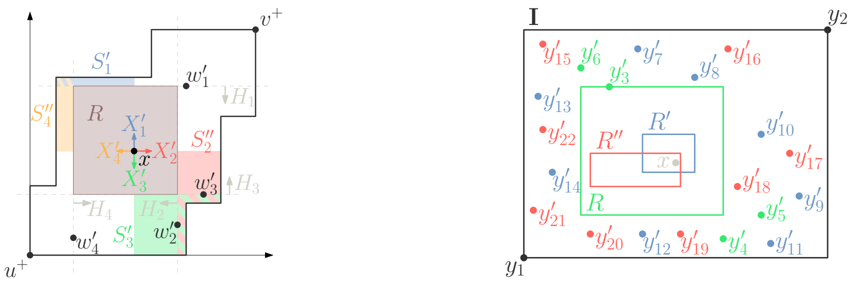


Figure 3 On the left, the region $\mathbf{R}$ and the halfstrips $\mathbf{S}'_1(x)$, $\mathbf{S}''_2(x)$, $\mathbf{S}'_3(x)$, and $\mathbf{S}''_4(x)$. On the right, the regions $\mathbf{R}$, $\mathbf{R}'$, and $\mathbf{R}''$ computed from $\alpha(X)$. Steps 1-4 of the reconstruction correspond to the black, green, blue, and red parts of the figure. The target center x is given in gray.

sets of all the vertices of X whose gate in $\mathbf{I}$ belongs to the strip $\mathbf{S}_i$. Consequently, $X'_i \subseteq \mathbf{S}_i$. Furthermore, for a vertex $z \in \mathbb{Z}^2$, each strip $\mathbf{S}_i$ is partitioned into two strips $\mathbf{S}'_i(z)$ and $\mathbf{S}''_i(z)$ by the vertical or horizontal line passing via z. The labeling of the strips is done in the clockwise order around z, see Fig. 3 (left). Let $\alpha_3(X) := (s_1, t_1, s_2, t_2, s_3, t_3, s_4, t_4)$, where

- s_1 is a vertex of X^+ furthest from x, whose gate s'_1 belongs to $\mathbf{S}'_1(x)$, and t_1 is a vertex of X^+ such that its gate t'_1 belongs to $\mathbf{S}''_1(x)$ and the abscissa of t'_1 is closest to x_a;
- s_2 is a vertex of X^+ furthest from x, whose gate s'_2 belongs to $\mathbf{S}''_2(x)$, and t_2 is a vertex of X^+ such that its gate t'_2 belongs to $\mathbf{S}'_2(x)$ and the ordinate of t'_2 is closest to x_b;
- s_3 is a vertex of X^+ furthest from x, whose gate s'_3 belongs to $\mathbf{S}'_3(x)$, and t_3 is a vertex of X^+ such that its gate t'_3 belongs to $\mathbf{S}''_3(x)$ and the abscissa of t'_3 is closest to x_a;
- s_4 is a vertex of X^+ furthest from x, whose gate s'_4 belongs to $\mathbf{S}''_4(x)$, and t_4 is a vertex of X^+ such that its gate t'_4 belongs to $\mathbf{S}'_4(x)$ and the ordinate of t'_4 is closest to x_b.

Let $\alpha_4(X) := (p_1, q_1, p_2, q_2, p_3, q_3, p_4, q_4)$, where p_i is a vertex of X^- closest to x, whose gate p'_i belongs to $\mathbf{S}'_i(x)$, and q_i is a vertex of X^- closest to x, whose gate q'_i belongs to $\mathbf{S}''_i(x)$. If any of the vertices of the four groups is not defined, then its corresponding coordinate in $\alpha(X)$ is set to $*$.

The reconstructor $\beta(Y)$. Let Y be a vector of 22 coordinates corresponding to a realizable sample and grouped into four parts $Y_1 := (y_1, y_2)$, $Y_2 := (y_3, y_4, y_5, y_6)$, $Y_3 := (y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14})$, and $Y_4 := (y_{15}, y_{16}, y_{17}, y_{18}, y_{19}, y_{20}, y_{21}, y_{22})$. The reconstructor $\beta(Y)$ returns a ball $B_{r''}(y)$ by performing the following steps (see Fig. 3 (right)):

1. Using Y_1, canonically isometrically embed $I(y_1, y_2)$ into $\mathbb{Z}^2$ as $\mathbf{I}$.
2. Using Y_2, compute the gates y'_i of y_i in $\mathbf{I}$ and compute the region $\mathbf{R}$ as the intersection of the halfplanes $\mathbf{H}_{\leq y_{1b}}$, $\mathbf{H}_{\leq y_{2a}}$, $\mathbf{H}_{\geq y_{3b}}$, and $\mathbf{H}_{\geq y_{4a}}$ with $\mathbf{I}$.
3. Using Y_3, compute the set $\mathbf{R}' \subseteq \mathbf{R}$ of all $y = (y_a, y_b) \in \mathbf{R}$ such that the gates $y'_7, y'_8, y'_9, y'_{10}, y'_{11}, y'_{12}, y'_{13}, y'_{14}$ belong to $\mathbf{S}'_1(y)$, $\mathbf{S}''_1(y)$, $\mathbf{S}''_2(y)$, $\mathbf{S}'_2(y)$, $\mathbf{S}'_3(y)$, $\mathbf{S}''_3(y)$, $\mathbf{S}''_4(y)$, $\mathbf{S}'_4(y)$, resp. For each $y \in \mathbf{R}'$, let r'_y be the *smallest* radius such that $Y_1 \cup Y_3 \subseteq B_{r'_y}(y)$.
4. Using Y_4, compute the region $\mathbf{R}'' \subseteq \mathbf{R}$ consisting of all the vertices $y \in \mathbf{R}$ such that the gates $y'_{15}, y'_{16}, \ldots, y'_{21}, y'_{22}$ belong to the strips $\mathbf{S}'_1(y)$, $\mathbf{S}''_1(y)$, …, $\mathbf{S}'_4(y)$, $\mathbf{S}''_4(y)$, respectively. For each $y \in \mathbf{R}''$, let r''_y be the *largest* radius such that $B_{r''_y}(y) \cap Y_4 = \varnothing$.
5. Let $\mathbf{R}_0 := \{y \in \mathbf{R}' \cap \mathbf{R}'' : r''_y \geq r'_y\}$ and return as $\beta(Y)$ any ball $B_{r''_y}(y)$ with $y \in \mathbf{R}_0$.

▶ **Proposition 10.** *For any cube-free median graph G, the pair (α, β) of vectors defines a proper labeled sample compression scheme of size 22 for $\mathcal{B} = \mathcal{B}(G)$.*

5 Interval Graphs

For any interval graph $G = (V, E)$, we construct proper LSCS of size 4 for $\mathcal{B}(G)$ and $\mathcal{B}_r(G)$. We consider a representation of G by a set of segments $J_v, v \in V$ of $\mathbb{R}$ with pairwise distinct ends. For any $u \in V$, we denote by $J_u = [s_u, e_u]$ its segment, where s_u is the start of J_u, and e_u is the end of J_u, *i.e.*, $s_u \le e_u$. We use the following property of interval graphs:

▶ **Lemma 11.** *If $u, v \in B_r(x)$, $s_u, s_z < s_v$, and $e_u < e_v, e_z$, then $z \in B_r(x)$.*

Proof. Since $s_z < s_v$ and $e_u < e_z$, if J_u and J_v intersect, then J_z covers the segment $[s_v, e_u]$, and otherwise, J_z intersects $[e_u, s_v]$. Let P be a path obtained from a shortest (x, u)-path of G by removing u, and Q be a path obtained from a shortest (x, v)-path by removing v. The union J_S of all segments of $S := P \cup \{x\} \cup Q$ intersects J_u and J_v. If J_u and J_v intersect, then J_z covers $[s_v, e_u]$, and thus, intersects J_S. Otherwise, J_S covers $[e_u, s_v]$, and J_z intersects $[e_u, s_v]$. In both cases, J_z and J_S intersect, whence a segment of S intersects J_z. Since all segments of S are at distance at most $r - 1$ from x, $z \in B_r(x)$. ◀

Let X be a realizable sample for $\mathcal{B}(G)$. A *farthest pair* of X^+ is a pair $\{u^+, v^+\}$ such that u^+ is the vertex in X^+ whose interval J_{u^+} ends farthest to the left, and v^+ is the vertex in X^+ whose interval J_{v^+} begins farthest to the right, *i.e.*, for any $w \in X^+$, we have $e_{u^+} < e_w$ and $s_w < s_{v^+}$. If $u^+ \ne v^+$, then $[e_{u^+}, s_{v^+}] \cap J_w \ne \varnothing$ for any $w \in X^+$. If $u^+ = v^+$, then $J_{u^+} \subseteq J_w$ for any $w \in X^+$. A vertex p^- of X^- is a *left-bounder* if there is a ball $B_r(x)$ realizing X such that $e_{p^-} < s_x$ and, for all $p \in X^-$ with $e_p < s_x$, it holds that $e_p \le e_{p^-}$. Analogously, a vertex q^- of X^- is a *right-bounder* if there is a ball $B_r(x)$ realizing X such that $e_x < s_{q^-}$ and, for all $q \in X^-$ with $e_x < s_q$, it holds that $s_{q^-} \le s_q$. If p^- is a left-bounder and q^- is a right-bounder, then $\{p^-, q^-\}$ is a *bounding pair* of X^-. The farthest pair $\{u^+, v^+\}$ of X^+ and the bounding pair $\{p^-, q^-\}$ of X^- have the following properties:

▶ **Lemma 12.** *If $u^+, v^+ \in B_r(x)$ and $r > 0$, then $X^+ \subseteq B_r(x)$.*

Proof. Pick any $w \in X^+ \setminus \{u^+, v^+\}$. By the definition of u^+ and v^+, we have $s_w < s_{v^+}$ and $e_{u^+} < e_w$. If $u^+ \ne v^+$, then $s_{u^+} < s_{v^+}$ and $e_{u^+} < e_{v^+}$, and so, $s_{u^+}, s_w < s_{v^+}$ and $e_{u^+} < e_{v^+}, e_w$. By Lemma 11, $w \in B_r(x)$. Now, let $u^+ = v^+$. Then, $J_{u^+} \subset J_w$, and thus, any segment intersecting J_{u^+} also intersects J_w. Consequently, w is included in any ball of G of radius $r > 0$ containing u^+, and, in particular, $w \in B_r(x)$. ◀

▶ **Lemma 13.** *If $e_{p^-} < s_x$ and $p^- \notin B_r(x)$, then, for all $z \in X^-$ with $e_z < e_{p^-}$, $z \notin B_r(x)$. Also, if $e_x < s_{q^-}$ and $q^- \notin B_r(x)$, then, for all $w \in X^-$ with $s_{q^-} < s_w$, $w \notin B_r(x)$.*

Proof. For the first statement, towards a contradiction, suppose that $e_{p^-} < s_x$ and $p^- \notin B_r(x)$, but there exists $z \in X^-$ such that $e_z < e_{p^-}$ and $z \in B_r(x)$. Then, $s_z, s_{p^-} < s_x$ since $s_z \le e_z < e_{p^-} < s_x$, and $e_z < e_x, e_{p^-}$ as $e_z < e_{p^-} < s_x \le e_x$. By Lemma 11, $p^- \in B_r(x)$, a contradiction. For the second statement, suppose by way of contradiction that $e_x < s_{q^-}$ and $q^- \notin B_r(x)$, but there exists $w \in X^-$ such that $s_{q^-} < s_w$ and $w \in B_r(x)$. Then, $s_x, s_{q^-} < s_w$ since $s_x \le e_x < s_{q^-} < s_w$, and $e_x < e_w, e_{q^-}$ as $e_x < s_{q^-} < s_w \le e_w$. By Lemma 11, $q^- \in B_r(x)$, a contradiction. ◀

The compressor $\alpha(X)$ of X is a vector with four coordinates grouped into two pairs: $\alpha(X) := (\alpha_1(X), \alpha_2(X))$. The pair $\alpha_1(X)$ is a farthest pair $\{u^+, v^+\}$ of X^+ and the pair $\alpha_2(X)$ is a bounding pair $\{p^-, q^-\}$ of X^-. We use the symbol $*$ to indicate that the respective coordinate of $\alpha(X)$ is empty. We define $\alpha(X)$ as follows:

(C1) if $X^+ = \varnothing$, then set $\alpha_1(X) = \alpha_2(X) := (*, *)$;
(C2) if $X^+ = \{x\}$, then set $\alpha_1(X) := (x, *)$ and $\alpha_2(X) := (*, *)$;
(C3) if $|X^+| \geq 2$, then set $\alpha_1(X) := (u^+, v^+)$ if $u^+ \neq v^+$ and $\alpha_1(X) := (*, v^+)$ if $u^+ = v^+$;
 (C3i) if $X^- = \varnothing$, then set $\alpha_2(X) := (*, *)$;
 (C3ii) if there exists a bounding pair of X^-, then set $\alpha_2(X) := (p^-, q^-)$;
 (C3iii) if there exists a left-bounder, but not a right-bounder of X^-, then set $\alpha_2(X) := (p^-, *)$;
 (C3iv) if there exists a right-bounder, but not a left-bounder vertex of X^-, then set $\alpha_2(X) := (*, q^-)$.

The reconstructor β takes any signed vector Y on four coordinates grouped into two pairs Y_1 and Y_2 from $\mathrm{Im}(\alpha)$, and returns a ball $\beta(Y)$ defined as follows:
(R1) if $Y_1 = Y_2 = (*, *)$, then $\beta(Y)$ is the empty ball;
(R2) if $Y_1 = (y_1, *)$ and $Y_2 = (*, *)$, then $\beta(Y)$ is the ball of radius 0 centered at y_1;
(R3) if $Y_1 = (y_1, y_2)$ or $Y_1 = (*, y_2)$, then $\beta(Y)$ is any ball $B_r(x)$ of radius $r \geq 1$ containing Y_1, not intersecting Y_2, and such that:
 (R3i) if $Y_2 = (*, *)$, then no condition;
 (R3ii) if $Y_2 = (y_3, y_4)$, then $e_{y_3} < s_x$ and $e_x < s_{y_4}$;
 (R3iii) if $Y_2 = (y_3, *)$, then $e_{y_3} < s_x$;
 (R3iv) if $Y_2 = (*, y_4)$, then $e_x < s_{y_4}$.

Now, let X be a realizable sample for $\mathcal{B}_r(G)$. If $|X^+| \geq 2$ or $r \geq 1$, then we define α and β as above, since, in these cases, we do not specify the radius of the ball realizing X in α, nor the radius of the ball returned by β. So, we can exhibit a proper LSCS of size 4 for $\mathcal{B}_r(G)$ if we can deal with the case $|X^+| \leq 1$. The only difference is that if $|X^+| \leq 1$, then we set $\alpha_2(X)$ as in Case (C3), but we set $\alpha_1(X) := (*, *)$ when $X^+ = \varnothing$, and $\alpha_1(X) := (*, x)$ when $X^+ = \{x\}$. Now, let $r = 0$. If $|X^+| = 0$ and there is a ball $B_0(y)$ such that $y \notin X^-$ and $e_y < e_z$ for any $z \in V, z \neq y$, then $\alpha(X) := ((*, *), (*, *))$. Otherwise, if $|X^+| = 0$, there is a ball $B_0(y)$ such that $y \notin X^-$, $w' \in X^-$, $e_{w'} < e_y$, and, for all $w \in V$ with $e_w < e_y$, we have $e_w \leq e_{w'}$. In this case, $\alpha(X) := ((*, *), (w', *))$. If $X^+ = \{x\}$, set $\alpha(X) := ((x, *), (*, *))$. Given any signed vector Y on four coordinates, β returns a ball $\beta(Y)$ defined as follows. If $Y = ((*, *), (*, *))$, then $\beta(Y)$ is the ball $B_0(x)$ such that $e_x < e_z$ for any $z \in V \setminus \{x\}$. If $Y = ((*, *), (y_3, *))$, then $\beta(Y)$ is the ball $B_0(x)$ such that $e_{y_3} < e_x$, and, for all $w \in V$ with $e_w < e_x$, it holds that $e_w \leq e_{y_3}$. Lastly, if $Y = ((x, *), (*, *))$, then $\beta(Y)$ is the ball $B_0(x)$.

▶ **Proposition 14.** *For any interval graph $G = (V, E)$, the pair (α, β) of vectors defines a proper labeled sample compression scheme of size 4 for $\mathcal{B}(G)$ and $\mathcal{B}_r(G)$.*

Proof. Let X be a realizable sample for $\mathcal{B}(G)$ (the case of $\mathcal{B}_r(G)$ is similar), $Y = \alpha(X)$, and $B_r(x) = \beta(Y)$. The cases (Rk) and their subcases in the definition of β correspond to the cases (Ck) and their subcases in the definition of α. The correctness is trivial if $k = 1, 2$. Now, let $k = 3$. Since Y_1 always contains a farthest pair of X^+ and the returned ball $B_r(x)$ contains Y_1 and $r \geq 1$, by Lemma 12, $X^+ \subseteq B_r(x)$. Furthermore, in Case (C3), any ball realizing X must have a radius $r \geq 1$ since $|X^+| \geq 2$. Now, we prove that $X^- \cap B_r(x) = \varnothing$. This is trivial in subcase (R3i) since $X^- = \varnothing$. In the remaining subcases of (R3), $X^- \cap B_r(x) = \varnothing$ follows from the definition of the corresponding subcase of case (C3) and Lemma 13. ◀

6 Hyperbolic graphs

A (ρ,μ)*-approximate proper labeled sample compression scheme of size* k for the family of balls $\mathcal{B}(G)$ of a graph G compresses any realizable sample X to a subsample $\alpha(X)$ of support of size k, such that $\beta(\alpha(X))$ is a ball $B_r(x)$ such that $X^+ \subseteq B_{r+\rho}(x)$ and $X^- \cap B_{r-\mu}(x) = \varnothing$. Let (V,d) be a metric space and $w \in V$. Let $\delta \geq 0$. A metric space (X,d) is δ*-hyperbolic* [18] if, for any four points u,v,x,y of X, the two larger of the sums $d(u,v)+d(x,y)$, $d(u,x)+d(v,y)$, and $d(u,y)+d(v,x)$, differ by at most $2\delta \geq 0$. Next, we show that δ-hyperbolic graphs admit a $(2\delta,3\delta)$-approximate labeled sample compression scheme of size 2.

An interval $I(u,v)$ of a graph is ν*-thin* if $d(x,y) \leq \nu$ for any two points $x,y \in I(u,v)$ with $d(u,x)=d(u,y)$ and $d(v,x)=d(v,y)$. Intervals of δ-hyperbolic graphs are 2δ-thin. A metric space (X,d) is *injective* if, whenever X is isometric to a subspace Z of a metric space (Y,d'), there is a map $f: Y \to Z$ such that $f(z)=z$ for any $z \in Z$ and $d'(f(x),f(y)) \leq d'(x,y)$ for any $x,y \in Y$. By a construction of Isbell [20] (rediscovered by Dress [15]), any metric space (V,d) has an *injective hull* $E(V)$, *i.e.*, the smallest injective metric space into which (V,d) isometrically embeds. Lang [21] proved that the injective hull of a δ-hyperbolic space is δ-hyperbolic. It was shown in [15] that the injective hull $T := T(u,v,y,w)$ of a metric space on 4 points u,v,y,w is a rectangle $R := R(u',v',y',w')$ with four attached tips uu',vv',yy',ww' (one or several tips may reduce to a single point or R may reduce to a segment or a single point). The smallest side of R is exactly the hyperbolicity of the quadruplet u,v,y,w.

Let X be a realizable sample of $\mathcal{B}(G)$ and $\{u^+,v^+\}$ be a diametral pair of X^+. Let $B_{r^*}(y)$ be a ball of smallest radius such that $X^+ \subseteq B_{r^*}(y)$ and $X^- \cap B_{r^*}(y) = \varnothing$. We set $\alpha(X) := \varnothing$ if $X^+ = \varnothing$, $\alpha(X) := X^+$ if $|X^+| = 1$, and $\alpha(X) := \{u^+,v^+\}$ if $|X^+| \geq 2$. Given a subset Y of size at most 2, the reconstructor returns $\beta(Y) = \varnothing$ if $Y = \varnothing$, $\beta(Y) = B_0(y)$ if $Y = \{y\}$, and $\beta(Y) = B_{d(y_1,y_2)/2}(x)$ if $Y = \{y_1,y_2\}$, where x is the middle of a (y_1,y_2)-geodesic.

▶ **Proposition 15.** *For any δ-hyperbolic graph $G=(V,E)$, the pair (α,β) defines a $(2\delta,3\delta)$-approximate proper labeled sample compression scheme of size 2 for $\mathcal{B}(G)$.*

Proof. We first show that $X^+ \subseteq B_{r+2\delta}(x)$, where $r = d(u^+,v^+)/2$ and x is a middle of a (u^+,v^+)-geodesic. Pick any $w \in X^+$. Since u^+,v^+ is a diametral pair of X^+, $d(u^+,w) \leq 2r$ and $d(v^+,w) \leq 2r$. We also have $d(u^+,v^+) = 2r$ and $d(x,u^+) = d(x,v^+) = r$. Thus, the three distance sums have the form $d(u^+,w)+d(x,v^+) \leq 3r$, $d(v^+,w)+d(x,u^+) \leq 3r$, and $d(u^+,v^+)+d(x,w) = 2r+d(x,w)$. By the definition of δ-hyperbolicity, we conclude that either $d(x,w) \leq r$ (if $d(u^+,v^+)+d(x,w)$ is at most $3r$) or $d(x,w) \leq r+2\delta$ (if $d(u^+,v^+)+d(x,w)$ is the largest sum). Hence, $w \in B_{r+2\delta}(x)$. We now show that $X^- \cap B_{r-3\delta}(x) = \varnothing$. Pick $w \in X^-$ and consider the injective hull T of the points $\{u^+,v^+,y,w\}$. T is a rectangle R with four tips (see Fig. 4) and is a subspace of the injective hull $E(V)$. Since $w \in X^-$, $w \notin B_{r^*}(y)$. Since $u^+,v^+ \in B_{r^*}(y)$, we deduce that $d(y,w) > d(y,u^+)$ and $d(y,w) > d(y,v^+)$. Let x' be a point of $I(u^+,v^+) \cap T$ such that $d(u^+,x') = d(u^+,x) = r$ and $d(v^+,x') = d(v^+,x) = r$. Since the injective hull T is δ-hyperbolic, its intervals are 2δ-thin, and thus, $d(x,x') \leq 2\delta$.

Case 1. u^+, v^+, y, and w are as in Fig. 4(1). First, suppose that x' belongs to the tip between u^+ and u' or to the segment between u' and v'. Since y' and w' belong to a common geodesic from y to w and from y to v^+, and since $v^+ \in B_{r^*}(y)$ and $w \notin B_{r^*}(y)$, we deduce that $d(w,w') > d(w',v^+) \geq d(v',v^+)$. Consequently, $d(v',w) > d(v',v^+)$. If x' is located on the tip between u^+ and u' or on the segment between u' and v', then, since $r = d(x',v^+) = d(x',v') + d(v',v^+)$ and $d(x',w) = d(x',v') + d(v',w)$, we obtain that $w \notin B_r(x')$. Since $d(x,x') \leq 2\delta$, $w \notin B_{r-2\delta}(x)$. If x' belongs to the tip between v' and v^+, then $r = d(x',v^+) \leq d(v',v^+) \leq d(v',w)$, whence $w \notin B_r(x')$ and $w \notin B_{r-2\delta}(x)$.

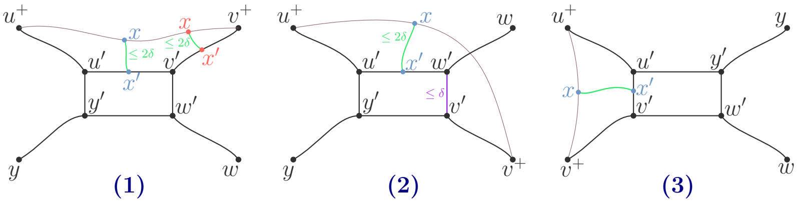


Figure 4 Cases 1-3 of Proposition 15.

Case 2. u^+ and v^+, and y and w are opposite in T as in Fig. 4(2). Consider x' to be on the boundary of T containing the vertices u', w', and v'. Since $v^+ \in B_{r^*}(y)$ and $w \notin B_{r^*}(y)$, then $d(v', w') + d(w', w) > d(v', v^+)$. Note also that $d(v', w') \le \delta$, and thus, $d(w, v') > d(v', v^+) - \delta$. Independently of the location of x' on the boundary of T, $w \notin B_{r-\delta}(x')$. Thus, $w \notin B_{r-3\delta}(x)$.

Case 3. u^+, v^+, y, and w are as in Fig. 4(3). Since w' belongs to a geodesic between y and w and between y and v^+, and $w \notin B_{r^*}(y), v^+ \notin B_{r^*}(y)$, we deduce that $d(w', w) > d(w', v') + d(v', v^+) \ge d(v', v^+)$. Independently of the location of x', we obtain that $w \notin B_{r-2\delta}(x)$. ◀

7 Perspectives

A direction of interest is to design proper sample compression schemes for balls of radius r in trees of cycles or cube-free median graphs. Designing sample compression schemes of size $O(d)$ for balls in general median graphs G of dimension d is also open, as well as whether the VC-dimension of $\mathcal{B}(G)$ is $O(d)$ or not. For general median graphs, it no longer holds that the interval between a diametral pair of X^+ contains a center of a ball realizing X. However, one can show that X^+ contains $2d$ vertices whose convex hull contains such a center. This convex hull can be d-dimensional and it is unclear how to encode the center in this region.

Other open questions are to design proper sample compression schemes of constant size for balls of planar graphs and of size $O(\omega(G))$ for balls of a chordal graph G. In [8], we showed that the former is possible for balls of radius 1, and that the latter is possible for split graphs. Finding proper sample compression schemes of size $O(\omega(G))$ for $\mathcal{B}(G)$ is also interesting for other classes of graphs from metric graph theory: bridged graphs (generalizing chordal graphs) and Helly graphs; for their definitions and characterizations, see [1].

References

1 H.-J. Bandelt and V. Chepoi. Metric graph theory and geometry: a survey. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry. Twenty Years later*, volume 453 of *Contemp. Math.*, pages 49–86. AMS, Providence, RI, 2008.

2 H.-J. Bandelt, V. Chepoi, and D. Eppstein. Ramified rectilinear polygons: coordinatization by dendrons. *Discr. Comput. Geom.*, 54:771–797, 2015.

3 H.-J. Bandelt, V. Chepoi, and K. Knauer. COMs: complexes of oriented matroids. *J. Combin. Th., Ser. A*, 156:195–237, 2018.

4 L. Beaudou, P. Dankelmann, F. Foucaud, M. A. Henning, A. Mary, and A. Parreau. Bounding the order of a graph using its diameter and metric dimension: A study through tree decompositions and VC dimension. *SIAM J. Discr. Math.*, 32(2):902–918, 2018.

5 S. Ben-David and A. Litman. Combinatorial variability of Vapnik-Chervonenkis classes with applications to sample compression schemes. *Discr. Appl. Math.*, 86:3–25, 1998.

6 A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. Ziegler. *Oriented Matroids*, volume 46 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1993.

7 N. Bousquet and S. Thomassé. VC-dimension and Erdős–Pósa property. *Discr. Math.*, 338:2302–2317, 2015.

8 J. Chalopin, V. Chepoi, F. Mc Inerney, S. Ratel, and Y. Vaxès. Sample compression schemes for balls in graphs. *arXiv*, 2022. `arXiv:2206.13254`.

9 J. Chalopin, V. Chepoi, S. Moran, and M. K. Warmuth. Unlabeled sample compression schemes and corner peelings for ample and maximum classes. *J. Comput. Syst. Sci.*, 127:1–28, 2022.

10 V. Chepoi, B. Estellon, and Y. Vaxès. On covering planar graphs with a fixed number of balls. *Discr. Comput. Geom.*, 37:237–244, 2007.

11 V. Chepoi and M. Hagen. On embeddings of CAT(0) cube complexes into products of trees via colouring their hyperplanes. *J. Combin. Th., Ser. B*, 103:428–467, 2013.

12 V. Chepoi, K. Knauer, and M. Philibert. Labeled sample compression schemes for complexes of oriented matroids. *arXiv*, 2021. `arXiv:2110.15168`.

13 V. Chepoi, A. Labourel, and S. Ratel. Distance and routing labeling schemes for cube-free median graphs. *Algorithmica*, 83:252–296, 2021.

14 V. Chepoi and D. Maftuleac. Shortest path problem in rectangular complexes of global nonpositive curvature. *Computational Geometry*, 46:51–64, 2013.

15 A. W. M. Dress. Trees, tight extensions of metric spaces, and the cohomological dimension of certain groups. *Advances in Mathematics*, 53:321–402, 1984.

16 G. Ducoffe, M. Habib, and L. Viennot. Diameter computation on H-minor free graphs and graphs of bounded (distance) VC-dimension. In *ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1905–1922, 2020.

17 S. Floyd and M. K. Warmuth. Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine Learning*, 21:269–304, 1995.

18 M. Gromov. Hyperbolic groups. *Essays in group theory*, pages 75–263, 1987.

19 D. Helmbold, R. Sloan, and M. K. Warmuth. Learning nested differences of intersection-closed concept classes. *Machine Learning*, 5:165–196, 1990.

20 J. R. Isbell. Six theorems about injective metric spaces. *Comment. Math. Helv.*, 39:65–76, 1964.

21 U. Lang. Injective hulls of certain discrete metric spaces and groups. *J. Topol. Anal.*, 5:297–331, 2013.

22 N. Littlestone and M. K. Warmuth. Relating data compression and learnability. *Unpublished*, 1986.

23 S. Moran and M. K. Warmuth. Labeled compression schemes for extremal classes. In *ALT 2016*, pages 34–49, 2016.

24 S. Moran and A. Yehudayoff. Sample compression schemes for VC classes. *J. ACM*, 63:1–21, 2016.

25 D. Pálvölgyi and G. Tardos. Unlabeled compression schemes exceeding the VC-dimension. *Discr. Appl. Math.*, 276:102–107, 2020.

26 M. Pilipczuk and S. Siebertz. Kernelization and approximation of distance-r independent sets on nowhere dense graphs. *Eur. J. Comb.*, 94:103223, 2021.

27 V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

On Algorithms Based on Finitely Many Homomorphism Counts

Yijia Chen
Shanghai Jiao Tong University, Shanghai, China

Jörg Flum
Albert-Ludwigs-Universität, Freiburg i.Br., Germany

Mingjun Liu
Fudan University, Shanghai, China

Zhiyang Xun
Shanghai Jiao Tong University, Shanghai, China

Abstract

It is well known [16] that up to isomorphism a graph G is determined by the homomorphism counts $\hom(F, G)$, i.e., by the number of homomorphisms from F to G where F ranges over all graphs. Moreover, it suffices that F ranges over the graphs with at most as many vertices as G. Thus, in principle, we can answer any query concerning G with only accessing the $\hom(\cdot, G)$'s instead of G itself. In this paper, we deal with queries for which there is a *hom algorithm*, i.e., there are *finitely many* graphs $F_1, \dots, F_k$ such that for any graph G whether it is a YES-instance of the query is already determined by the vector $\overrightarrow{\hom}_{F_1,\dots,F_k}(G) := (\hom(F_1, G), \dots, \hom(F_k, G))$.

We observe that planarity of graphs and 3-colorability of graphs, properties expressible in monadic second-order logic, have no hom algorithm. On the other hand, queries expressible as a Boolean combination of universal sentences in first-order logic FO have a hom algorithm. Even though it is not easy to find FO definable queries without a hom algorithm, we succeed to show this for the non-existence of an isolated vertex, a property expressible by the FO sentence $\forall x \exists y Exy$, somehow the "simplest" graph property not definable by a Boolean combination of universal sentences. These results provide a characterization of the prefix classes of first-order logic with the property that each query definable by a sentence of the prefix class has a hom algorithm.

For *adaptive* hom algorithms, i.e., algorithms that might access a $\hom(F_{i+1}, G)$ with F_{i+1} depending on $\hom(F_j, G)$ for $1 \le j \le i$ we show that *three* homomorphism counts $\hom(\cdot, G)$ are sufficient and in general necessary to determine the (isomorphism type of) G. In particular, by three adaptive queries we can answer any question on G. Moreover, adaptively accessing two $\hom(\cdot, G)$'s is already enough to detect an isolated vertex.

In 1993 Chaudhuri and Vardi [6] showed the analogue of the Lovász Isomorphism Theorem for the right homomorphism vector of a graph G, i.e, the vector of values $\hom(G, F)$ where F ranges over all graphs characterizes the isomorphism type of G. We study to what extent our results carry over to the right homomorphism vector.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases homomorphism numbers, hom algorithms, adaptive hom algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.32

Related Version *Previous Version*: https://arxiv.org/abs/2111.13269

Funding The collaboration of the first two authors is funded by the Sino-German Center for Research Promotion (GZ 1518). Yijia Chen is supported by National Natural Science Foundation of China (Project 61872092).

Acknowledgements Mingjun Liu is supervised by Yijia Chen in the Mathematical Logic Program for undergraduate students at Fudan University.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 32; pp. 32:1–32:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In [16], one of the first papers on graph homomorphisms, Lovász proved that graphs G and H are isomorphic if and only if for all graphs F the number $\hom(F, G)$ of homomorphisms from F to G is equal to the number $\hom(F, H)$ of homomorphisms from F to H. Recently, this result has attracted a lot of attention in various contexts, e.g., algorithms and complexity [3, 9], machine learning [2, 14], and logic [1, 15]. Among others, it provides a powerful reduction of problems concerning graph structures to questions on the number of homomorphisms.

Lovász' result says that the infinite vector $\overrightarrow{\hom}(G) := (\hom(F, G))_{F \text{ a graph}}$ determines the graph G up to isomorphism. For a class C of graphs set $\overrightarrow{\hom}_{\mathsf{C}}(G) := (\hom(F, G))_{F\in\mathsf{C}}$. Using Lovász' *Cancellation Law* [17] (see Theorem 7.9) it is easy to see that for some classes C, including the class of 3-colorable graphs and the class of graphs that can be mapped homomorphically to an odd cycle, $\overrightarrow{\hom}_{\mathsf{C}}(G)$ already determines G up to isomorphisms. A further example: the class of 2-degenerate graphs has this property [12]. For other natural classes of graphs, $\overrightarrow{\hom}_{\mathsf{C}}(G)$ does not have the full power of distinguishing non-isomorphic graphs but characterizes interesting graph properties (e.g., see [10]).

We turn to results more relevant for the algorithmic problems we are interested in. Actually Lovász' proof shows that in order to determine the isomorphism type of G it is sufficient to consider the homomorphism counts $\hom(F, G)$ for the graphs F with at most as many vertices as G. As a consequence, given an oracle to $\overrightarrow{\hom}(G)$, we might answer any query by first recovering G and then computing the query on G. However, such a naive algorithm requires *exponentially* many entries in $\overrightarrow{\hom}(G)$, i.e., $\hom(F, G)$ for all isomorphism types of graphs F with at most as many vertices as G, rendering any practical implementation beyond reach.

There are queries that can be answered very easily using $\overrightarrow{\hom}(G)$, e.g., to decide whether G has a clique of size k, all we need to know is $\hom(K_k, G)$ where K_k is the complete graph on k vertices. So ideally, one would hope that to answer a query on G it suffices to access $\overrightarrow{\hom}(G)$ for finitely many fixed graphs independent of G.

The question of using $\overrightarrow{\hom}(G)$ to answer queries algorithmically has been raised before. In [9] Curticapean et al. observed that counting (induced) subgraphs isomorphic to a fixed graph F can be reduced to computing appropriate linear combinations of subvectors of $\overrightarrow{\hom}(G)$. Thereby they introduced the so-called graph motif parameters. Using this framework, they were able to design some algorithms faster than the known ones to count various specific subgraphs and induced subgraphs. These results can be understood as answering counting queries using $\hom(\cdot, G)$'s. More explicitly, Grohe [15] asked whether it is possible to answer any C^{k+1}-query in *polynomial time* by accessing $\hom(F, G)$ for graphs F of tree-width bounded by k. Here, C^{k+1} denotes counting first-order logic with $k+1$ variables [5]. Observe that without the polynomial time constraint such an algorithm exists because graphs G and H cannot be distinguished by C^{k+1} if and only if $\hom(F, G) = \hom(F, H)$ for finitely many graphs F of tree-width bounded by k [12] (see also [10]).

Our contributions

In this paper we study what Boolean queries (equivalently, graph properties) can be answered using a constant number of homomorphism counts. More precisely, let C be a class of graphs closed under isomorphism. We ask: are there graphs $F_1, \dots, F_k$ such that for any graph G whether $G \in \mathsf{C}$ can be decided knowing $\overrightarrow{\hom}_{F_1,\dots,F_k}(G) := (\hom(F_1, G), \dots, \hom(F_k, G))$.

In Section 4 we observe that this is the case if C can be defined by a sentence of first-order logic (FO) that is a Boolean combination of *universal* sentences. For $d \geq 1$ this includes the class of graphs of maximum degree d, of tree-depth [7] exactly d, and the class of graphs of

SC-depth [8] exactly d (but also the classes where we replace "exactly d" by "at most d"). On the negative side, in Section 6 we show that for any $k \geq 1$ and any $F_1, \dots, F_k$ there are graphs G and H such that

- $\overrightarrow{\text{hom}}_{F_1,\dots,F_k}(G) = \overrightarrow{\text{hom}}_{F_1,\dots,F_k}(H)$ and G contains an isolated vertex but H does not.

That is, any $\overrightarrow{\text{hom}}_{F_1,\dots,F_k}(G)$ is not sufficient to detect the existence of an isolated vertex in G. This is our technically most challenging result; it requires non-trivial arguments using linear algebra. We introduce some of the tools and construction methods needed in this proof already in Section 5, thereby showing the corresponding result for the class of planar graphs and the class of 3-colorable graphs.

A graph G has no isolated vertex if it satisfies the FO-sentence $\forall x \exists y Exy$. Thus we know for what quantifier-prefix classes of FO-sentences all queries definable by a sentence of the class can be answered by $\overrightarrow{\text{hom}}_{F_1,\dots,F_k}(G)$ for some $F_1, \dots, F_k$ *independent* of G.

Answering a query using $\overrightarrow{\text{hom}}_{F_1,\dots,F_k}(\cdot)$ can be phrased as an algorithm checking this query with *non-adaptive* access to $\overrightarrow{\text{hom}}(G)$ on entries $F_1, \dots, F_k$. It is also very natural to allow access to $\overrightarrow{\text{hom}}(G)$ to be *adaptive*. Informally, on input G an *adaptive hom algorithm* still queries some $\text{hom}(F_1, G), \dots, \text{hom}(F_k, G)$, but for $i = 0, \dots, k-1$ the choice of F_{i+1} might depend on $\text{hom}(F_1, G), \dots, \text{hom}(F_i, G)$ (see Definition 7.2 for a precise description). It turns out that adaptive hom algorithms are extremely powerful. In Section 7 we first present an adaptive hom algorithm with *two* accesses to $\overrightarrow{\text{hom}}(G)$ that can decide whether G has no isolated vertices. Even more, the algorithm is able to compute the number of vertices of G of degree k for each $k \geq 0$. So in particular, it can decide whether G is regular. Furthermore, in Section 7 we provide an adaptive hom algorithm that queries *three* entries in $\overrightarrow{\text{hom}}(G)$ and determines (the isomorphism type of) G. Hence, it can answer any question on G. The downside of this algorithm is its superpolynomial running time, while all the aforementioned hom algorithms run in polynomial time (when provided with access to $\text{hom}(G)$). For graph classes that are relevant in applications it is a challenging task to study whether there is an algorithm where the size of the corresponding F's are polynomial in the size of G. We conjecture that there is no polynomial time algorithm that can reconstruct a graph G only accessing $\overrightarrow{\text{hom}}(G)$ (even without the requirement of a constant number of accesses).

Results in [1] may be interpreted as saying that often proper subvectors of the right homomorphism vector $\overleftarrow{\text{hom}}(G) := (\text{hom}(G, F))_{F \text{ a graph}}$ of a graph G are not so expressive as the corresponding subvectors of the (left) homomorphism vector. For our topic, the finite subvectors, in Section 8 we prove that our two "positive results" on hom algorithms (namely, Theorem 4.4 on Boolean combinations of universal sentences and Theorem 7.4 on the power of 3 adaptive hom algorithms) fail for the finite right subvectors. Even when the result is the same (e.g., there is no right hom algorithm showing the non-existence of isolated vertices) the proof and its complexity can be quite different.

Due to space limitations for some proofs we refer to the full version of the paper.

2 Preliminaries

We denote by $\mathbb{N}$ the set of natural numbers greater than or equal to 0. For $n \in \mathbb{N}$ let $[n] := \{1, 2, \dots, n\}$.

For graphs we use the notation $G = (V(G), E(G))$ common in graph theory. Here $V(G)$ is the nonempty set of vertices of G and $E(G)$ is the set of edges. We only consider finite, simple, and undirected graphs and briefly speak of graphs. To express that there is an edge connecting the vertices u and v of the graph G, we use (depending on the context) one of the notations $uv \in E(G)$ and $\{u, v\} \in E(G)$. For graphs G and H with disjoint vertex sets we denote by $G \mathbin{\dot\cup} H$ the *disjoint union of G and H*, i.e., the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. If the vertex sets are not disjoint, we tacitly pass to isomorphic copies with disjoint vertex sets. Similarly, $\dot\bigcup_{i \in I} G_i$ denotes the disjoint union of the graphs G_i with $i \in I$. For a graph G and $\ell \geq 1$ we denote by ℓG the disjoint union of ℓ copies of G.

For $n \geq 1$ we denote by K_n a clique with n vertices, by P_n a path of n vertices, and by C_n a cycle of n vertices.

For graphs G and H by $G \cong H$ we express that G and H are isomorphic. All classes of graphs considered in this paper are closed under isomorphism.

▶ **Definition 2.1.** Let G and H be graphs and $f : V(G) \to V(H)$. The function f is a *homomorphism* if $uv \in E(G)$ implies $f(u)f(v) \in E(H)$ for all $u, v \in V(G)$. It is an *embedding* (or *monomorphism*) if f is a homomorphism that is one-to-one. We call f an *epimorphism* if f is a homomorphism, the range of f is $V(H)$, and for every $u'v' \in E(H)$ there are $u, v \in V(G)$ such that $uv \in E(G)$ and $f(u) = u'$, $f(v) = v'$. We get the definitions of *strong homomorphism*, *strong embedding*, and *strong epimorphism* by additionally requiring in the previous definitions that $(uv \in E(G) \iff f(u)f(v) \in E(H))$ for all $u, v \in V(G)$.

We denote by $\textsc{Hom}(G, H)$ the set of homomorphisms from G to H, thus $\hom(G, H) := |\textsc{Hom}(G, H)|$ is the number of homomorphisms from G to H. Similarly, we define $s\text{-}\textsc{Hom}(G, H)$ and $\text{s-hom}(G, H)$ for strong homomorphisms and use corresponding notations for the other notions of morphisms. Finally, $\textsc{Aut}(G)$ and $\text{aut}(G)$ denote the set of automorphisms of G and their number, respectively.

The equalities (1) and (2) are easy to verify and will often be used tacitly. For graphs F_1, F_2, and G and $\text{xyz} \in \{\hom, \text{emb}\}$,

$$\hom(F_1 \mathbin{\dot\cup} F_2, G) = \hom(F_1, G) \cdot \hom(F_2, G), \tag{1}$$

$$\text{if } G \text{ is connected, then } \text{xyz}(G, F_1 \mathbin{\dot\cup} F_2) = \text{xyz}(G, F_1) + \text{xyz}(G, F_2). \tag{2}$$

Once and for all we fix an enumeration

$$F_1^0, F_2^0, F_3^0, \dots \tag{3}$$

of graphs such that each graph is isomorphic to exactly one graph in the list and such that $i \leq j$ implies $F_i^0 \leq F_j^0$. Here for graphs F and G by $F \leq G$ we mean that

$$|V(F)| < |V(G)| \;\text{ or }\; (|V(F)| = |V(G)| \text{ and } |E(F)| \leq |E(G)|).$$

In particular, F_1^0 is a graph whose vertex set is a singleton. We repeatedly use:

▶ **Theorem 2.2** (Lovász Isomorphism Theorem [16]). *Let G and H be graphs. If $\hom(F, G) = \hom(F, H)$ for all graphs F with $|V(F)| \leq \min\{|V(G)|, |V(H)|\}$, then G and H are isomorphic, i.e., the finite vector $\hom(F, G))_{F \text{ a graph with } |V(F)| \leq |V(G)|}$ determines G up to isomorphism.*

3 Algorithms accessing morphism counts

For what classes C of graphs is there a finite set F of graphs such that the membership of any graph G in C is determined by the values $\hom(F, G)$ where F ranges over F? This question leads to the following definition.

▶ **Definition 3.1.** Let C be a class of graphs. *A hom algorithm for* C *(with a constant number of non-adaptive accesses to homomorphism counts)* consists of a $k \geq 1$, graphs $F_1, \ldots, F_k$, and an $X \subseteq \mathbb{N}^k$ such that for all G,

$$G \in \mathsf{C} \iff (\hom(F_1, G), \ldots, \hom(F_k, G)) \in X.$$

We then say that the *hom algorithm decides* C. Analogously we define the notions of *emb algorithm*, *s-hom algorithm*, and *s-emb algorithm.*

Often we use the following fact, whose proof is immediate: A class C can be decided by a hom algorithm if and only if there is a finite set $\mathsf{F} = \{F_1, \ldots, F_k\}$ of graphs such that for all G and H (recall that $\overrightarrow{\hom}_{\mathsf{F}}(G) = (\hom(F_1, G), \ldots, \hom(F_k, G))$),

$$\overrightarrow{\hom}_{\mathsf{F}}(G) = \overrightarrow{\hom}_{\mathsf{F}}(H) \text{ implies } (G \in \mathsf{C} \iff H \in \mathsf{C}).$$

▶ **Remark 3.2.** If the set X in Definition 3.1 is decidable, then we easily extract an actual algorithm for C with an oracle to $\overrightarrow{\hom}(G)$. However the previous equivalence only holds for arbitrary X. Nevertheless, all our positive results have decidable X's. We use the current definition to ease presentation, and also to make our negative result, namely Theorem 6.1, stronger. Let C have a hom algorithm as in Definition 3.1. Then the set X can be chosen to be decidable if and only if C is decidable.

▶ **Examples 3.3.**

(a) By taking $k = 1$, a graph F whose vertex set is a singleton, and an arbitrary set $X \subseteq \mathbb{N}$, we get a hom algorithm for the class of graphs whose number of vertices is in X. In particular, for an undecidable X we get an undecidable class of graphs with a hom algorithm.

(b) Theorem 2.2 shows that every class that only contains finitely many graphs up to isomorphism can be decided by a hom algorithm.

(c) By passing from $k \geq 1$, $F_1, \ldots, F_k$, and $X \subseteq \mathbb{N}^k$ to $k \geq 1$, $F_1, \ldots, F_k$, and $\mathbb{N}^k \setminus X$, we see that with every class C also the class $\mathsf{C}^{\text{comp}} := \{G \mid G \notin \mathsf{C}\}$ has a hom algorithm.

By Definition 3.1 we have four types of algorithms (hom, emb, s-hom, s-emb). The following proposition shows that a class has an algorithm of one type if and only if it has an algorithm of any other type. This allows us to speak of a *query algorithm accessing morphism counts* (or *query algorithm* for short) if in the given context it is irrelevant to what type we refer.

▶ **Proposition 3.4.** *For a class* C *of graphs the following five statements are equivalent.*

(a) *There is a hom algorithm for* C.

(b) *There is an emb algorithm for* C.

(c) *There is an s-hom algorithm for* C.

(d) *There is an s-emb algorithm* C.

(e) *There is a hom algorithm for* C^c*, the class of graphs that are complements of graphs in* C *(the complement of a graph* G *is the graph* $G^c = \big(V(G), \{uv \mid u \neq v \text{ and } uv \notin E(G)\}\big)$.

The equivalence (a) ⇔ (b) and (c) ⇔ (d) are well known (e.g., see the proof of the Lovász Isomorphism Theorem in [15]). It uses the fact that every $h \in \textsc{Hom}(F, G)$ can be written as $h = f \circ g$, where for some graph F' we have $g \in \textsc{Epi}(F, F')$ and $f \in \textsc{Emb}(F', G)$. Clearly, $F' \leq F$ (as otherwise $\textsc{Epi}(F, F') = \emptyset$). Hence.

$$\hom(F, G) = \sum_{F' \leq F} \frac{1}{\operatorname{aut}(F')} \cdot \operatorname{epi}(F, F') \cdot \operatorname{emb}(F', G), \tag{4}$$

where the sum ranges over all isomorphism types of graphs F' with $F' \leq F$ and the corresponding equation for s-hom and s-emb.

As in the literature we did not find a proof of the equivalence (a) ⇔ (c), the main tool in our proof of Theorem 4.4, we present proofs of the equivalences (a) ⇔ (c) and its consequence (a) ⇔ (e) in the full version of the paper.

▶ **Remark 3.5.** The proof of the equivalences of (a) to (e) show: If for C we have a query algorithm of one type based on graphs $F_1, \dots, F_k$ and $m := \max\{|V(F_i)| \mid i \in [k]\}$, then for any other type we can compute finitely many graphs, all with at most m vertices, that are the graphs of a query algorithm for C of this other type.

4 FO-definable classes with query algorithms

We start by showing that every class of graphs that excludes a finite set of graphs as induced subgraphs has a query algorithm. Of course, the complement and the union of such classes again have such an algorithm. In terms of first-order logic this means that every class axiomatizable by a Boolean combination of universal sentences has a query algorithm.

For a finite set F of graphs we define the class $\textsc{Forb}(\mathsf{F})$ by

$$\textsc{Forb}(\mathsf{F}) := \big\{G \bigm| G \text{ has no induced subgraph isomorphic to a graph in } \mathsf{F}\big\}.$$

We say that *a class* C *of graphs is definable by a set of forbidden induced subgraphs* if there is a finite set F with $\mathsf{C} = \textsc{Forb}(\mathsf{F})$.

Examples of classes definable by a set of forbidden induced subgraphs are classes of bounded vertex cover number (attributed to Lovász), of bounded tree-depth [11], or even of bounded shrub-depth [13]. All these classes have a query algorithm:

▶ **Lemma 4.1.** *Every class of graphs definable by a set of forbidden induced subgraphs can be decided by a query algorithm.*

Proof. If $\mathsf{C} = \textsc{Forb}(\emptyset)$, we set $k = 1$, let F be an arbitrary graph, and take $X := \mathbb{N}$. Assume now that $\mathsf{C} = \textsc{Forb}(\mathsf{F})$ with $\mathsf{F} = \{F_1, \dots, F_k\}$ and $k \geq 1$. Then, for $X = \{(0, 0, \dots, 0)\} \subseteq \mathbb{N}^k$,

$$G \in \mathsf{C} \quad \iff \quad \big(\text{s-emb}(F_1, G), \dots, \text{s-emb}(F_k, G)\big) \in X.$$

Hence, k, $F_1, \dots, F_k$, and X constitute an s-emb algorithm for C. ◀

The following lemma shows that the universe of classes with query algorithms is closed under the Boolean operations. Part (a) was already mentioned as Examples 3.3 (c). We omit the straightforward proof.

▶ **Lemma 4.2.**

(a) *If* C *has a query algorithm, then so does* $\{G \mid G \notin \mathsf{C}\}$.

(b) *If* C *and* C' *have query algorithms, then* $\mathsf{C} \cap \mathsf{C}'$ *and* $\mathsf{C} \cup \mathsf{C}'$ *have query algorithms.*

Recall that *formulas* φ of *first-order logic* FO for graphs are built up from *atomic formulas* $x = y$ and Exy (where x, y are variables) using the Boolean connectives $\neg$, $\wedge$, and $\vee$ and the universal $\forall$ and existential $\exists$ quantifiers. A *sentence* is a formula without free variables (i.e., all variables of φ are in the scope of a corresponding quantifier). If φ is a sentence, we denote by $\mathsf{C}(\varphi)$ the class of graphs that are models of φ.

An FO-formula is *universal* if it is built up from atomic and negated atomic formulas by means of the connectives $\wedge$ and $\vee$ and the universal quantifier $\forall$. If in this definition we replace the universal quantifier by the existential one, we get the definition of an *existential formula*. The following result is well known (e.g., see [18]).

▶ **Lemma 4.3.** *Let* C *be a class of graphs. Then* C *is the class of graphs definable by a universal sentence if and only if* C *is definable by a set of forbidden induced subgraphs.*

By Lemma 4.1 – Lemma 4.3 we get:

▶ **Theorem 4.4.** *If the* FO*-sentence* φ *is a Boolean combination of universal sentences, then there is a query algorithm for* $\mathsf{C}(\varphi)$*.*

▶ **Remark 4.5.** The class $\mathsf{C}(3)$ of 3-regular graphs (each vertex has exactly 3 neighbors) is an example of a class decidable by a query algorithm, definable in FO but not by a Boolean combination of universal sentences. Indeed, using the following steps we get a query algorithm deciding whether a graph G belongs to $\mathsf{C}(3)$.

- We check whether $G \in \mathsf{C}(\leq 3)$, i.e., whether each vertex has at most 3 neighbors. Note that $\mathsf{C}(\leq 3)$ is definable by a universal sentence. Hence there is a hom algorithm for $\mathsf{C}(\leq 3)$, say consisting of $k \geq 1$, $F_1, \dots, F_k$, and $X_{\leq 3}$ $(\subseteq \mathbb{N}^k)$.
- We query $\hom(K_1, G)$ in order to get $n := |V(G)|$.
- We query $\hom(P_2, G)$, i.e., the number of homomorphisms from the path P_2 of two vertices to G. Then, G is 3-regular if and only if $\hom(P_2, G) = 3 \cdot n$.

Hence, we have a hom algorithm for $\mathsf{C}(3)$ consisting of $k+2$, $F_1, \dots, F_k, K_1, P_2$, and X where X consists of all tuples $(n_1, \dots, n_k, n, 3n)$ with $(n_1, \dots, n_k) \in X_{\leq 3}$ and $n \geq 1$.

The class $\mathsf{C}(3)$ is definable in FO by

$$\forall x \forall y_1 \dots \forall y_4 \exists z_1 \dots \exists z_3 \Big(\neg\big(\bigwedge_{1 \leq i < j \leq 4} y_i \neq y_j \wedge \bigwedge_{i \in [4]} Exy_i \big) \wedge \big(\bigwedge_{1 \leq i < j \leq 3} z_i \neq z_j \wedge \bigwedge_{j \in [3]} Exz_j \big) \Big).$$

If it would be definable by a Boolean combination of universal sentences, then it would be also definable by a sentence φ of the form $\varphi = \exists x_1 \dots \exists x_m \forall y_1 \dots \forall y_\ell \psi$ with $m, \ell \in \mathbb{N}$ and with quantifier-free ψ. Let G be a graph with more than $m+1$ vertices that is the disjoint union of copies of the clique K_4. Of course, G is 3-regular. Hence, G is a model of φ. In particular, there are vertices $u_1, \dots, u_m$ that satisfy in G the formula $\forall y_1 \dots \forall y_\ell \psi(x_1, \dots, x_m)$ if we interpret x_1 by u_1, …, x_m by u_m. Choose a vertex $u \in V(G) \setminus \{u_1, \dots, u_m\}$. Then, $G \setminus u$, the graph induced by G on $V(G) \setminus \{u\}$, is still a model of φ but not 3-regular.

By the previous remark the question arises whether every class $\mathsf{C}(\varphi)$ for an FO-sentence φ of the form$\forall x_1 \dots \forall x_m \exists y_1 \dots \exists y_\ell \psi$ with quantifier free ψ can be decided by a query algorithm. We will see that already for the simple formula $\forall x \exists y Exy$ of this type, the class $\mathsf{C}(\forall x \exists y Exy)$, i.e., the class of graphs not containing isolated vertices, has no query algorithm.

5 Planarity and 3-colorability

As just mentioned we want to show that no query algorithm detects the existence of an isolated vertex. In this section we prove the corresponding result for planarity, where some easy tools and construction methods relevant in the much more involved proof for isolated vertices are used. Essentially by a similar proof one can show the same result for 3-colorability. Note that the class of planar graphs and the class of 3-colorable graphs are definable in monadic second-order logic but not in first-order logic.

By the following lemma a class has no query algorithm if there is no emb algorithm for this class that only uses *connected graphs*:

▶ **Lemma 5.1.** *Let* C *be a class of graphs. Assume that for every finite set* K' *of connected graphs there are graphs* G *and* H *such that (a) and (b) hold.*

(a) *$G \in \mathsf{C}$ and $H \notin \mathsf{C}$.*

(b) *For all $F' \in \mathsf{K}'$ we have $\mathrm{emb}(F', G) = \mathrm{emb}(F', H)$.*

Then there is no hom algorithm for C, i.e., for every finite set K of graphs there are graphs G and H such that (c) and (d) hold

(c) *$G \in \mathsf{C}$ and $H \notin \mathsf{C}$.*

(d) *For all $F \in \mathsf{K}$ we have $\mathrm{hom}(F, G) = \mathrm{hom}(F, H)$.*

Proof. By (1), $\mathrm{hom}(F^1 \dot\cup F^2, F^3) = \mathrm{hom}(F^1, F^3) \cdot \mathrm{hom}(F^2, F^3)$ holds for arbitrary graphs F^1, F^2, F^3. Thus, if the class of connected components of graphs in a finite set K satisfies (d) for some G and H, then K satisfies (d), too. Hence, we can assume that the graphs in K are connected. Let $n := \max\{|V(F)| \mid F \in \mathsf{K}\}$ and $\mathsf{K}' := \{F_i^0 \mid i \geq 1, |V(F_i^0)| \leq n, F_i^0 \text{ connected}\}$. By assumption we know that there are graphs G and H such that (a) and (b) hold for K'. Now we recall (4), i.e.,

$$\mathrm{hom}(F, G) = \sum_{F' \leq F} \frac{1}{\mathrm{aut}(F')} \cdot \mathrm{epi}(F, F') \cdot \mathrm{emb}(F', G).$$

If F is connected, then $\mathrm{epi}(F, F') > 0$ implies that F' is connected, too. That is, the values $\mathrm{hom}(F, G)$ for $F \in \mathsf{K}$ are determined by the values of $\mathrm{emb}(F', G)$ for $F' \in \mathsf{K}'$. Therefore, (d) holds by (b). ◀

▶ **Theorem 5.2.** *The class P of planar graphs and the class $\mathsf{C}(3\text{-}col)$ of 3-colorable graphs have no query algorithm.*

Proof. Here we only present the proof for P. For a contradiction, by Lemma 5.1 we can assume that there is an emb algorithm for P that uses the finite set F of connected graphs. By induction on $n := |\mathsf{F}|$, we show that this cannot be the case. Let $|\mathsf{F}| = 1$, i.e., $\mathsf{F} = \{F\}$ for some connected F. Set $k := |V(F)| + 4$. Clearly, $K_k \notin \mathsf{P}$ and both, $\mathrm{emb}(F, F)$ and $\mathrm{emb}(F, K_k)$, are nonzero. If F is planar, we set (recall that for a graph G and $p \in \mathbb{N}$ by pG we denote the disjoint union of p copies of G),

$$G := \mathrm{emb}(F, K_k)F \quad \text{and} \quad H := \mathrm{emb}(F, F)K_k$$

By (2), $\mathrm{emb}(F, G) = \mathrm{emb}(F, K_k) \cdot \mathrm{emb}(F, F) = \mathrm{hom}(F, H)$ and $G \in \mathsf{P}$, $H \notin \mathsf{P}$, a contradiction. If F is not planar, we set $G := K_1$ and take as H a topological minor of K_k in which every edge in K_k is subdivided into $1 + |(V(F)|$ edges. Then $H \notin \mathsf{P}$ but $\mathrm{emb}(F, G) = 0 = \mathrm{emb}(F, H)$, again a contradiction.

Now assume $|\mathsf{F}| \geq 2$. If F contains no planar graphs, we essentially can proceed as in the preceding case. So assume that F contains a planar graph. Choose a "minimal" (w.r.t. $\leq$) planar graph $F \in \mathsf{F}$ and set $\mathsf{F}' := \mathsf{F} \setminus \{F\}$. By minimality, $\mathrm{emb}(F', F) = 0$ for all planar $F' \in \mathsf{F}'$. As there is no embedding from a non-planar graph to a planar graph, we have $\mathrm{emb}(F', F) = 0$ for all $F' \in \mathsf{F}'$. By induction hypothesis, there are G_0 and H_0 satisfying the desired properties with respect to F'. If $\mathrm{emb}(F, G_0) = \mathrm{emb}(F, H_0)$, then we can simply take $G := G_0$ and $H := H_0$. Otherwise, assume first that $\mathrm{emb}(F, G_0) < \mathrm{emb}(F, H_0)$. We set

$$G := \mathrm{aut}(F)G_0 \dot\cup (\mathrm{emb}(F, H_0) - \mathrm{emb}(F, G_0))F \quad \text{and} \quad H := \mathrm{aut}(F)H_0.$$

Hence, $G \in \mathsf{P}$, $H \notin \mathsf{P}$, and $\mathrm{emb}(F, G) = \mathrm{emb}(F, H)$ (by (2)). In case $\mathrm{emb}(F, G_0) > \mathrm{emb}(F, H_0)$ we argue similarly. ◀

6 No query algorithm detects isolated vertices

▶ **Theorem 6.1.** *The class $\mathsf{C}(\forall x\exists y Exy)$ of graphs without isolated vertices has no query algorithm.*

In the proof given in the full version of the paper we use some tools already used in the preceding section:

- By Lemma 5.1 it suffices to show that $\mathsf{C}(\forall x\exists y Exy)$ has no emb algorithm that only uses connected graphs.
- If C is one of the classes P or $\mathsf{C}(\forall x\exists y Exy)$, then the disjoint union $\dot{\bigcup}_{i\in I} G_i$ of a family $(G_i)_{i\in I}$ of graphs is in C if and only if each G_i is in C.
- By (2), for graphs F_1 and F_2, $p, q \geq 1$, and a connected graph G, we have

  $$\mathrm{emb}(G, pF_1 \mathbin{\dot\cup} qF_2) = p\,\mathrm{emb}(G, F_1) + q\,\mathrm{emb}(G, F_2),$$

  i.e., $\mathrm{emb}(G, pF_1 \mathbin{\dot\cup} qF_2)$ is a linear combination of $\mathrm{emb}(G, F_1)$ and $\mathrm{emb}(G, F_2)$.

Furthermore note:

- Let G be a connected graph with more than one vertex. If the graph F' is obtained from the graph F by adding a set of isolated vertices, then $\mathrm{emb}(G, F') = \mathrm{emb}(G, F)$.

Recall that in (3) we fixed the enumeration $F_1^0, F_2^0, \ldots$ of graphs that contains a copy of every isomorphism type and respects the relation $\leq$. Here we let $H_1, H_2, \ldots$ be the subsequence of $F_1^0, F_2^0, \ldots$ consisting of the connected graphs.

For $i \geq 1$ let $\alpha_i := (\mathrm{emb}(H_i, H_j))_{j\geq 1}$ be the vector containing the emb-values of H_i for connected graphs. We sketch the idea underlying the proof of Theorem 6.1. The central idea can be vaguely expressed by saying that for each $n \geq 1$ there is an $r_n \in \mathbb{N}$ such that appropriate "subvectors" of length r_n of r_n-many of the $\alpha_1, \ldots, \alpha_n$ are linearly independent vectors of the vector space $\mathbb{Q}^{r_n}$ and hence, a basis of $\mathbb{Q}^{r_n}$. In particular, every further vector of $\mathbb{Q}^{r_n}$ is a linear combination of these vectors. Furthermore, r_n tends to infinity when n increases.

For an arbitrary emb algorithm with connected graphs we must show the existence of graphs G and H, one with isolated vertices the other one without, that cannot be distinguished by this emb algorithm. We use the tools mentioned above to construct such graphs using the knowledge about the linear independence or linear dependence of some tuples of vectors obtained in the first steps of the proof.

7 Adaptive hom algorithms

For $r \geq 1$ let S_r denote the star of r vertices, i.e., a graph that consists of a vertex of degree $r-1$ (the center of the star) and $r-1$ vertices of degree 1, all neighbors of the center. For a vertex u of a graph we denote by $\deg(u)$ its degree. Note that $\deg(u) = 0$ means that u is isolated. The proof in the full paper of the following result is built on the well-known equality (see e.g., [4, 15]) obtained by looking at the value of the center of a star under a homomorphism: $\mathrm{hom}(S_r, G) = \sum_{v\in V(G)} \deg(v)^{r-1}$.

▶ **Proposition 7.1.** *Let G be a graph and $d_i := |\{u \in V(G) \mid \deg(u) = i\}|$ for $i \geq 0$. If $n := |V(G)|$, then $\mathrm{hom}(S_{n\cdot\log n}, G)$ determines $d_0, \ldots, d_{n-1}$.*

In Section 6 we showed that there is no query algorithm that decides whether a graph G is in $\mathsf{C}(\forall x\exists y Exy)$, i.e., whether $d_0 = 0$ for G. However, Proposition 7.1 shows that for a graph G with n vertices this can be decided by querying $\mathrm{hom}(S_{n\cdot\log n}, G)$. Thus, we have an algorithm for $\mathsf{C}(\forall x\exists y Exy)$ consisting of two homomorphism counts:

- query $n := \hom(F^0_1, G)$ $(= |V(G)|)$,
- query $\hom(S_{n \cdot \log n}, G)$.

That is, the selection of the graph for the second homomorphism count, in our case $S_{n\cdot \log n}$, depends on the answer to the first query. This leads to the notion of adaptive hom algorithm. Recall that $F^0_1, F^0_2, \dots$ is an enumeration of all graphs (up to isomorphism) respecting $\leq$.

▶ **Definition 7.2.** Let C be a class of graphs and $k \geq 1$. A *k adaptive hom algorithm for* C consists of a function $g : \{\emptyset\} \cup \bigcup_{i \in [k-1]} \mathbb{N}^i \to \mathbb{N}$ and a subset X of $\mathbb{N}^k$ such that for all G,

$$G \in \mathsf{C} \iff (n_1, \dots, n_k) \in X,$$

where $n_1 := \hom(F^0_{g(\emptyset)}, G)$, $n_2 := \hom(F^0_{g(n_1)}, G)$, $\dots$, $n_k := \hom(F^0_{g(n_1,n_2,\dots,n_{k-1})}, G)$. We then say that C *can be decided by a k adaptive hom algorithm.*

The main result of this section:

▶ **Theorem 7.3.** *Every class C can be decided by a 3 adaptive hom algorithm.*

To get this result we show:

▶ **Theorem 7.4.** *For $n \geq 1$ there exist graphs F_1 $(= F_1(n))$ and F_2 $(= F_2(n))$ such that for all graphs G and H with n vertices,*

$$\hom(F_1, G) = \hom(F_1, H) \quad \textit{and} \quad \hom(F_2, G) = \hom(F_2, H) \quad \textit{imply} \quad G \cong H.$$

In fact, by this result for any class C of graphs, we get the 3 adaptive hom algorithm that for a graph G queries

- $\hom(F^0_1, G)$; set $n := \hom(F^0_1, G)$
- $\hom(F_1(n), G)$ and $\hom(F_2(n), G)$

(where $F_1(n)$ and $F_2(n)$ are the graphs of Theorem 7.4) and has as set X the set

$$X := \{(n, \hom(F_1(n), H), \hom(F_2(n), H)) \mid n \geq 1,\ H \in \mathsf{C}, \text{ and } |V(H)| = n\}.$$

▶ **Corollary 7.5.** *For graphs G and H, if $n_0 := \hom(F^0_1, G) = \hom(F^0_1, H)$, $\hom(F_1(n_0), G) = \hom(F_1(n_0), H)$, and $\hom(F_2(n_0), G) = \hom(F_2(n_0), H)$, then $G \cong H$.*

Hence, by "3 adaptive hom counts" we can characterize the isomorphism type of any graph. It is not possible to do this by two queries, more precisely (for a proof see the full paper):

▶ **Theorem 7.6.** *There is no $s_0 \in \mathbb{N}$ such that for some function $g : \mathbb{N} \to \mathbb{N}$ and all graphs G and H, if $n_0 := \hom(F^0_{s_0}, G) = \hom(F^0_{s_0}, H)$ and $\hom(F^0_{g(n_0)}, G) = \hom(F^0_{g(n_0)}, H)$, then $G \cong H$.*

We turn to a proof of Theorem 7.4. An important tool will be the following lemma.

▶ **Lemma 7.7.** *Let $n \geq 1$ and K be a finite set of graphs. We can construct a graph F_{K} such that for all G and H with exactly n vertices we have $\hom(F_{\mathsf{K}}, G) = \hom(F_{\mathsf{K}}, H)$ if and only if G and H satisfy at least one of the conditions (a) and (b).*

(a) *There exist $F, F' \in \mathsf{K}$ such that $\hom(F, G) = 0$ and $\hom(F', H) = 0$.*

(b) *For all $F \in \mathsf{K}$, $\hom(F, G) = \hom(F, H)$.*

Proof. The idea of the construction is best seen by assuming $\mathsf{K} = \{F_1, F_2\}$ (iterating the following process one gets the general case). We set $r := n^{|V(F_1)|}$. As $|V(G)| = n$, we know that $\hom(F_1, G) \le r$. We set $F_{\mathsf{K}} := F_1 \dot\cup rF_2$. By (1) for every graph F,

$$\hom(F_{\mathsf{K}}, F) = \hom(F_1, F) \cdot \hom(F_2, F)^r. \tag{5}$$

Hence, if (a) or (b) hold, then $\hom(F_{\mathsf{K}}, G) = \hom(F_{\mathsf{K}}, H)$. Conversely, assume $z := \hom(F_{\mathsf{K}}, G) = \hom(F_{\mathsf{K}}, H)$. If $z = 0$, then (a) must hold by (5). If $z = 1$, then $\hom(F_i, G) = \hom(F_i, H) = 1$ for $i \in [2]$ and (b) holds. Otherwise, $z \ge 2$. Let $F := G$ or $F := H$ and set $x := \hom(F_1, F)$ and $y := \hom(F_2, F)$. Let p be a prime number with $p|z$ (i.e., p divides z). Choose the maximum k such that $p^k|z$. Write k in the form $k = \ell \cdot r + m$ with $0 \le m < r$. As $z = x \cdot y^r$ and $x \le r$, the factor p^m appears in x and the factor p^ℓ in y. This determines x and y; they do not depend on the $F \in \{G, H\}$ chosen. ◀

If the alternative (b) holds, we can replace in a hom algorithm the set K of graphs by the single graph F_{K}. The previous lemma doesn't help too much if the alternative (a) holds. To overcome this alternative essentially we consider the bipartite graphs and the non-bipartite graphs separately. For this purpose we recall the definition and some simple facts on bipartite graphs.

A graph G is *bipartite* if there is a partition $V(G) = X \dot\cup Y$ such that each edge has one end in X and one end in Y. The next lemma contains some simple facts on bipartite graphs.

▶ **Lemma 7.8.** *Let G be a graph. Then:*

(a) *G is bipartite $\iff$ $\hom(G, P_2) \ne 0$.*

(b) *If G is connected and bipartite, then $\hom(G, P_2) = 2$.*

(c) *G is bipartite $\iff$ $\hom(G, H) \ne 0$ for all graphs H with at least one edge.*

(d) *G is bipartite $\iff$ G does not contain a cycle of odd length.*

(e) *If G is bipartite and F is not, then $\hom(F, G) = 0$.*

(f) *If G is bipartite, then G is determined (up to isomorphism) by the values $\hom(F, G)$ for the bipartite graphs F with $F \le G$ (by the Lovász Isomorphism Theorem and part (e)).*

For graphs G and H the *product $G \times H$ of G and H* is the graph with $V(G \times H) := \{(u, v) \mid u \in V(G),\ v \in V(H)\}$ and $E(G \times H) := \{\{(u, v), (u', v')\} \mid \{u, u'\} \in E(G) \text{ and } \{v, v'\} \in E(H)\}$. One easily verifies that for any graph H,

$$\hom(F, G \times H) = \hom(F, G) \cdot \hom(F, H). \tag{6}$$

Besides the simple facts on bipartite graphs mentioned above, we also need a deep result:

▶ **Theorem 7.9** (Lovász Cancellation Law [17]). *A graph H is not bipartite if and only if $F \times H \cong G \times H$ implies $F \cong G$ for all graphs F and G.*

The following lemma contains a further step for the proof of Theorem 7.4.

▶ **Lemma 7.10.** *Let $n \ge 2$ and t be the smallest natural number with $n \le 2t$. We set (recall that C_m denotes a cycle of length m)*

$$\mathsf{K}_t := \{F \mid \hom(F, C_{2t+1}) > 0 \textit{ and } |V(F)| \le (2t+1)^2\}. \tag{7}$$

For graphs G and H with $|V(G)| = |V(H)| = n$, if $\hom(F, G) = \hom(F, H)$ for all $F \in \mathsf{K}_t$, then $G \cong H$.

Proof. Assume $G \ncong H$. By Lemma 7.8 (d), C_{2t+1} is not bipartite. Thus, by the Lovász Cancellation Law $G \times C_{2t+1} \ncong H \times C_{2t+1}$ As $G \times C_{2t+1}$ has $n \cdot (2t+1)$ vertices, by the Lovász Isomorphism Theorem (see Theorem 2.2) there is a graph F with $|V(F)| \leq n \cdot (2t+1) \leq (2t+1)^2$ such that

$$\hom(F, G \times C_{2t+1}) \neq \hom(F, H \times C_{2t+1}). \tag{8}$$

If F is not in K_t, then $\hom(F, C_{2t+1}) = 0$ and thus by (6) we get a contradiction to (8). Hence, $F \in \mathsf{K}_t$ and so by assumption, $\hom(F, G) = \hom(F, H)$. However, this contradicts (8) (use again (6)). ◀

Proof of Theorem 7.4. The case $n = 1$ is trivial. Assume $n \geq 2$. Let again t be the smallest natural number with $n \leq 2t$ and K_t be defined by (7). For the class K_{bip} of bipartite graphs in K_t let F_1 be the graph constructed in Lemma 7.7 for K_{bip} (i.e., $F_1 = F_{\mathsf{K}_{\text{bip}}}$). As the disjoint union of bipartite graphs is bipartite, the proof of Lemma 7.7 shows that F_1 is bipartite. Let F_2 be the graph constructed in Lemma 7.7 for the class $\mathsf{K}_t \setminus \mathsf{K}_{\text{bip}}$. We show that for graphs G and H with $|V(G)| = |V(H)| = n$,

$$\hom(F_i, G) = \hom(F_i, H) \text{ for } i \in [2] \text{ implies } G \cong H.$$

If G and H both have no edge, then clearly $G \cong H$. If, say G has an edge but $E(H) = \emptyset$, then, as F_1 is bipartite, $\hom(F_1, G) \neq 0 = \hom(F_1, H)$ by Lemma 7.8 (c) and $E(F_1) \neq \emptyset$. Hence we can assume that both graphs contain at least one edge. For a contradiction assume that $\hom(F_i, G) = \hom(F_i, H)$ for $i \in [2]$ and that $G \ncong H$. Then, by Lemma 7.10 there is a graph $F_0 \in \mathsf{K}_t$ with

$$\hom(F_0, G) \neq \hom(F_0, H). \tag{9}$$

Assume first that $F_0 \in \mathsf{K}_{\text{bip}}$. As G and H contain at least one edge, by Lemma 7.8 (c)

$$\hom(F, G) > 0 \quad \text{and} \quad \hom(F, H) > 0,$$

for every bipartite graph F. In particular, this holds for all graphs F in K_{bip}. Thus, $\hom(F_1, G) = \hom(F_1, H)$ implies the second case in Lemma 7.7, i.e., for every $F \in \mathsf{K}_{\text{bip}}$,

$$\hom(F, G) = \hom(F, H).$$

In particular, this holds for $F = F_0$ contradicting (9).

Thus $F_0 \in \mathsf{K}_t \setminus \mathsf{K}_{\text{bip}}$. Then $\hom(F_0, F) = 0$ for all bipartite graphs F $\big($by Lemma 7.8 (e)$\big)$. Hence, by (9) at least one of G and H must be non-bipartite. By $\hom(F_2, G) = \hom(F_2, H)$, Lemma 7.7, and (9) there exist graphs F^G and F^H in $\mathsf{K}_t \setminus \mathsf{K}_{\text{bip}}$ with

$$\hom(F^G, G) = \hom(F^H, H) = 0.$$

W.l.o.g. suppose that G is not bipartite and thus contains an odd cycle, say of length ℓ. Since $\ell \leq n \leq 2t+1$, we have $\hom(C_{2t+1}, C_\ell) > 0$. In fact, one easily verifies that $\hom(C_k, C_m) > 0$ for odd m and k with $m < k$. As $F^G \in \mathsf{K}_t$, $\hom(F^G, C_{2t+1}) > 0$. Therefore, $\hom(F^G, C_\ell) > 0$, which implies $\hom(F^G, G) > 0$, a contradiction. ◀

8 Right hom algorithms

In 1993 Chaudhuri and Vardi [6] (see also [1]) showed the analogue of the Lovász Isomorphism Theorem for the right homomorphism vector of a graph G. More precisely, the vector of values $\hom(G, F)$, where F ranges over all graphs, characterizes the isomorphism type of G. In this section we will see that our main "positive" results on hom algorithms fail for right hom algorithms (see Proposition 8.2) and that various results that survive use a completely different proof technique. Note that a graph G is 3-colorable if $\hom(G, K_3) > 0$ in contrast to Theorem 5.2.

▶ **Definition 8.1.** A class C of graphs can be *decided by a right hom algorithm* if and only if there is a $k \geq 1$ and graphs $F_1, \ldots, F_k$ such that for all graphs G and H,

$$\hom(G, F_1) = \hom(G, F_k), \ldots, \hom(G, F_k) = \hom(G, F_k) \text{ imply } (G \in \mathsf{C} \iff H \in \mathsf{C}),$$

or equivalently, if in addition to $F_1, \ldots, F_k$ there is a set $X \subseteq \mathbb{N}^k$ such that for any graph G, $\big(G \in \mathsf{C} \iff (\hom(G, F_1), \ldots, \hom(G, F_k)) \in X\big)$.

Again one can show that C is decidable if and only if as X can be chosen a decidable set.

It should be clear how we define k *adaptive right hom algorithms* for a class C of graphs.

The failure in the "right world" of the "positive" results Theorem 4.4 and Theorem 7.3 is shown by (it should be clear how we define k *adaptive right hom algorithms* for a class C of graphs):

▶ **Proposition 8.2.** *let $k \geq 1$. The class $\mathsf{K}(3)$ of graphs containing a clique of size 3 (expressible by the existential sentence $\exists x \exists y \exists z(Exy \wedge Eyz \wedge Exz)$) cannot be decided by a k adaptive right hom algorithm (and hence not by a right hom algorithm).*

Proof. For a graph G we denote by $\chi(G)$ the chromatic number of G, i.e., the least s such that G is s-colorable. Clearly, $\big(m < \chi(G) \iff \hom(G, K_m) = 0\big)$ for the clique K_m with m elements, and hence, for every graph F,

$$\text{if } |V(F)| < \chi(G), \text{ then } \hom(G, F) = 0. \tag{10}$$

For a contradiction, assume that g and X (compare Definition 7.2) witness the existence of a k adaptive right hom algorithm for C. Then set

$$n_1 := g(\emptyset),\ n_2 := g(0),\ n_3 := g(0,0), \ldots, n_k := g(\underbrace{0, \ldots, 0}_{k-1 \text{ times}}).$$

Let $s > 3$ be bigger than any of the $|V(F^0_{n_i})|$'s. According to [19] there is a $G_0 \notin \mathsf{K}(3)$ such that $\chi(G_0) = s$. Thus by (10), we have $\hom(G_0, F^0_{n_1}) = 0, \ldots, \hom(G_0, F^0_{n_k}) = 0$ and hence, $(0, 0, \ldots, 0) \notin X$. However, $K_s \in \mathsf{K}(3)$ and $\hom(K_s, F^0_{n_1}) = 0, \ldots, \hom(K_s, F^0_{n_k}) = 0$, thus $(0, 0, \ldots, 0) \in X$, a contradiction. ◀

Note that Theorem 10 in [1] cannot be applied to show (directly) the existence of an FO-sentence φ with no right hom algorithm. To apply this theorem to such a φ we should have for all graphs G and H, $(G \models \varphi \iff H \models \varphi)$ implies $|V(G)| = |V(H)|$. One easily verifies that this condition is not satisfied by any FO-sentence φ.

We mention a positive result on right hom algorithms (proven in the full version of the paper).

▶ **Theorem 8.3.** *Every class* C *of graphs with the property that there is a bound on the number of edges of graphs in* C *has a right hom algorithm.*

Finally we remark that some results with nontrivial proofs for hom algorithms are trivial for right hom algorithms. For example, the following simple proof shows that there is no right hom algorithm for the class $\mathsf{C}(\forall x \exists y Exy)$ of graphs with no isolated vertices.

Let $F_1, \dots, F_k$ be any finite set of graphs and set $m := 1 + \max\{|V(F_i)| \mid i \in [k]\}$. Then, $\hom(K_m, F_i) = 0 = \hom(K_m \dot\cup K_1)$ for every $i \in [k]$. Thus there is no right hom algorithm that detects an isolated vertex.

References

1 A. Atserias, P. Kolaitis, and W. 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. IEEE, 2021.

2 J. Böker. Graph similarity and homomorphism densities. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 32:1–32:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

3 J. Böker, Y. Chen, M. Grohe, and G. Rattan. The complexity of homomorphism indistinguishability. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, pages 54:1–54:13, 2019.

4 C. Borgs, J. Chayes, L. Lovász, V.T. Sós, and K. Vesztergombi. Counting graph homomorphisms. In *Topics in Discrete Mathematics*, pages 315–371, 2006.

5 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

6 S. Chaudhuri and M. Y. Vardi. Optimization of *Real* conjunctive queries. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1993*, pages 59–70. ACM Press, 1993.

7 Y. Chen and J. Flum. Tree-depth, quantifier elimination, and quantifier rank. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 225–234, 2018.

8 Y. Chen and J. Flum. FO-definability of shrub-depth. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, pages 15:1–15:16, 2020.

9 R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 210–223. ACM, 2017.

10 H. Dell, M. Grohe, and G. Rattan. Lovász meets Weisfeiler and Leman. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

11 G. Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992.

12 Z. Dvorák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010.

13 R. Ganian, P. Hlinený, J. Nesetril, J. Obdrzálek, P. Ossona de Mendez, and R. Ramadurai. When trees grow low: Shrubs and fast MSO_1. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, pages 419–430, 2012.

14 M. Grohe. Counting bounded tree depth homomorphisms. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, 2020*, pages 507–520. ACM, 2020.

15 M. 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 2020*, pages 1–16. ACM, 2020.

16 L. Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18:321–328, 1967.

17 L. Lovász. On the cancellation law among finite relational structures. *Periodica Mathematica Hungarica*, 1:145–156, 1971.

18 T. A. McKee. Forbidden subgraphs in terms of forbidden quantifiers. *Notre Dame Journal of Formal Log.*, 19:186–188, 1978.

19 J. Mycielski. Sur le coloriage des graphes. *Information Processing Letters*, 108(6):412–417, 2008.

Higher-Order Quantified Boolean Satisfiability

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, Coventry, UK

Christoph Haase
Department of Computer Science, University of Oxford, Oxford, UK

Zahra Hadizadeh
Sharif University of Technology, Tehran, Iran

Alessio Mansutti
Department of Computer Science, University of Oxford, Oxford, UK

Abstract

The Boolean satisfiability problem plays a central role in computational complexity and is often used as a starting point for showing NP lower bounds. Generalisations such as Succinct SAT, where a Boolean formula is succinctly represented as a Boolean circuit, have been studied in the literature in order to lift the Boolean satisfiability problem to higher complexity classes such as NEXP. While, in theory, iterating this approach yields complete problems for k-NEXP for all $k > 0$, using such iterations of Succinct SAT is at best tedious when it comes to proving lower bounds.

The main contribution of this paper is to show that the Boolean satisfiability problem has another canonical generalisation in terms of higher-order Boolean functions that is arguably more suitable for showing lower bounds beyond NP. We introduce a family of problems HOSAT(k, d), $k \geq 0$, $d \geq 1$, in which variables are interpreted as Boolean functions of order at most k and there are d quantifier alternations between functions of order exactly k. We show that the unbounded HOSAT problem is TOWER-complete, and that HOSAT(k, d) is complete for the weak k-EXP hierarchy with d alternations for fixed $k, d \geq 1$ and d odd.

We illustrate the usefulness of HOSAT by characterising the complexity of weak Presburger arithmetic, the first-order theory of the integers with addition and equality but without order. It has been a long-standing open problem whether weak Presburger arithmetic has the same complexity as standard Presburger arithmetic. We answer this question affirmatively, even for the negation-free fragment and the Horn fragment of weak Presburger arithmetic.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Boolean satisfiability problem, higher-order Boolean functions, weak k-EXP hierarchies, non-elementary complexity, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.33

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).

Acknowledgements We would like to thank the anonymous reviewers for their thoughtful comments, and in particular for pointing us to [22], which uncovered profound connections between our work and the basic language of set theory and typed λ-calculi studied by Statman [30].

1 Introduction

The Boolean satisfiability problem (SAT) plays a central role in computational complexity. It was the first problem shown to be NP-complete [8] and has ever since been used a countless number of times to show NP lower bounds for numerous combinatorial problems, a prime example being Karp's list of twenty-one NP-complete problems [17]. What makes SAT stand

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 33; pp. 33:1–33:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

out is its simplicity: describing an instance of SAT does not require tapes and automata as in the case of Turing machines, and neither grids nor dominoes like in tiling problems; it also does not require the introduction of graphs and their properties as in, e.g., graph colouring problems. To obtain a reduction from SAT, it suffices to show how to encode assignments to Boolean variables and how to model connectives. Unfortunately, this simplicity does not transfer over to higher complexity classes.

Succinct representations have been used as a technical tool to lift NP-complete problems to exponential classes [27, 20, 2, 32]. The idea is that the main object of interest is succinctly encoded by a Boolean circuit. For instance, a Boolean formula represented as a DAG whose vertices are labelled by a type (e.g., a Boolean connective or constant value) and a unique integer index can be encoded by a Boolean circuit with outputs. Given a binary encoding of an index, this circuit returns the type of the node with that index and, if present, the indices of its two predecessor nodes. Succinct variants of P- or NP-complete problems become EXP- or NEXP-complete, e.g., the Succinct Circuit Value problem and the Succinct SAT Problem are respectively EXP- and NEXP-complete [26]. In theory, this approach can be iterated indefinitely, and, e.g., the problem of deciding whether a propositional formula encoded by a Boolean circuit that is itself encoded by a Boolean circuit evaluates to true is a "canonical" 2-EXP-complete problem. One does not need too much imagination to see that these iterated succinct problems become unbearable for showing lower bounds.

This loss of simplicity for higher complexity classes compared to classical SAT is at odds with other canonical problems, such as the halting problem for a (non)deterministic Turing machine running in time $f(n)$ on an input of length n, or the problem of tiling a grid of size $f(n) \times f(n)$. Here, the function f can be chosen appropriately in order to obtain hard problems for many complexity classes such as k-EXP and k-NEXP [26]. The same picture emerges in the presence of alternation: whereas QBF [12, Chapter 7.4] elegantly extends SAT to the whole polynomial hierarchy [33], no simple extension of SAT has been defined to capture every level of the weak k-EXP hierarchies, for any $k \geq 1$ (cf. [15]). Again, this is in contrast with Turing machines and tiling problems: both admit extensions to deal with alternation in the context of complexity classes above NP [4, 5, 23].

The main contribution of this paper is to identify a canonical generalisation of SAT, called HOSAT, that gives complete problems for all weak k-EXP hierarchies and that shares the simplicity of classical SAT. Instead of Boolean variables, the building blocks of HOSAT are *function applications*. The functions considered are higher-order Boolean function f of order k, i.e., functions that take as input higher-order functions of order $k-1$ and return a Boolean value, and Boolean values themselves are functions of order zero. HOSAT closes function applications under Boolean connectives as well as quantification.

The development of HOSAT is a result of an attempt of the authors to settle the open problem of the computational complexity of *weak Presburger arithmetic*, the first-order theory of the integers with addition and equality, but without an order predicate (which is provably not definable in this theory). This less expressive theory is in fact the "original" arithmetic theory studied by Presburger in his seminal paper [28], see also [7], and it has been an open problem whether it is computationally as hard as what is nowadays commonly understood as Presburger arithmetic; see, e.g., [6]. The lower bound for Presburger arithmetic given by Berman [3] reduces from an alternating Turing machine running in doubly exponential time with a linear number of alternations. This reduction glosses over some technical details, which is unproblematic in the presence of the sufficient expressive power that the order predicate provides, but becomes problematic for weak Presburger arithmetic. Giving a clean reduction from HOSAT enables us to settle the complexity of weak Presburger arithmetic and to show that it has indeed the same complexity as Presburger arithmetic with the order predicate. We hope that HOSAT will be as beneficial as it was for us for other researchers for proving their lower bounds, in particular since the question of finding canonical complete problems for weak k-EXP hierarchies frequently comes up; see, e.g., [31, 21].

2 Preliminaries

We write $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_+$ to denote the set of integers, natural numbers including zero, and natural numbers without zero, respectively. Unless otherwise stated, we assume integers to be encoded in binary. Given $l, u \in \mathbb{N}$, we define $[l, u] := \{l, l+1, \dots, u\}$, $[l, u) := [l, u-1]$ and $[u] := [0, u)$. The cardinality of a finite set A is denoted by $\#A$. For $k, n \in \mathbb{N}$, we write $\exp_2^k(n)$ for the tetration function inductively defined as $\exp_2^0(n) := n$ and $\exp_2^k(n) := 2^{\exp_2^{k-1}(n)}$. Intuitively, $\exp_2^k(n)$ is a tower of exponentials of height k, base 2, and top-most exponent n.

We recall complexity classes based on the notion of alternating Turing machines [5]. The class $\Sigma_d^{k\text{-EXP}}$ contains all problems that can be decided by an alternating Turing machine running in time $\exp_2^k(f(n))$ on an input of length n, where $f : \mathbb{N} \to \mathbb{N}$ is some polynomial that does not depend on the input, and starting in an existential state and making at most $d-1$ alternations on every computation path. By definition, $\Sigma_1^{k\text{-EXP}} = k\text{-NEXP}$. The *weak* k-EXP *hierarchy* is defined as $\bigcup_{d \geq 0} \Sigma_d^{k\text{-EXP}}$; for $k = 1$ see [15].

Given functions $a, s, t \colon \mathbb{N} \to \mathbb{N}$, the class $\mathrm{STA}(s(n), t(n), a(n))$ contains all problems that can be decided by an alternating Turing machine in time $t(n)$ using space $s(n)$ making at most $a(n)$ alternations on every computation path, where n is the length of the input. We use $*$ to indicate an unbounded availability of a certain resource. For instance, the polynomial hierarchy can be characterized as $\bigcup_{d \in \mathbb{N}} \mathrm{STA}(*, n^{O(1)}, d)$, and the d-th level of the weak k-EXP hierarchy $\Sigma_d^{k\,\text{EXP}}$ corresponds to $\mathrm{STA}(*, \exp_2^k(n^{O(1)}), d)$. The STA complexity measure was introduced by Berman [3] to show the following result.

▶ **Theorem 1** ([3]). *Presburger arithmetic is complete for* $\mathrm{STA}(*, 2^{2^{n^{O(1)}}}, O(n))$.

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *elementary* if there is a $k \in \mathbb{N}$ such that $f(n) < \exp_2^k(n)$ for all $n \in \mathbb{N}$. The (non-elementary) class TOWER [29] contains all problems decidable by a Turing machine in time $\exp_2^{g(n)}(f(n))$ for some fixed polynomial f and elementary function g, on inputs of length n. We have $\cup_{k \geq 1} k\text{-EXP} \subsetneq \text{TOWER}$, as TOWER contains problems decidable in time $\exp_2^n(1)$.

3 The higher-order quantified satisfiability problem

We introduce the *higher-order quantified satisfiability problem*, a problem whose instances form a hierarchy $(\mathrm{HOSAT}(k, d))_{k,d}$ of problems that characterise every complexity class $\Sigma_d^{k\text{-EXP}}$.

We write $\mathbb{B}$ to denote the Boolean domain $\{0, 1\}$. We often treat $\mathbb{B}$ as the set of the two nullary Boolean functions $() \to \{0\}$ and $() \to \{1\}$. Fix $n \in \mathbb{N}_+$ and let $\mathbb{B}_{0,n} := \mathbb{B}$. Given $k \geq 1$, we write $\mathbb{B}_{k,n}$ for the set of all functions with domain $(\mathbb{B}_{k-1,n})^n$ and codomain $\mathbb{B}$. For instance, $\mathbb{B}_{1,n}$ is the set of all n-ary Boolean functions $f \colon \{0,1\}^n \to \{0,1\}$, whereas $\mathbb{B}_{2,n}$ is the set of all n-ary second-order Boolean functions $f \colon (\{0,1\}^n \to \{0,1\})^n \to \{0,1\}$. For $k \geq 1$, the number of functions in $\mathbb{B}_{k,n}$ satisfies the recurrence relation $\#(\mathbb{B}_{k,n}) = 2^{\#(\mathbb{B}_{k-1,n})^n}$, and therefore $\exp_2^{k+1}(n) \leq \#\mathbb{B}_{k,n} \leq \exp_2^{k+1}((k+1) \cdot n)$. We refer to the indices k and n as the *order* and the *arity* of $\mathbb{B}_{k,n}$, respectively. Both n and k are written in unary.

We discuss the encoding $enc(f)$ of Boolean functions as bit-strings over $\{0, 1\}$, which we often treat as the binary number from $\mathbb{N}$ represented by the bit-string, with least significant digit first. For $b \in \mathbb{B}$, $enc(b) := b$. Functions $f \in \mathbb{B}_{1,n}$ can be encoded as a string of length

2^n over $\mathbb{B}$, whose i-th position encodes the truth value of f on the tuple $(b_1,\dots,b_n) \in \mathbb{B}^n$ such that $b_1 \cdots b_n$ is the binary encoding of i, with least significant digit first. For $k > 1$, functions $f \in \mathbb{B}_{k,n}$ can be encoded as a string over $\mathbb{B}$ whose i-th position encodes the truth value of f on input $(g_1,\dots,g_n) \in (\mathbb{B}_{k-1,n})^n$ such that the concatenation $\mathit{enc}(g_1)\cdots\mathit{enc}(g_n)$ of the encodings of $g_1,\dots,g_n$ is a binary encoding of i, with least significant digit first. The length $|f|$ of a function f from $\mathbb{B}_{k,n}$ is defined as the length of $\mathit{enc}(f)$. Therefore, $|b| = 1$ for all $b \in \mathbb{B}$, and for every $f \in \mathbb{B}_{k,n}$ with $k \geq 1$, we have $|f| = (\#(\mathbb{B}_{k-1,n}))^n \geq (\exp_2^k(n))^n$.

A *quantifier-free generalised Boolean formula* is a formula from the following grammar

$$\Phi ::= \top \mid f(g_1,\dots,g_n) \mid \neg\Phi \mid \Phi \wedge \Phi$$

where $f(g_1,\dots,g_n)$ is said to be a *function application*, and each f, $g_1,\dots,g_n$ are *function symbols* taken from an infinite alphabet Σ. Each function symbol in Σ is implicitly endowed with a type $\mathbb{B}_{k,n}$, with $k,n \in \mathbb{N}$. A (quantifier-free) generalised Boolean formula is said to be *well-formed* whenever every function application is consistent with the type of the function symbol, i.e., a function application $f(g_1,\dots,g_n)$ requires f to be of arity n, if $f \in \mathbb{B}$ then $n = 0$, and otherwise $f \in \mathbb{B}_{k,n}$ for some $k \geq 1$ and every g_i with $i \in [1,n]$ belongs to $\mathbb{B}_{k-1,n}$.

Given a vector-variable of function symbols $\boldsymbol{f} = (f_1,\dots,f_m)$ of type $\mathbb{B}_{k,n}$, we write $\exists \boldsymbol{f} \colon \mathbb{B}_{k,n}$ as a shorthand for the *existential quantifier block* $\exists f_1 \colon \mathbb{B}_{k,n} \ \dots\ \exists f_m \colon \mathbb{B}_{k,n}$; the *universal quantifier block* $\forall \boldsymbol{f} \colon \mathbb{B}_{k,n}$ stands for $\forall f_1 \colon \mathbb{B}_{k,n} \ \dots\ \forall f_m \colon \mathbb{B}_{k,n}$. The set of all *generalised Boolean formulae of order* 0 and alternation depth d is the set of all formulae $\exists \boldsymbol{b}_1 \colon \mathbb{B}\ \forall \boldsymbol{b}_2 \colon \mathbb{B}\ \dots\ \exists \boldsymbol{b}_d \colon \mathbb{B}\,.\,\Phi$, where Φ is a quantifier-free generalised Boolean formula. *Generalised Boolean formulae of order* $k \geq 1$ and alternation depth d are formulae

$$\exists \boldsymbol{f}_1 \colon \mathbb{B}_{k,n}\ \forall \boldsymbol{f}_2 \colon \mathbb{B}_{k,n}\ \dots\ \exists \boldsymbol{f}_d \colon \mathbb{B}_{k,n}\,.\,\Phi,$$

where the arity n is arbitrary and Φ is a generalised Boolean formula of order $k-1$, arbitrary alternation depth, and same arity n. The semantics of generalised Boolean formulae is as expected, e.g., $\exists f \colon \mathbb{B}_{k,n}\ \Psi$ states that there is a function $f \in \mathbb{B}_{k,n}$ that makes Ψ true. We write $\Phi \equiv \Psi$ to denote that Φ and Ψ are *equivalent*.

The size $|\Phi|$ of a generalised Boolean formula Φ is the number of symbols required to write it down, where "$\colon \mathbb{B}_{k,n}$" is a lexeme that decorates the function symbols, providing their type, and should not be confused with the actual set $\mathbb{B}_{k,n}$. We write $\mathrm{fv}(\Phi)$ for the set of *free* function symbols of Φ, i.e., the set of those function symbols that do not appear in the scope of a quantifier. A *sentence* is a well-formed generalised Boolean formula Φ where all function symbols are quantified, i.e. $\mathrm{fv}(\Phi) = \emptyset$. We sometimes write $\Phi(f_1,\dots,f_m)$ or $\Phi(\boldsymbol{f})$, with $\boldsymbol{f} = (f_1,\dots,f_m)$, for a formula Φ with $\mathrm{fv}(\Phi) = \{f_1,\dots,f_m\}$. Given function symbols $g_1,\dots,g_m$ and a formula $\Phi(f_1,\dots,f_m)$, we write $\Phi(g_1,\dots,g_m)$ for the formula obtained from Φ by replacing each f_i with g_i.

Generalised Boolean formulae are formulae in prenex normal form in which the type $\mathbb{B}_{k,n}$ of Boolean functions of the quantifier prefix weakly decreases with respect to k. For presentational convenience, throughout the remainder of the paper we relax these constraints and consider formulae that are not in prenex formal form. This is done w.l.o.g., as standard ways of efficiently translating formulae in prenex normal form also work for generalised Boolean formulae. Moreover, we use standard Boolean connectives $\vee$, $\rightarrow$ and $\leftrightarrow$, and $\bot$.

Let k and d in $\mathbb{N}_+$. We introduce the problem $\mathrm{HOSAT}(k,d)$:

$\mathrm{HOSAT}(k,d)$: d-ALTERNATING SATISFIABILITY PROBLEM OF ORDER k

INPUT: A sentence Φ of order k and alternation depth d.
QUESTION: Is Φ valid?

We define $\mathrm{HOSAT}(k,*) := \bigcup_{d \in \mathbb{N}_+} \mathrm{HOSAT}(k,d)$, i.e., the problem of deciding the validity of a sentence of order k and arbitrary alternation depth; and $\mathrm{HOSAT} := \bigcup_{k \in \mathbb{N}_+} \mathrm{HOSAT}(k,*)$.

▶ **Theorem 2.** *(I)* $\mathrm{HOSAT}(k,d)$ *is complete for* $\Sigma_d^{k\text{-EXP}}$ *for odd* d.
(II) $\mathrm{HOSAT}(k,*)$ *is complete for* $\mathrm{STA}(*, \exp_2^k(n^{O(1)}), O(n))$.

To keep the presentation simple, we only formulate the result for odd d. The reduction we use to show the lower bound of Theorem 2(I) is uniform for all $k \geq 1$. By uniform polynomial time reduction [29], this implies that HOSAT is TOWER-complete.

Related work. In view of the fact that HOSAT is a natural generalisation of SAT, it comes with no surprise that some of its instances have previously been defined and have found diverse application in the past. In [1], Babai, Fortnow and Lund relied on $\mathrm{HOSAT}(1,1)$ (called *Oracle-3-satisfiability* in the paper) to show that NEXP $\subseteq$ MIP, where MIP is the class of all languages with multiple-prover interactive proof systems. In [19], Lohrey relied on $\mathrm{HOSAT}(1,d)$ (called $\mathrm{QO}\Sigma_d$-SAT in the paper) to show Σ_d^{EXP}-hardness of model checking Σ_d-MSO sentences over hierarchical graph unfoldings. These works already hint at how considering Boolean functions instead of succinct circuits is already beneficial, in terms of directness of the reductions, for NEXP-hard problems.

Above Σ_d^{EXP}, closely related is the work of Statman on the typed λ-calculus. In [30], he considers the λ-calculus with single ground type $\mathbf{0}$, no constants, only power types ($\to$) and β-conversion, and shows that checking whether two λ-terms of the calculus reduce to the same normal form is non-elementary recursive (in fact, it shows that the problem is TOWER-complete). The proof of TOWER-hardness follows thanks to a Church encoding of the basic language of set theory denoted by Statman with Ω. The variables in formulae from Ω are associated with a number type from $\mathbb{N}$. A variable of type n ranges over $\mathcal{D}_n$ where $\mathcal{D}_0 := \{\mathbf{0}, \mathbf{1}\}$, with $\mathbf{0}$ and $\mathbf{1}$ constants, and $\mathcal{D}_{n+1}$ is the powerset of $\mathcal{D}_n$. Formulae of Ω are obtained by taking the closure under Boolean connectives and quantification of membership queries of the form $\mathbf{0} \in x$, $\mathbf{1} \in x$ and $y \in z$, where x is of type 1 and y and z are of types n and $n+1$, respectively, for some $n \in \mathbb{N}$. While HOSAT and the satisfiability of formulae from Ω are essentially the same problem, the logic Ω is not suitable to capture any of the levels of the weak k-EXP hierarchies. To see this, fix $k \geq 1$ and let $\Omega(k)$ be the subset of the formulae in Ω having variables of type at most k. Differently from $\mathrm{HOSAT}(k,*)$, the satisfiability problem of $\Omega(k)$ can be shown to be in PSPACE (more precisely, it is equivalent to QBF) since the sets $\mathcal{D}_j$ with $j \leq k$ are now fixed a priori. Here, the reason why $\mathrm{HOSAT}(k,*)$ is instead k-NEXP-hard is because the sets $\mathbb{B}_{j,n}$ with $j \leq k$ still have some degree of freedom given by the unbounded number of choices for $n \in \mathbb{N}$.

According to [30], TOWER-completeness of the satisfiability problem of Ω was announced by Meyer in [24, Theorem 1(7)] as part of a forthcoming paper coauthored with Fischer. To the best of our knowledge, the latter paper was never published. To resolve this issue, in [22] Mairson gives a revision of [30] that provides a standalone proof of the TOWER-hardness of Ω and a simplification to the aforementioned Church encoding.

4 The complexity of $\mathrm{HOSAT}(k,d)$

We prove Theorem 2(I). The proof of the $\Sigma_d^{k\text{-EXP}}$ upper bound is quite simple: given a sentence $\exists \boldsymbol{f}_1 \colon \mathbb{B}_{k,n}\ \forall \boldsymbol{f}_2 \colon \mathbb{B}_{k,n} \cdots \exists \boldsymbol{f}_d \colon \mathbb{B}_{k,n} \,.\, \Phi$, where Φ is a generalised Boolean formula of order $k-1$ and arbitrary alternation depth, an alternating Turing machine running in k-EXP time and performing d alternations implements the following recursive procedure.

1. Guess functions from $\mathbb{B}_{k,n}$ for each of the function symbols in the vectors $\boldsymbol{f}_1, \dots, \boldsymbol{f}_d$, alternating between existential and universal states according to the quantifier prefix.
2. Recursively on Φ, if $\Phi = \exists f \colon \mathbb{B}_{j,n}\ \Psi$ (resp. $\Phi = \forall f \colon \mathbb{B}_{j,n}\ \Psi$) with $j < k$ then check whether some (resp. each) function of order j satisfies Φ when assigned to f; if Φ is quantifier-free, then check whether it is satisfied under the current assignment of function symbols.

Since k-exponential time is available, the second step can be performed deterministically by iterating through all function in $\mathbb{B}_{j,n}$, thus without introducing further alternation.

The $\Sigma_d^{k\text{-EXP}}$ lower bound of $\mathrm{HOSAT}(k,d)$ is shown by reducing from the *d-alternating multi-tiling problem of order k* (in short, $\mathrm{AMTP}(k,d)$) considered in [10]. A *multi-tiling system* $\mathcal{S}$ is a tuple $(\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\mathrm{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, m)$ such that $\mathcal{T}$ is a finite set of *tile types*, $\mathcal{T}_0, \mathcal{T}_{\mathrm{acc}} \subseteq \mathcal{T}$ are sets of *initial* and *accepting* tiles, respectively, $\mathcal{H}, \mathcal{V}, \mathcal{M} \subseteq \mathcal{T} \times \mathcal{T}$ represent the *horizontal*, *vertical* and *multi-tiling* matching relations, respectively, and $m \in \mathbb{N}_+$ is a number written in unary. We write $|\mathcal{S}|$ for the number of symbols required to encode $\mathcal{S}$.

Fix $k \in \mathbb{N}_+$, and let $\mathrm{grid}(k,m)$ be the two-dimensional grid $[\exp_2^k(m)] \times [\exp_2^k(m)]$, where we recall that $[u] := [0,u)$. Each $(h,v) \in \mathrm{grid}(k,m)$ is said to be a *position* of the grid, comprised of a *horizontal position* h and a *vertical position* v. Let $d \in \mathbb{N}$ odd (so that the innermost quantifier of the $\mathrm{AMTP}(k,d)$ problem we are about to formalise is existential). A *d-level $\mathcal{S}$-tiling* for $\mathrm{grid}(k,m)$ is a tuple $(\mathfrak{f}_1, \mathfrak{f}_2, \mathfrak{f}_3, \dots, \mathfrak{f}_d)$ such that for all $\ell \in [1,d]$:

maps: $\mathfrak{f}_\ell \colon \mathrm{grid}(k,m) \to \mathcal{T}$ assigns a tile type to each position of $\mathrm{grid}(k,m)$;
hori: $(\mathfrak{f}_\ell(i,j), \mathfrak{f}_\ell(i,j+1)) \in \mathcal{H}$ for every $j \in [\exp_2^k(m)-1]$ and $i \in [\exp_2^k(m)]$;
vert: $(\mathfrak{f}_\ell(i,j), \mathfrak{f}_\ell(i+1,j)) \in \mathcal{V}$ for every $i \in [\exp_2^k(m)-1]$ and $j \in [\exp_2^k(m)]$;
multi: if $\ell < d$, then $(\mathfrak{f}_\ell(i,j), \mathfrak{f}_{\ell+1}(i,j)) \in \mathcal{M}$ for every $i,j \in [\exp_2^k(m)]$; and
accept: $\mathfrak{f}_d(\exp_2^k(m)-1, j) \in \mathcal{T}_{\mathrm{acc}}$, for some $j \in [\exp_2^k(m)]$.

The *initial row* $I(\mathfrak{f})$ of a map $\mathfrak{f} \colon \mathrm{grid}(k,m) \to \mathcal{T}$ is the word $\mathfrak{f}(0,0)\mathfrak{f}(0,1)\dots\mathfrak{f}(0,\exp_2^k(m)-1)$.

$\mathrm{AMTP}(k,d)$: d-ALTERNATING MULTI-TILING PROBLEM OF ORDER k

INPUT: A multi-tiling system $\mathcal{S} = (\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\mathrm{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, m)$.
QUESTION: Is it true that there is $w_1 \in (\mathcal{T}_0)^{\exp_2^k(m)}$ such that for all $w_2 \in (\mathcal{T}_0)^{\exp_2^k(m)}$ there is $\cdots$ there is $w_d \in (\mathcal{T}_0)^{\exp_2^k(m)}$ such that $\mathrm{grid}(k,m)$ has a d-level tiling $(\mathfrak{f}_1, \dots, \mathfrak{f}_d)$ where $I(\mathfrak{f}_\ell) = w_\ell$ for all $\ell \in [1,d]$?

▶ **Proposition 3** ([10]). *The problem* $\mathrm{AMTP}(k,d)$ *is complete for* $\Sigma_d^{k\text{-EXP}}$. *When d is given as part of the input instead of being fixed, the problem is* $\mathrm{STA}(*, \exp_2^k(n^{O(1)}), O(n))$*-complete.*

The $\mathrm{AMTP}(k,d)$ problem arose from [4], in which the case $k = 1$ and d not fixed is shown to be $\mathrm{STA}(*, 2^{n^{O(1)}}, O(n))$-complete. See [25, Appendix E.7] or [23] for self-contained proofs.

In the remaining part of this section, we describe a reduction from $\mathrm{AMTP}(k,d)$ to $\mathrm{HOSAT}(k,d)$. First, we introduce a family of generalised Boolean formulae that allows us to compare the bit-strings $\mathit{enc}(f)$ and $\mathit{enc}(g)$ of two functions f and g in $\mathbb{B}_{k,n}$, with $n,k \in \mathbb{N}$. Subsequently, we define formulae to encode the tiling conditions (maps)–(accept) as well as the alternation on elements of $(\mathcal{T}_0)^{\exp_2^k(m)}$ required by the $\mathrm{AMTP}(k,d)$ problem. We remind the reader that, w.l.o.g., we define generalised Boolean formulae without constraining them to be in prenex normal form, though we need to keep track of the quantifier alternation for function symbols of type k.

Comparing bit-strings. We define formulae $\mathrm{eq}_k(f,g)$, $\mathrm{less}_k(f,g)$ and $\mathrm{succ}_k(f,g)$ stating that $\mathit{enc}(f) = \mathit{enc}(g)$, $\mathit{enc}(f) < \mathit{enc}(g)$ and $\mathit{enc}(f)+1 = \mathit{enc}(g)$, respectively, and $(h_1, \dots, h_n) <_k (h_1', \dots, h_n')$, with all h_i and h_i' in $\mathbb{B}_{k,n}$, which checks if the concatenation $\mathit{enc}(h_1)\cdots\mathit{enc}(h_n)$ encodes a number smaller than $\mathit{enc}(h_1')\cdots\mathit{enc}(h_n')$. The formula $\mathrm{eq}_k(f,g)$ is defined as:

$$\mathrm{eq}_k(f,g) := \forall h_1, \dots, h_n \colon \mathbb{B}_{k-1,n}.f(h_1, \dots, h_n) \leftrightarrow g(h_1, \dots, h_n).$$

The formulae $\mathrm{less}_k(f,g)$ and $(h_1,\dots,h_n) <_k (h'_1,\dots,h'_n)$ have a mutually recursive definition: $\mathrm{less}_{k+1}(f,g)$ is defined using $(h_1,\dots,h_n) <_k (h'_1,\dots,h'_n)$, which in turn requires $\mathrm{less}_k(f,g)$. First, we define the base case $(h_1,\dots,h_n) <_0 (h'_1,\dots,h'_n)$, with each h_i and h'_i in $\mathbb{B} = \{0,1\}$:

$$(h_1,\dots,h_n) <_0 (h'_1,\dots,h'_n) := \bigvee_{i=1}^{n} \neg h_i \wedge h'_i \wedge \bigwedge_{j=i+1}^{n} (h'_j \leftrightarrow h_j).$$

Intuitively, this formula forces $\mathit{enc}(h_1)\dots\mathit{enc}(h_n) < \mathit{enc}(h'_1)\dots\mathit{enc}(h'_n)$ by requiring that there is a bit $i \in [1,n]$ that is set to 0 in $\mathit{enc}(h_1)\dots\mathit{enc}(h_n)$, set to 1 in $\mathit{enc}(h'_1)\dots\mathit{enc}(h'_n)$, and the binary representations of $\mathit{enc}(h'_1)\dots\mathit{enc}(h'_n)$ and $\mathit{enc}(h_1)\dots\mathit{enc}(h_n)$ coincide on all bits $j > i$. This is indeed the characterisation of $<$ on binary numbers in least significant digit first encoding. The same idea is used to define the formula $\mathrm{less}_k(f,g)$. Inductively, assume that the formula $(h_1,\dots,h_n) <_k (h'_1,\dots,h'_n)$ was defined. We use it to define $\mathrm{less}_{k+1}(f,g)$:

$$\mathrm{less}_{k+1}(f,g) := \exists \boldsymbol{h} \colon \mathbb{B}_{k,n} \,.\, \neg f(\boldsymbol{h}) \wedge g(\boldsymbol{h}) \ \wedge \forall \boldsymbol{h'} \colon \mathbb{B}_{k,n} \,.\, \boldsymbol{h} <_k \boldsymbol{h'} \to (f(\boldsymbol{h'}) \leftrightarrow g(\boldsymbol{h'})),$$

where $\boldsymbol{h} = (h_1,\dots,h_n)$ and $\boldsymbol{h'} = (h'_1,\dots,h'_n)$.

To complete the definitions of $\mathrm{less}_k(f,g)$ and $(h_1,\dots,h_n) <_k (h'_1,\dots,h'_n)$, what is left is to define $(h_1,\dots,h_n) <_{k+1} (h'_1,\dots,h'_n)$ using less_{k+1}. We need to respect the equivalence $(h_1,\dots,h_n) <_{k+1} (h'_1,\dots,h'_n) \equiv \bigvee_{i=1}^{n} \mathrm{less}_{k+1}(h_i,h'_i) \wedge \bigwedge_{j=i+1}^{n} \mathrm{eq}_{k+1}(h_j,h'_j)$. However, we cannot define $(h_1,\dots,h_n) <_{k+1} (h'_1,\dots,h'_n)$ as the formula in the right hand side of this equivalence, as this formula uses n occurrences of less_k and would thus lead to both less_k and $(h_1,\dots,h_n) <_k (h'_1,\dots,h'_n)$ being of size exponential in k. To solve this issue and obtain formulae of polynomial size, we rely on a variant of a trick used by Fisher and Rabin in [11], based on the equivalence $\Phi(a) \vee \Phi(b) \equiv \exists c : \Phi(c) \wedge (c = a \vee c = b)$:

$$\begin{aligned}&(h_1,\dots,h_n) <_{k+1} (h'_1,\dots,h'_n) \ := \\ &\quad \exists a,b \colon \mathbb{B}_{k+1,n} \,.\, \mathrm{less}_{k+1}(a,b) \wedge \bigvee_{i=1}^{n} \mathrm{eq}_{k+1}(a,h_i) \wedge \mathrm{eq}_{k+1}(b,h'_i) \wedge \bigwedge_{j=i+1}^{n} \mathrm{eq}_{k+1}(h_j,h'_j).\end{aligned}$$

▶ **Lemma 4.** *Let $f,g \in \mathbb{B}_{k,n}$. Then, $\mathrm{less}_k(f,g)$ iff $\mathit{enc}(f){<}\mathit{enc}(g)$; and $|\mathrm{less}_k(f,g)| \le O(n^3k)$.*

The definition of $\mathrm{succ}_k(f,g)$ follows a similar characterisation of successor for binary numbers: we have $a + 1 = b$ whenever (1) there is a bit i that is set to 0 in a and to 1 in b, (2) the binary representations of a and b coincide on every bit $j > i$ (exactly as in the case of $<$) and moreover (3) every bit $j < i$ is set to 1 in a and it is set to 0 in b. For instance, in least significant digit first encoding, $(1111001)_2 + 1 = (0000101)_2$. We have:

$$\begin{aligned}\mathrm{succ}_{k+1}(f,g) := \exists \boldsymbol{h} \colon \mathbb{B}_{k,n} \,.\, \neg f(\boldsymbol{h}) \wedge g(\boldsymbol{h}) \ \wedge \forall \boldsymbol{h'} \colon \mathbb{B}_{k,n} \,.& \\ \big(\boldsymbol{h} <_k \boldsymbol{h'} \to (f(\boldsymbol{h'}) \leftrightarrow g(\boldsymbol{h'}))\big) \wedge \big(\boldsymbol{h'} <_k \boldsymbol{h} \to f(\boldsymbol{h'}) \wedge \neg g(\boldsymbol{h'})\big),&\end{aligned}$$

where $\boldsymbol{h} = (h_1,\dots,h_n)$ and $\boldsymbol{h'} = (h'_1,\dots,h'_n)$.

▶ **Lemma 5.** *For $f,g \in \mathbb{B}_{k,n}$, $\mathrm{succ}_k(f,g)$ iff $\mathit{enc}(f) + 1 = \mathit{enc}(g)$; and $|\mathrm{succ}_k(f,g)| \le O(n^3k)$.*

From $\mathbf{AMTP}(k,d)$ to $\mathbf{HOSAT}(k,d)$. We are ready to prove the lower bounds of Theorem 2. For $k = 1$, the Σ_d^{EXP}-hardness of $\mathrm{HOSAT}(1,d)$ was already established by Lohrey in [19, Proposition 33] via a reduction from Turing machines. Therefore, we consider $k \ge 2$ (our proof can nonetheless be adapted to the case $k = 1$). Let $\mathcal{S} = (\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\mathrm{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, m)$ be a multi-tiling system. We assume $\mathcal{T} = [1,r]$ for some $r \in \mathbb{N}_+$ (thus $\mathcal{H}, \mathcal{V}, \mathcal{M} \subseteq [1,r] \times [1,r]$).

Let $n := 2 + \#\mathcal{T} + m$. We write $\bot_k$ for the function in $\mathbb{B}_{k,n}$ such that $\mathit{enc}(\bot_k) = 0$, i.e. $\bot_k$ is the only solution f of the formula $\mathrm{bot}_k(f) := \forall g_1,\dots,g_n \colon \mathbb{B}_{k-1,n} \,.\, \neg f(g_1,\dots,g_n)$. Similarly, we write $\top_k$ for the function in $\mathbb{B}_{k,n}$ with maximal encoding, that is the only solution f to the formula $\mathrm{top}_k(f) := \forall g_1,\dots,g_n \colon \mathbb{B}_{k-1,n} \,.\, f(g_1,\dots,g_n)$. The first step of the reduction consists of encoding the d functions of a d-level $\mathcal{S}$-tiling $(\mathfrak{f}_1,\dots,\mathfrak{f}_d)$. We encode each of these functions using a function $f \in \mathbb{B}_{k,n}$ satisfying the following two properties:

1: If $f(h,v,t_1,\dots,t_r,u_1,\dots,u_m) = 1$ then $enc(h)$ and $enc(v)$ belong to $[\exp_2^k(m)]$, each t_i belongs to $\{\bot_{k-1},\top_{k-1}\}$ and every u_j is $\bot_{k-1}$.

2: Given $h,v \in \mathbb{B}_{k-1,n}$, if $enc(h), enc(v) \in [\exp_2^k(m)]$ then there is exactly one tuple $\boldsymbol{t} = (t_1,\dots,t_r)$ such that $f(h,v,\boldsymbol{t},u_1,\dots,u_m) = 1$; and exactly one among $t_1,\dots,t_r$ is $\top_{k-1}$.

Here, the inputs h and v are used to represent horizontal and vertical positions in the grid, the inputs $t_1,\dots,t_r$ are used to encode the tiles, and the inputs $u_1,\dots,u_m$ are only required to makes sure we have enough inputs to encode the fact that h and v belong to $[\exp_2^k(m)]$ (see the formula ok_1 below). Together, Properties (1) and (2) characterise the condition (maps) of the $\mathcal{S}$-tiling (we add the other tiling conditions later). To capture Property 1 above, notice first that h and v should be taken from a space of exactly $\exp_2^k(m)$ functions. As $\#(\mathbb{B}_{k-1,n}) \geq \exp_2^k(m)$, this obliges us to introduce a formula $\mathrm{ok}_k(g)$ characterising the fact that a function $g \in \mathbb{B}_{k,n}$ is such that $enc(g) \in [\exp_2^{k+1}(m)]$. We have

$$\begin{aligned}\mathrm{ok}_1(g) &:= \forall h_1,\dots,h_n \colon \mathbb{B} \,.\, g(h_1,\dots,h_n) \to \textstyle\bigwedge_{i=m+1}^n \neg h_i\\ \mathrm{ok}_{k+1}(g) &:= \forall h_1,\dots h_n \colon \mathbb{B}_{k,n} \,.\, g(h_1,\dots,h_n) \to (\mathrm{ok}_k(h_1) \wedge \textstyle\bigwedge_{i=2}^n \mathrm{bot}_k(h_i)).\end{aligned}$$

Intuitively, for $k \geq 1$ the formula $\mathrm{ok}_k(g)$ holds whenever for all inputs $(h_1,\dots,h_n)$ and $j := enc(h_1)\cdots enc(h_n)$, if the j-th bit of $enc(g)$ is set to 1 then $j \in [\exp_2^k(m)]$. This means that $enc(g)$ corresponds to a binary number with $\exp_2^k(m)$ bits, i.e. $enc(g) \in [\exp_2^{k+1}(m)]$.

In all the formulae below, we let $\boldsymbol{t} = (t_1,\dots,t_r)$, $\boldsymbol{u} = (u_1,\dots,u_m)$ and $\boldsymbol{t'} = (t'_1,\dots,t'_r)$. We define the formula $\mathrm{mapsOne}_k(f)$ stating that $f \in \mathbb{B}_{k,n}$ satisfies Property 1:

$$\begin{aligned}\mathrm{mapsOne}_k(f) :=& \forall h,v,\boldsymbol{t},\boldsymbol{u} \colon \mathbb{B}_{k-1,n} \,.\, f(h,v,\boldsymbol{t},\boldsymbol{u}) \to \mathrm{ok}_{k-1}(h) \wedge \mathrm{ok}_{k-1}(v) \wedge\\ &\textstyle\bigwedge_{i=1}^r(\mathrm{top}_{k-1}(t_i) \vee \mathrm{bot}_{k-1}(t_i)) \wedge \bigwedge_{j=1}^m \mathrm{bot}_{k-1}(u_j).\end{aligned}$$

Suppose that $f \in \mathbb{B}_{k,n}$ satisfies $\mathrm{mapsOne}_k(f)$. In a similar fashion, one defines a formula $\mathrm{mapsTwo}_k(f)$ stating that f satisfies Property 2 of the encoding:

$$\begin{aligned}&\mathrm{mapsTwo}_k(f) := \forall h,v \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{ok}_{k-1}(h) \wedge \mathrm{ok}_{k-1}(v) \to \exists \boldsymbol{t},\boldsymbol{u} \colon \mathbb{B}_{k-1,n} \,.\, f(h,v,\boldsymbol{t},\boldsymbol{u}) \wedge\\ &\textstyle\bigvee_{i=1}^r(\mathrm{top}_{k-1}(t_i) \wedge \bigwedge_{\substack{j=1\\ j\neq i}}^r \mathrm{bot}_{k-1}(t_j)) \wedge \forall \boldsymbol{t'} \colon \mathbb{B}_{k-1,n} \,.\, f(h,v,\boldsymbol{t'},\boldsymbol{u}) \to \bigwedge_{i=1}^r \mathrm{eq}_{k-1}(t_i,t'_i).\end{aligned}$$

We define $\mathrm{maps}_k(f) := \mathrm{mapsOne}_k(f) \wedge \mathrm{mapsTwo}_k(f)$, and move to the remaining tiling conditions. The conditions (hori) and (vert) can be defined with the help of the formula succ_k. Below, we present the definition of the formula $\mathrm{hori}_k(f)$ that encodes the condition (hori):

$$\begin{aligned}\mathrm{hori}_k(f) := \forall h,v,v' \colon \mathbb{B}_{k-1,n} \,.\, &\mathrm{ok}_{k-1}(h) \wedge \mathrm{ok}_{k-1}(v) \wedge \mathrm{ok}_{k-1}(v') \wedge \mathrm{succ}_{k-1}(v,v')\\ &\to \textstyle\bigvee_{(i,j)\in\mathcal{H}} f(h,v,\underline{i}) \wedge f(h,v',\underline{j}),\end{aligned}$$

where $f(h,v,\underline{i})$ is a shortcut for $\exists \boldsymbol{t},\boldsymbol{u} \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{top}_{k-1}(t_i) \wedge f(h,v,\boldsymbol{t},\boldsymbol{u})$, i.e. a formula stating that the tile $i \in \mathcal{T}$ is assigned to the position $(enc(h), enc(v))$ of the grid, under the hypothesis that f satisfies $\mathrm{maps}_k(f)$. The definition of the formula $\mathrm{vert}_k(f)$ encoding the condition (vert) is defined analogously. For the condition (multi), let f and g be two functions of $\mathbb{B}_{k,n}$ satisfying maps_k. We define:

$$\mathrm{multi}_k(f,g) := \forall h,v \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{ok}_{k-1}(h) \wedge \mathrm{ok}_{k-1}(v) \to \textstyle\bigvee_{(i,j)\in\mathcal{M}} f(h,v,\underline{i}) \wedge g(h,v,\underline{j}).$$

Lastly, we define the formula $\mathrm{acc}_k(f)$ corresponding to the condition (accept):

$$\begin{aligned}\mathrm{acc}_k(f) := \exists h, v \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{ok}_{k-1}(h) \land \mathrm{ok}_{k-1}(v) \land \big(\textstyle\bigvee_{i \in \mathcal{T}_{\mathrm{acc}}} f(h, v, \underline{i})\big) \land \\ \forall h' \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{less}_{k-1}(h, h') \to \neg \mathrm{ok}_{k-1}(h').\end{aligned}$$

Here, the subformula $\forall h' \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{less}_{k-1}(h, h') \to \neg \mathrm{ok}_{k-1}(h')$ forces $enc(h) = \exp_2^2(m) - 1$.

To complete the reduction, what is left is to encode the quantifier prefix "$\exists w_1 \in (\mathcal{T}_0)^{\exp_2^k(m)}$ $\forall w_2 \in (\mathcal{T}_0)^{\exp_2^k(m)}$ $\dots$ $\exists w_d \in (\mathcal{T}_0)^{\exp_2^k(m)}$" of the $\mathrm{AMTP}(k, d)$ problem. To this end, we introduce two formulae $\mathrm{row}_k(f)$ and $\mathrm{rowCp}_k(f, g)$. The former states that $f \in \mathbb{B}_{k,n}$ satisfies maps_k, and whenever $f(h, v, \boldsymbol{t}, \boldsymbol{u})$ holds the only t_i equal to $\top_{k-1}$ is such that $i \in \mathcal{T}_0$:

$$\mathrm{row}_k(f) := \mathrm{maps}_k(f) \land \forall h, v, \boldsymbol{t}, \boldsymbol{u} : \mathbb{B}_{k-1,n} \,.\, f(h, v, \boldsymbol{t}, \boldsymbol{u}) \to \textstyle\bigvee_{i \in \mathcal{T}_0} \mathrm{top}_k(t_i).$$

We use this formula to quantify on the initial row $I(\mathfrak{f})$ of a $\mathcal{S}$-tiling function $\mathfrak{f}$, forgetting about the information stored by f elsewhere. This is done thanks to the formula $\mathrm{rowCp}_k(f, g)$, which given $f, g \in \mathbb{B}_{k,n}$ satisfying maps_k, states that f and g agree on the first row:

$$\mathrm{rowCp}_k(f, g) := \exists h \colon \mathbb{B}_{k-1,n} \,.\, \mathrm{bot}_{k-1}(h) \land \forall v, \boldsymbol{t}, \boldsymbol{u} \colon \mathbb{B}_{k-1,n} \,.\, f(h, v, \boldsymbol{t}, \boldsymbol{u}) \leftrightarrow g(h, v, \boldsymbol{t}, \boldsymbol{u}),$$

The quantifier prefix of $\mathrm{AMTP}(k, d)$ is encoded in $\mathrm{HOSAT}(k, d)$ by first quantifying over functions $w_1, \dots, w_d \in \mathbb{B}_{k,n}$ satisfying row_k, appropriately alternating between existential and universal quantification, and then existentially quantifying over functions $f_1, \dots, f_d \in \mathbb{B}_{k,n}$ that encode the $\mathcal{S}$-tiling functions $(\mathfrak{f}_1, \dots, \mathfrak{f}_d)$. The formula $\mathrm{rowCp}_k(f_i, w_i)$ is used to "copy" the first row of w_i in f_i. This leads to the formula $\mathrm{amtp}_k(\mathcal{S})$:

$$\begin{aligned}&\exists w_1 \colon \mathbb{B}_{k,n} \,.\, \mathrm{row}_k(w_1) \land \\ &\quad \forall w_2 \colon \mathbb{B}_{k,n} \,.\, \mathrm{row}_k(w_2) \to \\ &\quad\quad \dots \\ &\quad\quad\quad \exists w_d \colon \mathbb{B}_{k,n} \,.\, \mathrm{row}_k(w_d) \land \\ &\quad\quad\quad\quad \exists f_1, \dots, f_d \colon \mathbb{B}_{k,n} \,.\, \textstyle\bigwedge_{i=1}^{d} (\mathrm{maps}_k(f_i) \land \mathrm{rowCp}_k(f_i, w_i) \land \mathrm{hori}_k(f_i) \land \mathrm{vert}_k(f_i)) \\ &\quad\quad\quad\quad\quad\quad \land \textstyle\bigwedge_{i=1}^{d-1} \mathrm{multi}_k(f_i, f_{i+1}) \land \mathrm{acc}_k(f_d).\end{aligned}$$

▶ **Proposition 6.** *Let $d \in \mathbb{N}_+$ and $k \geq 2$. The formula $\mathrm{amtp}_k(\mathcal{S})$ is valid if and only if $\mathrm{AMTP}(k, d)$ accepts on input $\mathcal{S}$. Moreover, the formula $\mathrm{amtp}_k(\mathcal{S})$ has size $O(d \cdot k \cdot |\mathcal{S}|^4)$.*

Notice that $\mathrm{amtp}_k(\mathcal{S})$ is a generalised Boolean formula of order k and alternation depth d, since bringing it into prenex normal form yields a formula of the form $\exists w_1 \colon \mathbb{B}_{k,n} \, \forall w_2 \colon \mathbb{B}_{k,n} \dots$ $\exists w_d, f_1, \dots, f_d \colon \mathbb{B}_{k,n} \,.\, \Phi$, where Φ is a generalised Boolean formula of order $k - 1$ and size in $O(d \cdot k \cdot |\mathcal{S}|^4)$. By Proposition 3 (first part), this completes the proof of Theorem 2(I). Theorem 2(II) is proven analogously, by simply treating d as part of the input instead of being fixed. The upper bound follows the same strategy as the case of $\mathrm{HOSAT}(k, d)$, and the lower bound follows form Proposition 6 together with the second part of Proposition 3.

5 Weak Presburger arithmetic is as hard as Presburger arithmetic

In this section, we illustrate the usefulness of the higher order satisfiability problem by employing it to derive a $\mathrm{STA}(*, \exp_2^2(n^{O(1)}), O(n))$ lower bound for the weak fragment of Presburger arithmetic (*Weak PA*), hence showing that this logic matches the complexity of (standard) Presburger arithmetic. Weak PA is the first-order theory of the structure $\langle \mathbb{Z}, (c)_{c \in \mathbb{Z}}, +, = \rangle$, where $(c)_{c \in \mathbb{Z}}$ are constant symbols interpreted as their homographic integer, the binary function symbol $+$ is interpreted as addition on $\mathbb{Z}$ and the binary relation $=$ is interpreted as equality on $\mathbb{Z}$. Subsequently, linear Terms $t, t', \dots$ are of the form $a_1x_1 + \dots + a_dx_d + c$, where d is arbitrary, $a_1, \dots, a_d, c \in \mathbb{Z}$, and $x_1, \dots, x_d$ are first-order variables interpreted over $\mathbb{Z}$. Formulae of Weak PA close equalities between linear terms $t = t'$ under Boolean connectives $\neg$, $\wedge$, $\vee$, etc., and first-order quantifiers $\exists x$ and $\forall x$. For example, given $m \in \mathbb{N}_+$ and linear terms t, t', the *modulo constraint* $t \equiv_m t'$ stating that t is congruent to t' modulo m is characterised by the Weak PA formula $\exists x \,.\, t - t' = x \cdot m$, where x is a variable not appearing in t or t'.

Surprisingly, our lower bound holds for the following two fragments of Weak PA:

- The *positive fragment* that forbids negation, allowing formulae from the grammar
$$\Phi ::= a_1x_1 + \dots + a_dx_d = c \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists x \,.\, \Phi \mid \forall x \,.\, \Phi.$$
- The *Horn fragment*, in which formulae are of the form $\exists \boldsymbol{x}_1 \forall \boldsymbol{x}_2 \dots \exists \boldsymbol{x}_m \,.\, \bigwedge_{i \in I}(\Phi_i \to \Psi_i)$, where $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$ are vectors of variables, and each Φ_i and Ψ_i is a system of equalities, that is a conjunction of equalities between linear terms.

▶ **Theorem 7.** *The positive and Horn fragments of Weak PA are* $\mathrm{STA}(*, 2^{2^{n^{O(1)}}}, O(n))$*-hard.*

To prove this theorem we provide a reduction from $\mathrm{HOSAT}(2, *)$ to the validity problem for the positive fragment of Weak PA, and a translation from Weak PA to its Horn fragment. The reduction defines arithmetic with multiplication on doubly exponential finite segments of integers, such as $[2^{2^n}]$, and, in a more restricted way, one exponential higher. Here we strengthen Berman's lower bound argument for standard PA [3] (see also [18, Lecture 22]), which relies on the order predicate throughout.

Encoding second-order Boolean functions in Weak PA. Let $n \in \mathbb{N}_+$ be encoded in unary. In a nutshell, the main difficulty in the reduction from $\mathrm{HOSAT}(2, *)$ is to understand how to encode the functions in $\mathbb{B}_{1,n}$ and $\mathbb{B}_{2,n}$ using integer numbers, as well as translating the function applications $f(g_1, \dots, g_n)$. For the set $\mathbb{B}_{1,n}$, we rely on the encoding $\mathit{enc}(.)$ defined in Section 3 that maps every function $f \in \mathbb{B}_{1,n}$ into the number $\mathit{enc}(f) \in [2^{2^n}]$ written in binary. To encode functions in $\mathbb{B}_{2,n}$, we borrow ideas from [13, 14]. We say that $z \in \mathbb{Z}$ *encodes some function in* $\mathbb{B}_{2,n}$ if and only if for every $i \in [2^{n2^n}]$ and all prime numbers $p, q \in [i^3, (i+1)^3)$, $z \equiv_p 0$ or $z \equiv_p 1$, and $z \equiv_p 0$ if and only if $z \equiv_q 0$. Furthermore, we say that $z \in \mathbb{Z}$ *(precisely) encodes the function* $f \in \mathbb{B}_{2,n}$ if and only if z encodes some function in $\mathbb{B}_{2,n}$ and moreover for every $\boldsymbol{g} = (g_0, \dots, g_{n-1}) \in (\mathbb{B}_{1,n})^n$ and $b \in \{0, 1\}$,

$$f(\boldsymbol{g}) = b \quad \text{if and only if} \quad z \equiv_p b \text{ for a prime } p \in [\beta(\boldsymbol{g})^3, (\beta(\boldsymbol{g}) + 1)^3),$$

where $\beta : (\mathbb{B}_{1,n})^n \to [0, 2^{n2^n})$ is the bijection $\beta(g_0, \dots, g_{n-1}) := \sum_{j=0}^{n-1} \mathit{enc}(g_j) \cdot 2^{j \cdot 2^n}$. The fact that any function in $\mathbb{B}_{2,n}$ is encoded by some $z \in \mathbb{Z}$ follows from the Chinese remainder theorem together with Ingham's theorem [16], a theorem ensuring that for i sufficiently large, $[i^3, (i+1)^3)$ contains at least one prime. As in, e.g., [13, 14], to simplify the presentation we apply Ingham's theorem as if it was known to be true for all $i \in \mathbb{N}$. An alternative would be to use an analogous result on primes between fixed powers that holds for all i [9], or to add constant offsets throughout.

From $\mathrm{HOSAT}(2, *)$ to the positive fragment of Weak PA. We formalise the translation τ that, given a formula from $\mathrm{HOSAT}(2, *)$ returns an equivalent formula in the positive fragment of Weak PA. We divide the translation into three parts, depending on whether we are dealing with a generalised Boolean formula of order 0, 1 or 2. The translation τ is homomorphic for

formula	semantics	formula	semantics
$\varphi_n(x,z)$	$x = 2^{2^n} \cdot z$	$\mathrm{intdiv}_n(x,y,q,r)$	$x = q \cdot y + r$, $y \in [2^{2^n}]$, $r \in [y]$
$\mathrm{mult}_n(x,y,z)$	$x = y \cdot z$, $z \in [2^{2^n}]$	$\mathrm{quot}_n(x,y,q)$	$\exists r \in [y]$ s.t. $\mathrm{intdiv}_n(x,y,q,r)$
$I_n(x)$	$z \in [2^{2^n}]$	$\mathrm{rem}_n(x,y,r)$	$\exists q \in \mathbb{Z}$ s.t. $\mathrm{intdiv}_n(x,y,q,r)$
$\mathrm{power}_n(y,j)$	$y = 2^j$ and $j \in [2^n]$	$\mathrm{(not)prime}_n(p)$	$p \in [2^{2^n}]$ is (not) prime
$\psi_{k,n}(x)$	$x = 2^{k2^n}$	$x =_n y^3$	$y \in [2^{2^{2(n+1)}}]$ and $x = y^3$

Figure 1 Auxiliary formulae required to reduce HOSAT$(2,*)$ to positive Weak PA.

binary Boolean connectives: $\tau(\Phi \wedge \Psi) := \tau(\Phi) \wedge \tau(\Psi)$ and $\tau(\Phi \vee \Psi) := \tau(\Phi) \vee \tau(\Psi)$. Without loss of generality we assume that negations occurring in the quantifier-free part of generalised Boolean formulae only appear in front of function applications (recall that we see Boolean values as 0-ary functions). A *(function application) literal* is either a function application $f(g_1, \dots, g_\ell)$ or its negation $\neg f(g_1, \dots, g_\ell)$.

Formulae of order 0. We translate $\exists f \colon \mathbb{B}$, $\forall f \colon \mathbb{B}$ and literals f and $\neg f$, with f of type $\mathbb{B}$. Let $\mathbb{B}(x) := x = 0 \vee x = 1$, i.e. the formula stating $x \in \mathbb{B}$. Clearly, $\tau(f) := f = 1$ and $\tau(\neg f) := f = 0$. The quantifiers $\exists f \colon \mathbb{B}$ and $\forall f \colon \mathbb{B}$ are translated as follows:

$$\tau(\exists f \colon \mathbb{B}\ \Phi) := \exists f \,.\, \mathbb{B}(f) \wedge \tau(\Phi), \qquad \tau(\forall f \colon \mathbb{B}\ \Phi) := \forall f' \exists f \,.\, f \equiv_2 f' \wedge \mathbb{B}(f) \wedge \tau(\Phi).$$

A few words on the translation of $\forall f \colon \mathbb{B}$. We cannot translate the universal quantifier $\forall f \colon \mathbb{B}\ \Phi$ as $\forall f \,.\, \mathbb{B}(f) \to \tau(\Phi)$, i.e. relying on the $\exists$-$\forall$ duality, as $\mathbb{B}(f) \to \tau(\Phi)$ is not in the positive fragment of Weak PA. Our definition circumvents this problem by relying on the equivalence $\forall x \,.\, x \in [0,\ell] \to \Phi(x, \boldsymbol{z}) \quad \equiv \quad \forall x' \exists x \,.\, x' \equiv_{\ell+1} x \wedge x \in [0,\ell] \wedge \Phi(x, \boldsymbol{z})$.

Formulae of order 1. We treat quantifiers $\exists f \colon \mathbb{B}_{1,n}$, $\forall f \colon \mathbb{B}_{1,n}$ and literals $f(g_1, \dots, g_n)$ and $\neg f(g_1, \dots, g_n)$, where f is of type $\mathbb{B}_{1,n}$ and each g_i is of type $\mathbb{B}$. To achieve this we rely on the auxiliary formulae formally specified in Figure 1 and defined below ($\psi_{k,n}(x)$, $\mathrm{notprime}_n(p)$ and $x =_n y^3$ are defined later as only used for formulae of order 2). The formulae φ_n, mult_n, I_n and intdiv_n are an adaptation of the homonymous formulae provided by Kozen in [18, Lecture 22] in the context of linear real arithmetic. For instance, the formula $\varphi_n(x,z)$ is inductively defined in [18, p. 147] as follows:

$$\varphi_0(x,z) := x = 2z, \qquad \varphi_{n+1}(x,z) := \exists y \forall a, b \,.\, (a = x \wedge b = y) \vee (a = y \wedge b = z) \to \varphi_n(a,b).$$

The inductive case of $\varphi_{n+1}(x,z)$ is defined relying on the trick of Fisher and Rabin [11] we already encountered in Section 3, used here to obtain a linear size formula equivalent to $\exists y : \varphi_n(x,y) \wedge \varphi_n(y,z)$. As it stands $\varphi_{n+1}(x,z)$ is not in the positive fragment of Weak PA. We rely on modulo constraints to resolve this issue, redefining $\varphi_{n+1}(x,z)$ as follows:

$$\varphi_{n+1}(x,z) := \exists y \forall i \exists a, b \,.\, \big((i \equiv_2 0 \wedge a = x \wedge b = y) \vee (i \equiv_2 1 \wedge a = x \wedge b = y)\big) \wedge \varphi_n(a,b)\,.$$

The definitions of $\mathrm{mult}_n(x,y,z)$, $I_n(x)$ and $\mathrm{intdiv}_n(x,y,q,r)$ require similar adaptations w.r.t. [18, p. 148f], which are omitted due to space constraints. The formulae quot_n and rem_n are simple shortcuts of intdiv_n, e.g., $\mathrm{quot}_n(x,y,q) := \exists r\, \mathrm{intdiv}_n(x,y,q,r)$.

Finally, $\mathrm{power}_n(y,j)$ is intuitively defined by bit-blasting j into n bits $j_0, \dots, j_{n-1}$, so that $j = \sum_{i=0}^{n-1} j_i \cdot 2^i$, and forcing $y = 2^j = \prod\{\exp_2^2(i) : i \in [n] \text{ and } j_i = 1\}$ to hold:

$$\begin{aligned}\mathrm{power}_n(y,j) := \exists (j_0, y_0, z_0), \dots, (j_{n-1}, y_{n-1}, z_{n-1}) \,.\, j = \textstyle\sum_{i=0}^{n-1} j_i \cdot 2^i \ \wedge z_0 = y_0 \wedge y = z_{n-1} \\ \wedge \textstyle\bigwedge_{i=1}^{n-1} \big(\mathrm{mult}_n(z_i, z_{i-1}, y_i) \wedge \big((j_i = 0 \wedge y_i = 1) \vee (j_i = 1 \wedge \varphi_i(y_i, 1))\big)\big).\end{aligned}$$

The subformula $z_0 = y_0 \land y = z_{n-1} \land \bigwedge_{i=1}^{n-1} \mathrm{mult}_n(z_i, z_{i-1}, y_i)$ computes $y = \prod_{i=0}^{n-1} y_i$. We are ready to translate the generalised Boolean formula of order 1. Recall that we encode a function f in $\mathbb{B}_{1,n}$ with the number $\mathit{enc}(f) \in [2^{2^n}]$, and we have $f(g_0, \dots, g_{n-1}) = 1$ if and only if the jth bit of $\mathit{enc}(f)$ is 1, where $j = \sum_{i=1}^{n-1} g_i \cdot 2^i$. With this in mind, the case for existential quantifiers closely follows the definition for formulae of order 0: $\tau(\exists f\colon \mathbb{B}_{1,n}\ \Phi) := \exists f \,.\, I_n(f) \land \tau(\Phi)$. The case of universal quantifiers is more involved: whereas for the case of formulae of order 0 we resorted to reasoning modulo 2, we now reason modulo 2^{2^n}. We can bind 2^{2^n} to a variable ℓ with the formula $\varphi_n(\ell, 1)$. For $m \in \mathbb{Z}$ and $r \in [2^{2^n}]$, we can then check if $m \equiv_\ell r$ holds using the formula $\mathrm{rem}_{n+1}(m, \ell, r)$. Note that we use rem_{n+1} instead of rem_n, as ℓ belongs to the set $[2^{2^{n+1}}] \setminus [2^{2^n}]$. Here is the translation:

$$\tau(\forall f\colon \mathbb{B}_{1,n}\ \Phi) := \forall f' \exists f, \ell \,.\, \varphi_n(\ell, 1) \land \mathrm{rem}_{n+1}(f', \ell, f) \land \tau(\Phi).$$

For the function application literal $f(g_0, \dots, g_{n-1})$, where each g_i is mapped through τ into a homonymous Boolean variable according to the translation of formulae of order 0, we consider the number $j = \sum_{i=0}^{n-1} g_i \cdot 2^i$ and check that the jth bit of $\mathit{enc}(f)$ is set to 1 by verifying that the quotient of the division $f/2^j$ is odd. As a formula:

$$\tau(f(g_0, \dots, g_{n-1})) := \exists j, y, q \,.\, j = \textstyle\sum_{i=0}^{n-1} g_i \cdot 2^i \land \mathrm{power}_n(y, j) \land \mathrm{quot}_n(f, y, q) \land q \equiv_2 1.$$

We treat the literal $\neg f(g_0, \dots, g_{n-1})$ in a similar way, by checking whether $f/2^j$ is even. So, $\tau(\neg f(g_0, \dots, g_{n-1}))$ is obtained from $\tau(f(g_0, \dots, g_{n-1}))$ by replacing $q \equiv_2 1$ with $q \equiv_2 0$.

Formulae of order 2. To complete the definition of τ, we now show how to handle quantifiers $\exists f : \mathbb{B}_{2,n}$ and $\forall f : \mathbb{B}_{2,n}$, and function application literals $f(g_1, \dots, g_n)$ and $\neg f(g_1, \dots, g_n)$, where f is of type $\mathbb{B}_{2,n}$ and every g_i is of type $\mathbb{B}_{1,n}$. As explained at the beginning of the section, functions in $\mathbb{B}_{2,n}$ are encoded as integers $z \in \mathbb{Z}$ having the property that for every $i \in [2^{n2^n}]$ and all prime numbers $p, q \in [i^3, (i+1)^3)$, $z \equiv_p 0$ or $z \equiv_p 1$, and $z \equiv_p 0$ if and only if $z \equiv_q 0$. Below, we aim at defining the formula $\mathrm{enc}_n^2(z)$ stating that z encodes some function in $\mathbb{B}_{2,n}$. We start by defining the formula $\mathrm{notprime}_n(p)$ from Figure 1:

$$\mathrm{notprime}_n(p) := I_n(p) \land \exists d \,.\, I_n(d) \land I_n(d-2) \land I_n(p-d-1) \land \mathrm{rem}_n(p, d, 0).$$

Informally, $\mathrm{notprime}_n(p)$ states that p is not a prime number by finding a divisor $d \in [2, p-1]$. Notice that if we assume $p, d \in [2^{2^n}]$ then $I_n(d-2) \equiv d \geq 2$ and $I_n(p-d-1) \equiv d < p$.

Two further comments on the definition of encoding for functions in $\mathbb{B}_{2,n}$ given above: first, observe that this definition requires the construction of numbers $(i+1)^3$ with $i \in [2^{n2^n}]$. Since $(2^{n2^n} + 1)^3 < 2^{2^{2(n+1)}}$, all these numbers satisfy the formula $I_{2(n+1)}(x)$. This explains the contribution of $\mathrm{notprime}_{2(n+1)}$ and similar formulae in the forthcoming definition of $\mathrm{enc}_n^2(z)$. Second, the encoding requires to iterate over all $i \in [2^{n2^n}]$. As in the definition of τ for the case $\forall f : \mathbb{B}_{1,n}$, this is done by binding 2^{n2^n} to a variable ℓ and then considering all numbers in $[\ell]$ by relying on the formula $\mathrm{rem}_{2(n+1)}$. To characterise 2^{n2^n} we use the following formula that, given $k \in \mathbb{N}_+$ in unary, is satisfied whenever $x = 2^{k2^n}$:

$$\psi_{k,n}(x) := \exists z_1, \dots, z_k \,.\, z_k = 1 \land \varphi_k(x, z_1) \land \textstyle\bigwedge_{i=1}^{k-1} \varphi_n(z_i, z_{i+1}).$$

We define $x =_n y^3 := \exists z \,.\, \mathrm{mult}_{2(n+1)}(z, y, y) \land \mathrm{mult}_{2(n+1)}(x, z, y)$ and the formula $\mathrm{enc}_n^2(z)$:

$$\begin{aligned}
\mathrm{enc}_n^2(z) := {} & \forall i' \exists i, \ell, a, b \ .\ \psi_{n,n}(\ell) \land \mathrm{rem}_{2(n+1)}(i', \ell, i) \land a =_n i^3 \land b =_n (i+1)^3 \land \\
& \quad \forall p', q' \exists \ell', p, q \ .\ \varphi_{2(n+1)}(\ell', 1) \land \mathrm{rem}_{2n+3}(p', \ell', p) \land \mathrm{rem}_{2n+3}(q', \ell', q) \land \\
& \quad \Big(\textstyle\bigvee_{r \in \{p,q\}} \big(I_{2(n+1)}(a - r - 1) \lor I_{2(n+1)}(r - b) \lor \mathrm{notprime}_{2(n+1)}(r) \big) \\
& \quad\quad \lor \textstyle\bigvee_{s \in \{0,1\}} \big(\mathrm{rem}_{2(n+1)}(z, p, s) \land \mathrm{rem}_{2(n+1)}(z, q, s) \big) \Big).
\end{aligned}$$

Let us dissect this formula line by line. The first line iterates through all $i \in [\ell] = [2^{n2^n}]$, binding a and b to i^3 and $(i{+}1)^3$ respectively. The second line iterates through all $p, q \in [\ell'] = [2^{2^{2(n+1)}}]$. Following the definition of our encoding of functions in $\mathbb{B}_{2,n}$, we check that

- one among p and q lies outside $[a, b)$ or is not prime (third line of the formula), or
- $z \equiv_p 0$ or $z \equiv_p 1$, and $z \equiv_p 0$ if and only if $z \equiv_q 0$ (fourth line of the formula).

▶ **Lemma 8.** *A number $z \in \mathbb{Z}$ satisfies $\mathrm{enc}_n^2(z)$ iff z encodes some function in $\mathbb{B}_{2,n}$.*

We can now define τ for $\exists$ quantifiers: $\tau(\exists f\colon \mathbb{B}_{2,n}\ \Phi) := \exists f : \mathrm{enc}_n^2(f) \wedge \tau(\Phi)$. For universal quantifiers we reason dually and define a positive Weak PA formula $\mathrm{notenc}_n^2(z)$ stating that z does not encode any function in $\mathbb{B}_{2,n}$. Defining this formula requires a positive Weak PA formula to test for the primality of $p \in [2^{2^n}]$:

$$\mathrm{prime}_n(p) := I_n(p) \wedge \forall z \exists d, r \,.\, \mathrm{rem}_n(z, p, d) \wedge \big(d = 0 \vee d = 1 \vee \big(\mathrm{rem}_n(p, d, r) \wedge I_n(r - 1)\big)\big).$$

Afterwards, $\mathrm{notenc}_n^2(z)$ is defined by slightly revisiting $\mathrm{enc}_n^2(z)$:

$$\begin{aligned}\mathrm{notenc}_n^2(z) := \exists i, \ell, a, b, p, q \,.\, &\psi_{n,n}(\ell) \wedge I_{2(n+1)}(\ell - i - 1) \wedge a =_n i^3 \wedge b =_n (i+1)^3 \wedge \\ &\textstyle\bigwedge_{r \in \{p,q\}} \big(I_{2(n+1)}(r - a) \wedge I_{2(n+1)}(b - r - 1) \wedge \mathrm{prime}_{2(n+1)}(r)\big) \wedge \\ &\exists s, t \,.\, \mathrm{rem}_{2(n+1)}(z, p, s) \wedge \mathrm{rem}_{2(n+1)}(z, p, t) \wedge (I_{2(n+1)}(s - 2) \vee s \equiv_2 t + 1).\end{aligned}$$

We define the translation for universal quantifiers as $\tau(\forall f\colon \mathbb{B}_{2,n}\ \Phi) := \forall f : \mathrm{notenc}_n^2(f) \vee \tau(\Phi)$.

Let $\boldsymbol{g} = (g_0, \dots, g_{n-1})$. What is left is to treat the literals $f(\boldsymbol{g})$ and $\neg f(\boldsymbol{g})$. Once more, recall that every g_i is a function in $\mathbb{B}_{1,n}$, and thus it is translated through τ into a homonymous integer variable that is constrained to be in $[2^{2^n}]$. Suppose that z encodes the function f. To check whether $f(\boldsymbol{g})$ holds (resp. does not hold), we must check whether $z \equiv_p 1$ (resp. $z \equiv_p 0$) for some prime number p in the interval $[i^3, (i+1)^3)$ with $i := \sum_{j=0}^{n-1} g_j \cdot 2^{j \cdot 2^n}$. Formally, given $d \in \{0, 1\}$ and $i \in [2^{n2^n}]$, we define the macro $z[i] =_n d$ to check this property:

$$\begin{aligned}z[i] =_n d \ := \ \exists a, b, p \,.\ a =_n i^3 \wedge b =_n (i+1)^3 \wedge \mathrm{prime}_{2(n+1)}(p) \wedge \\ I_{2(n+1)}(p - a) \wedge I_{2(n+1)}(b - p - 1) \wedge \mathrm{rem}_{2(n+1)}(z, p, d).\end{aligned}$$

It then suffices to compute i from $\boldsymbol{g} = (g_0, \dots, g_{n-1})$, with the following formula

$$\gamma_n(i, \boldsymbol{g}) := \exists x_0, y_0, \dots, x_{n-1}, y_{n-1} \,.\, i = \textstyle\sum_{j=0}^{n-1} x_j \ \wedge \bigwedge_{j=0}^{n-1} \big(\psi_{j,n}(y_j) \wedge \mathrm{mult}_{2(n+1)}(x_j, y_j, g_j)\big),$$

leading to the following translations for the literals $f(\boldsymbol{g})$ and $\neg f(\boldsymbol{g})$:

$$\tau(f(\boldsymbol{g})) := \exists i \,.\, \gamma_n(i, \boldsymbol{g}) \wedge f[i] =_n 1, \qquad \tau(\neg f(\boldsymbol{g})) := \exists i \,.\, \gamma_n(i, \boldsymbol{g}) \wedge f[i] =_n 0.$$

The correctness of the translation τ is provided by the following proposition, shown by structural induction on the generalised Boolean formula Φ.

▶ **Proposition 9.** *A generalised Boolean formula Φ of order 2 is valid if and only if so is $\tau(\Phi)$. The size of $\tau(\Phi)$ is polynomial in the size of Φ.*

Horn Weak PA = Weak PA. To complete the proof of Theorem 7, we need to show that Weak PA reduces to its Horn fragment. Briefly, this is done with standard formula manipulations and by relying on the equivalences below (notice the similarities to the trick used to define φ_n):

$$\Phi \wedge \Psi \ \equiv \ \forall x \,.\, (x \equiv_2 0 \to \Phi) \wedge (x \equiv_2 1 \to \Psi), \qquad \Phi \vee \Psi \ \equiv \ \exists x \,.\, (x \equiv_2 0 \to \Phi) \wedge (x \equiv_2 1 \to \Psi);$$

where the variable x above does not occur in Φ nor in Ψ.

References

1 László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complex.*, 1:3–40, 1991. doi:10.1007/BF01200056.

2 José L. Balcázar, Antoni Lozano, and Jacobo Torán. *The Complexity of Algorithmic Problems on Succinct Instances*, pages 351–377. Springer, 1992. doi:10.1007/978-1-4615-3422-8_30.

3 Leonard Berman. The complexity of logical theories. *Theor. Comput. Sci.*, 11(1):71–77, 1980. doi:10.1016/0304-3975(80)90037-7.

4 Laura Bozzelli, Alberto Molinari, Angelo Montanari, and Adriano Peron. On the complexity of model checking for syntactically maximal fragments of the interval temporal logic HS with regular expressions. In *International Symposium on Games, Automata, Logics, and Formal Verification, GandALF*, volume 277 of *EPTCS*, pages 31–45, 2017. doi:10.4204/EPTCS.256.3.

5 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

6 Dmitry Chistikov and Christoph Haase. On the Power of Ordering in Linear Arithmetic Theories. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 168 of *LIPIcs*, pages 119:1–119:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.119.

7 Christian Choffrut and Achille Frigeri. Deciding whether the ordering is necessary in a Presburger formula. *Discret. Math. Theor. C.*, 12(1):21–38, 2010. URL: http://dmtcs.episciences.org/510.

8 Stephen A. Cook. The complexity of theorem-proving procedures. In *Symposium on Theory of Computing, STOC*, pages 151–158, 1971. doi:10.1145/800157.805047.

9 Adrian W. Dudek. An explicit result for primes between cubes. *Functiones et Approximatio Commentarii Mathematici*, 55(2):177–197, 2016.

10 Raul Fervari and Alessio Mansutti. Modal logics and local quantifiers: A zoo in the elementary hierarchy. In *Foundations of Software Science and Computation Structures, FoSSaCS*, volume 13242 of *Lecture Notes in Computer Science*, pages 305–324, 2022. doi:10.1007/978-3-030-99253-8_16.

11 Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 122–135, 1998. doi:10.1007/978-3-7091-9459-1_5.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

13 Erich Grädel. Dominoes and the complexity of subclasses of logical theories. *Ann. Pure Appl. Log.*, 43(1):1–30, 1989. doi:10.1016/0168-0072(89)90023-7.

14 Christoph Haase and Alessio Mansutti. On deciding linear arithmetic constraints over p-adic integers for all primes. In *Mathematical Foundations of Computer Science, MFCS*, volume 202 of *LIPIcs*, pages 55:1–55:20, 2021. doi:10.4230/LIPIcs.MFCS.2021.55.

15 Lane A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989. doi:10.1016/0022-0000(89)90025-1.

16 Albert E. Ingham. On The Estimation Of $N(\sigma, T)$. *Q. J. Math.*, os-11(1):201–202, January 1940. doi:10.1093/qmath/os-11.1.201.

17 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

18 Dexter Kozen. *Theory of Computation.* Texts in Computer Science. Springer, 2006.

19 Markus Lohrey. Model-checking hierarchical structures. *J. Comput. Syst. Sci.*, 78(2):461–490, 2012. Games in Verification. doi:10.1016/j.jcss.2011.05.006.

20 Antoni Lozano and José L. Balcázar. The complexity of graph problems fore succinctly represented graphs. In *Graph-Theoretic Concepts in Computer Science, WG*, volume 411 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 1990. doi:10.1007/3-540-52292-1.

21 Martin Lück. Canonical models and the complexity of modal team logic. In *Computer Science Logic, CSL*, volume 119 of *LIPIcs*, pages 30:1–30:23, 2018. doi:10.4230/LIPIcs.CSL.2018.30.

22 Harry G. Mairson. A simple proof of a theorem of Statman. *Theor. Comput. Sci.*, 103(2):387–394, 1992. doi:10.1016/0304-3975(92)90020-G.

23 Alessio Mansutti. Notes on kAExp(pol) problems for deterministic machines. *CoRR*, abs/2110.05630, 2021. arXiv:2110.05630.

24 Albert R. Meyer. The inherent computational complexity of theories of ordered sets. In *Proceedings of the International Congress of Mathematicians*, volume 2, pages 477–482, 1974.

25 Alberto Molinari. *Model checking: the interval way.* PhD thesis, Università degli Studi di Udine, 2019. arXiv:1901.03880.

26 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

27 Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Inf. Control.*, 71(3):181–185, 1986. doi:10.1016/S0019-9958(86)80009-2.

28 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt, 1929. In *Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101.

29 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

30 Richard Statman. The typed λ-calculus is not elementary recursive. *Theor. Comput. Sci.*, 9(1):73–81, 1979. doi:10.1016/0304-3975(79)90007-0.

31 user4625. Complexity class $\mathrm{NEXP}^{\mathrm{NP}}$. Theoretical Computer Science Stack Exchange, 2011. URL: https://cstheory.stackexchange.com/q/6001.

32 Helmut Veith. Succinct representation, leaf languages, and projection reductions. *Inf. Comput.*, 142(2):207–236, 1998. doi:10.1006/inco.1997.2696.

33 Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976. doi:10.1016/0304-3975(76)90062-1.

On Dynamic $\alpha + 1$ Arboricity Decomposition and Out-Orientation

Aleksander B. G. Christiansen
Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Denmark

Jacob Holm
Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Eva Rotenberg
Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Denmark

Carsten Thomassen
Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Denmark

Abstract

A graph has arboricity α if its edges can be partitioned into α forests. The dynamic arboricity decomposition problem is to update a partitioning of the graph's edges into forests, as a graph undergoes insertions and deletions of edges. We present an algorithm for maintaining partitioning into $\alpha + 1$ forests, provided the arboricity of the dynamic graph never exceeds α. Our algorithm has an update time of $\tilde{O}(n^{3/4})$ when α is at most polylogarithmic in n.

Similarly, the dynamic bounded out-orientation problem is to orient the edges of the graph such that the out-degree of each vertex is at all times bounded. For this problem, we give an algorithm that orients the edges such that the out-degree is at all times bounded by $\alpha + 1$, with an update time of $\tilde{O}(n^{5/7})$, when α is at most polylogarithmic in n. Here, the choice of $\alpha + 1$ should be viewed in the light of the well-known lower bound by Brodal and Fagerberg which establishes that, for general graphs, maintaining only α out-edges would require linear update time.

However, the lower bound by Brodal and Fagerberg is non-planar. In this paper, we give a lower bound showing that even for planar graphs, linear update time is needed in order to maintain an explicit three-out-orientation. For planar graphs, we show that the dynamic four forest decomposition and four-out-orientations, can be updated in $\tilde{O}(n^{1/2})$ time.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graphs, bounded arboricity, data structures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.34

Funding *Aleksander B. G. Christiansen*: Partially supported by the VILLUM Foundation grant 37507 "Efficient Recomputations for Changeful Problems".
Jacob Holm: Partially supported by the VILLUM Foundation grant 16582, "BARC"
Eva Rotenberg: Partially supported by Independent Research Fund Denmark grants 2020-2023 (9131-00044B) "Dynamic Network Analysis" and 2018-2023 (8021-00249B) "AlgoGraph", and the VILLUM Foundation grant 37507 "Efficient Recomputations for Changeful Problems".
Carsten Thomassen: Partially supported by Independent Research Fund Denmark grant 2018-2023 (8021-00249B) "AlgoGraph".

Acknowledgements We thank Irene Parada for helpful discussions.

1 Introduction

Dynamic graphs have been the subject of much study. Here one is typically interested in maintaining some property of or information about the graph, as edges of the graph are inserted and deleted. Sometimes one studies more restricted classes of graphs with more structure

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 34; pp. 34:1–34:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the hopes of improving algorithmic performance. The existence of efficient algorithms for testing planarity of a fully-dynamic graph [17, 24] motivates efforts to dynamically maintain well-known properties of planar graphs, such as e.g. bounded out-degree orientations and colourings.

The *arboricity* of a graph G is the smallest number α such that α forests can cover the edges of G. Planar graphs have arboricity at most 3. If a forest is rooted arbitrarily, and all of the edges in the forest are oriented towards the roots, then every vertex has out-degree 1. In particular, this implies that the edges of a planar graph resp. graphs of arboricity α can be oriented such that no vertex has out-degree more than 3 resp. α . We call an orientation of a graph such that no vertex has more than k out-edges a *k-bounded out-orientation.* Since there exists planar graphs on n vertices with more than $2n$ edges, it follows that there exists planar graphs for which a 3-bounded out-orientation is the lowest out-orientation possible.

In light of the above, we ask the question: can one efficiently maintain a 3-bounded out-orientation of a dynamic planar graph? Here the aim is to be more efficient than the fastest static algorithm – running in linear time – as one could construct a dynamic algorithm by statically recomputing a new solution after every update.

It turns out that it is not possible to improve upon this; at least not if we want to maintain an *explicit* orientation. Here, one has to store the orientation of every edge in memory opposed to an *implicit* orientation, where one is allowed to compute the orientation of an edge at query-time. More precisely we show that any algorithm maintaining an explicit 3-bounded out-orientation of a dynamic planar graph must spend $\Omega(n)$ update time, even amortised. For the broader class of graphs with bounded arboricity, Brodal & Fagerberg [9] showed that any algorithm maintaining an explicit α-bounded out-orientation must spend $\Omega(n)$ update time, even amortised. However for $\alpha = 3$ their example is far from planar. Our lower bound shows that the same bounds indeed hold for planar graphs.

In light of this negative result, it is natural to ask if one can do better if a little slack on the number of out-edges is allowed. We show that this is indeed the case, as we provide an algorithm maintaining a 4-bounded out-orientation of a dynamic planar graph with $\tilde{O}(\sqrt{n})$ amortised update time. In fact, this generalises to maintaining an $(\alpha + 1)$-bounded out-orientation of a dynamic graph whose arboricity never exceeds α through-out the entire update sequence. Here the algorithm has an amortised update-time of $\tilde{O}(n^{5/7})$. Here $\tilde{O}$ hides α and $\log n$ factors, so one should think of $\alpha = O(1)$.

An *arboricity decomposition* of a graph is a decomposition of the graph's edges into forests. We show how to dynamically maintain a 4-arboricity decomposition of a dynamic planar graph with $\tilde{O}(\sqrt{n})$ amortised update time. This also generalises to graphs of bounded degree α: here we present an algorithm maintaining an $(\alpha + 1)$ arboricity decomposition with an amortised update-time of $\tilde{O}(n^{3/4})$.

The presented algorithms all follow the same idea: we show how to update the out-orientation or arboricity decomposition in such a way that 1) every update only uses very little of the extra slack provided by having one more out-edge or one more forest available and 2) the update time scales with the amount of extra slack used. Combining these two properties allow us to truncate the update algorithm if it runs for too long and instead statically recompute an optimal solution.

1.1 Results

We consider dynamic graphs on n vertices undergoing a sequence of updates such that the arboricity of the graph at all times is bounded by α. We refer to such a sequence of updates as an α preserving sequence of updates.

We show the following theorem for maintaining an $\alpha + 1$ out-orientation.

▶ **Theorem 1.** *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α-bounded out-degree orientation of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$-bounded out-degree orientation with an amortised insertion time of $O(\sqrt{\alpha \cdot S(\alpha, n)})$, and a worst-case deletion time of $O(1)$.*

Specifying a black box algorithm gives the following corollary:

▶ **Corollary 2.** *Given an initially empty dynamic planar graph undergoing edge insertions and deletions, there is an algorithm that maintains a 4-bounded out-degree orientation with an amortised insertion time of $O(\sqrt{n})$, and a worst-case deletion time of $O(\log n)$.*

Proof. Chrobak & Eppstein gave a linear-time algorithm for computing a 3-bounded out-degree orientation in planar graphs [12]. Now applying Theorem 1 yields the result. ◀

For bounded arboricity graphs, applying the fastest static algorithm [31] gives an amortised update time of $\tilde{O}(n^{5/7})$. We show the following theorem for maintaining an $\alpha + 1$ arboricity decomposition:

▶ **Theorem 3.** *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α arboricity decompostion of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$ arboricity decomposition with an amortised insertion time of $O(\alpha\sqrt{S(\alpha, n)} \log n)$, and a worst-case deletion time of $O(\log n)$.*

Specifying black box algorithms gives the following corollaries:

▶ **Corollary 4.** *Given an initially empty dynamic planar graph undergoing edge insertions and deletions, then there exists an algorithm maintaining a 4-arboricity decomposition with an amortised insertion time of $O(\sqrt{n} \log n)$, and a worst-case deletion cost of $O(\log n)$.*

Proof. Chrobak & Eppstein also showed how to compute a 3-forest partition in linear time in planar graphs [12]. ◀

For bounded arboricity graphs, applying the fastest static algorithm [18, 19] gives an amortised update time of $\tilde{O}(n^{3/4})$. Finally, we show a lower bound for maintaining explicit 3-bounded out-orientations in dynamic planar graphs:

▶ **Theorem 5.** *Let $\mathcal{A}$ be an algorithm explicitly maintaining a 3-bounded out-degree orientation of an n-vertex planar graph under insertion and deletion of edges. Then there exists a sequence of updates taking $\Omega(n)$ amortised time per update.*

Note that some results rely on ideas similar to those that appeared in the master's thesis by A. B. G. Christiansen [10].

1.2 Related Work

Dynamic Planar Graphs. Dynamic planar graphs have been studied both in the incremental (edge-insertion only) [13, 25, 35, 38] and fully-dynamic (insertion/deletion) setting [16, 17, 20, 24, 26].

Table 1 Overview of dynamic algorithms for maintaining out-orientations. This table is inspired by a similar table in [11].

Reference	Out-degree	Update time	α preserving seq.
Brodal & Fagerberg [9]	$2(\alpha + 1)$	$\tilde{O}(\log n)$ am.	yes
Kopelowitz et al. [27]	$\beta\alpha + \log_\beta n$	$\tilde{O}(\beta^2 + \beta \log n)$	no
He et al. [22]	$O(\alpha\sqrt{\log n})$	$O(\sqrt{\log n})$ am.	yes
Berglin & Brodal [5]	$O(\alpha + \log n)$	$O(\log n)$	no
Henzinger et al. [23]	40α	$O(\log^2 n)$ am.	no
Kowalik [29]	$O(\alpha \log n)$	$O(1)$ am.	yes
Christiansen & Rotenberg [11]	$\alpha + 2$	$\tilde{O}(\log^3 (n)))$	yes
New (Thm. 1)	$\alpha + 1$	$\tilde{O}(n^{5/7})$	yes

Bounded Out-Orientations. For the more general class of graphs with bounded arboricity, the out-orientation problem is well studied – both from a dynamic view-point [9, 27, 37, 29, 5] and a static view-point [18, 34, 6, 1]. For dynamic arboricity-bounded graphs, Brodal & Fagerberg gave an algorithm maintaining a $2(\alpha + 1)$-bounded out-orientation with amortised $O(\alpha + \log n)$ update time. There also exists an algorithm [11] for maintaining an $(\alpha + 2)$-bounded out-orientation with worst-case update time in $O(\text{poly}(\log n, \alpha))$.See Table 1 for an overview.

From a static view-point, the fastest algorithms have running time $\tilde{O}(m^{10/7})$ [31] and $\tilde{O}(m\sqrt{n})$ [30]. Furthermore, Kowalik [28] also gave an algorithm computing a $(\alpha + 1)$ out-orientation in $\tilde{O}(m)$ time. Specialising to the planar case, where the graphs are assumed to be planar at all times, the lowest out-degree one can get in the dynamic setting while still achieving sublinear update time becomes 5 with update time in $O(\text{poly}(\log n))$. In the static case, Chrobak & Eppstein showed how to compute a 3 out-orientation in linear time [12].

Arboricity Decompositions. There has been a lot of work dedicated to computing an arboricity decomposition [18, 19, 14, 34]. The state-of-the-art for computing exact arboricity decompositions run in $\tilde{O}(m^{3/2})$ time [18, 19]. There also exists approximation algorithms. There is a 2-approximation algorithm [3, 15] as well as an algorithm for computing an $\alpha + 2$ arboricity decomposition in near-linear time [7]. From the dynamic side, Bannerjee et al. [4] give a dynamic algorithm for maintaining the current arboricity. The algorithm has a near-linear update time. They also provide a lower bound of $\Omega(\log n)$ for dynamically maintaining arboricity. Henzinger et al. [23] provide a dynamic algorithm for maintaining a $2\alpha'$ arboricity decomposition, given access to any black box dynamic α' out-degree orientation algorithm. Combining this technique with the results from Table 1 yields different trade-offs. Finally there also exists an algorithm maintaining an $(\alpha + 2)$ arboricity decomposition with $O(\text{poly}(\log(n), \alpha))$ update-time [11].

Specialising to the planar case, the last of the algorithms above yields a sublinear update time dynamic algorithm for computing a 5 arboricity decomposition. In the static case, Chrobak & Eppstein also showed how to compute a 3-forest partition in linear time [12].

1.3 Paper Outline

In Section 2 we recall preliminaries. Section 3 is dedicated to Theorem 1. In Section 4 we show Theorem 3. Finally in Section 5, we prove Theorem 5.

2 Preliminaries & Notation

We follow standard graph-terminology. We say that a graph $G = (V, E)$ is *k-degenerate* if every subgraph of G has a vertex of degree at most k. If the edges of G receives an orientation, we let the *out-edges* of a vertex v be the edges oriented away from v. The *out-degree* of v is then the number of out-edges incident to v.

Nash-Williams showed the following density formula for graphs of arboricity α:

▶ **Theorem 6** ([33, 32]). *Let G be a graph with no loops. Then*

$$\alpha(G) = \left\lceil \max_{J \subset G, |V(J)| \geq 2} \frac{|E(J)|}{|V(J)| - 1} \right\rceil$$

A plane graph is a planar graph together with an embedding into the plane such that no edges cross. For plane graphs we have the usual notion of faces. A *triangulation* is a maximal planar subgraph (wrt. the edges). If the triangulation is equipped with an embedding, we call it a *plane triangulation.* For planar graphs, we have the following well-known theorem:

▶ **Theorem 7** (Euler's Theorem). *If G is a connected plane graph with n vertices, m edges and f faces, then $n - m + f = 2$. In particular, if G is a planar triangulation, then $m = 3n - 6$.*

3 Low Out-orientations

In this section, we present a dynamic algorithm that maintains an $(\alpha + 1)$-bounded out-degree orientation. Suppose we are given an initially empty dynamic graph G on n vertices undergoing an arboricity α preserving sequence of updates. Then the algorithm has an update time of $\tilde{O}(\sqrt{S(\alpha, n)})$ provided that it is given access to a black box static algorithm for computing an α-bounded out-degree orientation in $O(S(\alpha, n))$ time.

The idea behind the algorithm is the following: suppose we are given a graph G, an edge $e = uv \in G$ and an $\alpha + 1$ orientation of $G - e$. Then we can extend the orientation of $G - e$ to an $\alpha + 1$ orientation of G in the following way. Find a minimal oriented path P beginning at u that ends at a vertex w with out-degree at most α. We will refer to such a path as an *augmenting path.* After finding P, we reorient all edges along the path. Finally, we make uv an out-edge of u. This yields an $\alpha + 1$ orientation of G since the only vertex to have its out-degree increased is w, and w was specifically chosen to have out-degree $< \alpha + 1$.

In order to find P, we look for w by doing a breadth-first-search (BFS) beginning from u. In Lemma 8, we show that such a BFS always succeeds and that it takes time proportional to the number of vertices visited. Finally, we show that if the BFS visits too many vertices, we have witnessed enough updates to be able to truncate the search and recompute an α orientation.

The Algorithm. Whenever an edge is deleted, we can simply delete it in the graph, so we focus on insertions. Suppose $e = uv$ is inserted. Either u has out-degree $< \alpha + 1$ and $P = u$ is an augmenting path or u has out-degree $\alpha + 1$ in which case, we push u's neighbours to a queue Q and mark them as visited. Now we can recursively visit vertices in Q by pushing the un-visited neighbours of vertices with degree $\alpha + 1$ to Q, and terminating with an augmenting path if we locate a vertex with out-degree $< \alpha + 1$. In the end of the update algorithm, we mark all visited vertices as un-visited. The search for P has the following properties:

▶ **Lemma 8.** *Suppose that $G - uv$ is given an $(\alpha + 1)$-bounded out-degree orientation, and we begin a search from u. Then after visiting t vertices the following holds:*

1. *If no vertex with out-degree $< \alpha + 1$ has been visited, then Q is not empty.*
2. *The algorithm has spent $O(\alpha \cdot t)$-time.*

Proof. We begin by showing that 1) holds. Suppose for contradiction that the algorithm has visited all vertices pushed to Q without finding a vertex with out-degree $< \alpha + 1$. Let J be the set of visited vertices. Since the algorithm pushes all new out-neighbours of a vertex v to Q, when visiting v, it must be the case that all out-edges of vertices in J go to other vertices in J. Since the algorithm did not find a vertex with out-degree $< \alpha + 1$, all vertices of J have out-degree $\alpha + 1$, and as a result, there must be at least $(\alpha + 1)|J|$ edges between vertices in J. But then, $\frac{|E(J)|}{|J|-1} = \frac{(\alpha+1)|J|}{|J|-1} > \alpha + 1 > \alpha$, contradicting Theorem 6.

Now, we show that 2) also holds. After visiting t vertices, the algorithm has traversed every out-edge out of the visited vertices when pushing new out-neighbours to Q. The time needed to perform these steps is $O(1)$ per visited vertex and $O(1)$ per out-edge leaving a visited vertex. Since every visited vertex has out-degree $\alpha + 1$ by assumption, the total time spent in the search is $O(\alpha \cdot t)$. ◀

▶ Remark 9. Lemma 2 in [9] implies that P has length $O(\alpha \log(n))$.

Now we make the following observation:

▶ **Observation 10.** *Inserting an edge e into G increases the number of vertices with out-degree $\alpha + 1$ by at most one.*

Proof. The only vertex that has its out-degree increased during an insertion is the final vertex of the augmenting path. ◀

In particular this means that if we visit t vertices when searching for P, then we must have witnessed at least t insertions, since the last time we statically re-computed an α-bounded out-orientation. As such, we arrive at Theorem 1 (restated below for convenience):

▶ **Theorem 11** (Theorem 1). *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α-bounded out-degree orientation of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha+1)$-bounded out-degree orientation with an amortised insertion time of $O(\sqrt{\alpha \cdot S(\alpha, n)})$, and a worst-case deletion time of $O(1)$.*

Proof. The algorithm clearly maintains an $(\alpha + 1)$-bounded out-degree orientation. All that remains is to analyse the amortised cost of running the black box algorithm. To this end, say a vertex is *bad*, if it has out-degree $\alpha + 1$. Suppose that we have witnessed i insertions since the last time that the static black-box algorithm was run, and suppose furthermore that inserting e_{i+1} causes the insertion search to visit $\sqrt{S(\alpha, n)/\alpha}$ bad vertices. Then Observation 10 implies that $i \geq \sqrt{S(\alpha, n)/\alpha}$, and so setting aside $O(\sqrt{\alpha \cdot S(\alpha, n)})$ credit for each insertion, ensures that at least $O(S(\alpha, n))$ credit is stored, when the black box algorithm is run. Furthermore, by Lemma 8 each truncated search spends no more than $O(\alpha \cdot \sqrt{S(\alpha, n)/\alpha})$-time, yielding a total amortised insertion time in $O(\sqrt{\alpha \cdot S(\alpha, n)})$, as claimed. To delete an edge, we may remove it from the graph in constant time (assuming that we are given a pointer to the edge as part of the query). ◀

4 Low Arboricity Decompositions

In this section, we present a dynamic algorithm that maintains an $(\alpha + 1)$-arboricity decomposition. The setup is the same as in the previous section: we assume that we are given an initially empty dynamic graph G on n vertices undergoing an arboricity α preserving sequence of updates. The algorithm has an update time of $\tilde{O}(\sqrt{S(\alpha, n)})$ provided that it is given access to a black box static algorithm for computing an α-arboricity decomposition in $O(S(\alpha, n))$ time.

The idea behind this algorithm is the same as in the previous section. We give an algorithm based on a search procedure with slow update time, and arrive at the final algorithm via truncation of the search. The main difficulty of turning the algorithm from the previous section into an algorithm maintaining an arboricity decomposition is that it is not clear how to direct a search to obtain a sequence of forest alterations that allows one to accommodate the insertion of an edge.

We will now present how to accommodate the edge insertion using techniques that are inspired by [18] and [21]. To this end, suppose we are given a graph G, an edge $e = uv \in G$ and an $\alpha + 1$ arboricity decomposition of $G - e$. We are then interested in extending this to an $\alpha + 1$ arboricity decomposition of G. If e cannot be added to forest F_i, it is because u and v are connected by a path T_i^1 in F_i. If this is the case for all i, then we must move one of the edges in T_j^1 for some j to be able to put e into F_j. If this is not possible for any j, it is because all vertices on T_j^1 sit in the same tree in all forests. So for all j we can let T_j^2 be the smallest tree in F_j spanning the vertices $\bigcup_{i=1}^{\alpha+1} V(T_i^1)$. Continuing like this yields trees T_i^r (see Figure 1). We cannot continue this construction indefinitely since we cannot partition G's edges into $\alpha + 1$ spanning trees without contradicting the fact that G has arboricity α. Hence, there must exist a largest k for which T_j^k has no movable edges for all choices of j. For an edge $e' \neq e$, we let the *rank* $r(e')$ of e' be the smallest r such that $e' \in T_i^r$ for some i that we denote $t(e')$. We set $r(e) = 0$. Furthermore, we define $T^r = \bigcup_i T_i^r$. Similar to [18, 21], we can now define an *augmenting sequence* of edges that we can move between forests to accommodate an insertion of e:

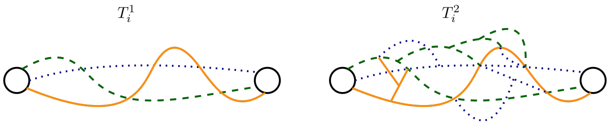


Figure 1 An illustration of the first two layers of the forests. The forests are represented by different colours.

▶ **Definition 12.** *Suppose G is a graph, $e_0 \in E(G)$ an edge in G, and that we are given an $\alpha + 1$ arboricity decomposition $F_1, \ldots, F_{\alpha+1}$ of $G - e_0$. Then an augmenting sequence of edges is a sequence of edges $e_0, e_1, \ldots, e_k$ satisfying the following conditions:*

1. *For all $i \geq 0$, the rank of e_i satisfies $r(e_i) \leq i$.*
2. *For all $i \geq 1$, if $e_{i-1} = xy$, then x and y sit in different trees in $F_{t(e_i)} - e_i$.*
3. *There exists some $j \neq k$ such that $F_j \cup e_k$ is a forest.*

Given such a sequence of edges $e_1, \ldots, e_k$, we can extend the arboricity decomposition of $G - e$ to contain e. Indeed, we move e_k from $F_{t(k)}$ to F_j. Now $e_1, \ldots, e_{k-1}$ is an augmenting sequence. Eventually, e can be inserted into $F_{t(e_1)}$.

We shall search for an augmenting sequence differently than in [18]. Here, when visiting an edge e' belonging to forest F_i, for $j \neq i$ they en-queue the entire fundamental cycle in $F_j \cup e_i$. However, this cycle could possibly have length $\Omega(n)$. Instead, we visit the paths blocking e' one edge at time, only en-queuing the visited edge. This ensures that at all times, the time spent searching for an augmenting sequence is proportional to the number of vertices that we have processed. This property is key in ensuring that we can afford to truncate our search and run a static algorithm, if we do not identify an augmenting sequence fast.

The algorithm. Similar to the previous section, we first describe a slow, search–based algorithm, and then we obtain a faster algorithm via truncation. We accommodate edge deletions by simply deleting the edge from the dynamic forest data structure, it resides in. To handle an insertion of an edge $e = uv$, we construct a short augmenting sequence as follows. Beginning with e, we process an edge by trying to insert it into all forests. If this is not possible, u and v must be connected by a path in F_i for all i. For each i, beginning with $i = 1$, we then try to move every edge on this path in F_i. If we fail to do so, it means that there is no augmenting sequence ending with an edge in $T^1 = \bigcup_i T_i^1$. Now we search for an augmenting sequence ending with an edge in T^2. If we fail to locate such an augmenting sequence, we try to find one ending with an edge in T^3 and so on. To do so for T^r, having already tried all edges in T^{r-1}, we try to move all edges blocking an edge in T^{r-1} i.e. we try to move edges blocking the previously explored edges. When we try to move such an edge, but fail to do so, it is because the endpoints of these edges are connected by blocking paths in all forests. Conceptually we would like to push these blocking paths to a queue as we encounter them, in order to remember them. This is relevant, since we might have to visit them later to try to locate an augmenting sequence. However, these paths might have length $\Omega(n)$, and, similar to the last section, we would like the time spent during a search to depend on the number of visited vertices. Hence, in practice, we traverse the blocking paths one by one. Whenever we visit an edge f on a blocking path in a forest F_j, we try to move it to another forest. If this is not possible, we push f to a queue Q, store that f is blocking e by saving a value $b(f) = e$, and mark both endpoints of f as visited in F_j. Now we can always recover the edges on the paths blocking f one-by-one using standard dynamic forest data-structures. When we are done trying to move edges from the previous set of blocking paths, we recursively try to move the un-processed edges blocking $e' = \texttt{pop}(Q)$. Note that whenever a blocking path reaches a previously visited edge, we terminate that blocking path and push no further edges of it to Q. The nature of the search ensures that the blocking paths form trees in each forest, and therefore no blocking edge is left unvisited by this early termination. Indeed, one of the endpoints of the blocking path was already in the tree, and so as soon as the path connects to this tree we can be certain that the rest of the edges on the path have been processed. If we encounter an edge g that we can move to a new forest, we extract an augmenting sequence by moving the edge and then recursively moving the edge it was blocking. We have the following claim:

▷ Claim 13. Let e and g be as above. We have that $e = b^{r(g)}(g), \ldots, b(b(g)), b(g), g$ is an augmenting sequence, where $b^k(g)$ is the edge obtained by following the blocked edge k steps back.

Proof. Observe that we process all edges of rank i before processing any edges of rank $i + 1$. In particular, we only visit edges of rank $i + 1$ when processing edges of rank i. Therefore, for all processed edges h, we have that $r(b(h)) = r(h) - 1$. Hence, $b^{r(g)}(g) = e$, as e is the only edge of rank 0. It is now easy to verify that the conditions of Definition 12 hold. ◁

By storing each vertex in a data structure for maintaining dynamic forests for instance top-trees [2] or link-cut trees [36], we can determine whether a blocking path exists, and if it does, we can traverse it spending $O(\log n)$ time to find the path and $O(1)$ time per visited edge on the path. Thus the search satisfy the following properties:

▶ **Lemma 14.** *Suppose that we have an $\alpha + 1$ arboricity decomposition of $G - uv$, and we begin a search to extend it to an $\alpha + 1$ arboricity decomposition of G. Then after visiting t vertices the following holds:*

1. *If no moveable edge has been located, then Q is not empty.*
2. *The algorithm has spent $O(\alpha^2 \cdot t \cdot \log n)$-time.*

Proof.

1. Let E denote the set of edges, visited by the algorithm. Every time we unsuccessfully try to move an edge $e = xy$, we conceptually push all edges on the unique x to y path in F_i to Q for all i. Thus, the blocking paths necessarily form trees in each forest. In particular, this means that if Q is empty, the graph formed by all the visited edges, $J = G[E]$, will be a tree if restricted to F_i for all i. Indeed, suppose it is not true, and that the restriction of J to F_j is not a tree. Then we must have visited some first edge leaving the component, containing u, of the restriction of J to F_j for which we did not explore a blocking path in F_j, contradicting the fact that Q is empty. This means, in particular, that $|E(J)| \geq (\alpha + 1)(|V(J)| - 1)$, and hence J contradicts Theorem 6, as in the proof of Lemma 8.
2. Since the algorithm has visited t vertices, it cannot have visited more than $\alpha \cdot t$ edges. Otherwise, the t visited vertices form a subgraph with a density contradicting Theorem 6. Each time the algorithm visits an edge, it spends only $O(\alpha \log n)$-time. Indeed, it takes $O(\alpha \log n)$-time to check if the edge can be moved and $O(1)$-time to push it to Q. Finally, we can update pointers to the blocked edge with only $O(1)$ overhead per visit. Since the algorithm has visited no more than $\alpha \cdot t$ edges, 2) follows. ◀

Furthermore, we make the following observations:

▶ **Observation 15.** *If the search phase is unsuccessful after visiting t vertices, then every vertex visited by the algorithm during the search phase must belong to the same tree in every forest F_i. In particular, this means that $|E(F_{\alpha+1})| \geq t - 1$.*

Proof. We prove the observation by contraposition. Suppose that we initially tried to insert uv. If at some point we try to move an edge $e = xy \in F_j$ with endpoints belonging to different trees in F_k, $k \neq j$, then e could be moved to F_k rendering the search phase successful. In particular, this means that if $u \in T \subset F_i$ and there is some other vertex w in a different tree T' in F_i that we also visit, then at some point, we must have tried to move an edge e' with only one endpoint in T. But then e' could have been moved to F_i, contradicting the fact that the search was unsuccessful. ◀

▶ **Observation 16.** *Augmenting along an augmenting sequence $e_0, e_1, \ldots, e_k$ increases the size of the forest that e_k is moved to by one. All other forests remain the same size.*

Proof. We prove the statement by induction on k. The induction basis is clear, so we move to the induction step. After moving e_k from $F_{t(e_k)}$ to some forest F_i, the size of F_i is increased by one, the size of $F_{t(e_k)}$ is decreased by one, and all other forests have the same size. Now $e_0, e_1, \ldots, e_{k-1}$ is an augmenting sequence, so we can apply induction on it. This sequence increases the size of $F_{t(e_k)}$ by exactly one, and all other forests remain the same size. Hence, we arrive at the statement. ◀

Since a deletion never increases the size of a forest, the two above observations together imply that choosing the truncation $t = \sqrt{\frac{S(\alpha,n)}{\alpha^2}} + 1$ means that we witness at least $\sqrt{\frac{S(\alpha,n)}{\alpha^2}}$ insertions between any two runs of the static black box algorithm. Since we can extend the black box algorithm to maintain the dynamic forests using only $O(\log n)$ overhead per operation, the total time needed for each run is in $O(S(\alpha, n) \log n)$. Combining this with Lemma 14, we find that setting aside $O(\alpha\sqrt{S(\alpha, n)} \log n)$ per insertion yields a total amortised update time in $O(\alpha\sqrt{S(\alpha, n)} \log n)$. Hence:

▶ **Theorem 17** (Identical to Theorem 3). *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α arboricity decompostion of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$ arboricity decomposition with an amortised insertion time of $O(\alpha\sqrt{S(\alpha, n)}\log n)$, and a worst-case deletion time of $O(\log n)$.*

5 Dynamic 3-Out Orientations in planar graphs

In this section, we show that any dynamic algorithm maintaining a 3-bounded out-orientation of a dynamic planar graph can be forced to spend $\Omega(n)$ update time. The idea is to first consider a different problem: let G be a plane triangulation. Then G has an outer face xyz. Every vertex incident to the outer face is an *outer* vertex. All vertices that are not outer vertices are *inner* vertices. A 3-orientation of G is then an orientation of all edges incident to inner vertices such that all inner vertices have out-degree 3. Given a plane triangulation, we have the notion of a *diagonal flip*: a diagonal flip is the action of removing an edge xy incident on faces xyz and vxy and replacing it with the edge vz (see Figure 2). note that such a flip is only possible, if the edge vz does not already exist. The *flip distance* between two undirected graphs is then the minimum number of diagonal flips needed to go from one one graph to the other. We show that any algorithm explicitly maintaining a 3-orientation of a dynamic plane triangulation under diagonal flips of inner edges, can be made to spend linear update time. This is done by constructing two planar graphs of constant flip distance, but with unique 3-orientations differing on $\Omega(n)$ edges. By slightly generalising and by considering a graph containing multiple copies of this construction, we arrive at the explicit lower bound for maintaining 3-orientations in dynamic planar graphs.

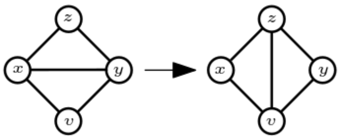


Figure 2 A diagonal flip.

3-Orientations. Brehm [8] showed that a plane triangulation has a unique 3-orientation if and only if it is *stacked* which is equivalent to being 3-degenerate. In fact this can be slightly generalised - and we will use this slight generalisation later on, when dealing with graphs where no embedding is specified. The proof of the generalisation is similar, so we only provide a sketch:

▶ **Lemma 18.** *Let G be a planar triangulation with a 3-bounded out-degree orientation O. Let x, y, z form a triangle in G, and let H be a component of $G - \{x, y, z\}$, such that the out-degree $d^+_G(v) = 3$ for all $v \in H$. Then the restriction of any 3-bounded out-degree orientation of G to all edges incident to H is unique if and only if $G[V(H) \cup \{x, y, z\}]$ is 3-degenerate.*

Proof (Sketch). A counting argument shows that all edges between H and x, y, z are oriented away from H. Indeed, $G[H \cup \{x, y, z\}]$ is planar and so contains at most $3(|H| + 3) - 6$ edges. 3 of these go between x, y, z, so the remaining $3|H|$ edges must be out-edges of vertices in H.

Next, we show that the restriction of any 3-bounded out-degree orientation O to H is unique iff it is acyclic. Indeed, suppose it is not acyclic. Then it has a directed cycle. Reversing the orientation along this cycle creates a new 3-bounded out-degree orientation with a different restriction. The other direction is as follows: suppose that there exists an orientation for which the restriction to H is acyclic, but that this restriction is not unique. Comparing two such restrictions gives an edge which is oriented differently in the two orientations. Now, since every point has out-degree 3 an endpoint cannot be incident to only one such edge, so there is a new edge - oriented differently by the two orientations - that one can follow. Continuing like this eventually gives you a directed cycle in H as, by above, one never reaches x, y, z. This is a contradiction.

The Lemma then follows by noting that having an acyclic 3-bounded out-degree orientation is equivalent to being 3-degenerate. Indeed, beginning at an arbitrary vertex in H and following incoming edges backwards ensures that one ends up in a source in H. This source has degree at most 3. We can remove this vertex and apply induction to see that any subgraph not containing this vertex also has a vertex of degree at most 3. The other direction follows, since the 3-degeneracy implies that in any non-empty subgraph $S \subset H$ one can always find a vertex $v \in S$ of degree 3 in $G[S \cup \{x, y, z\}]$. Beginning from H one can remove such a vertex and orient its incident edges so that it becomes a source. Continuing like this never creates cycles and therefore yields an acyclic 3-bounded out-degree orientation. ◀

As noted earlier, we give the lower bound by first considering explicit 3-orientations in plane graphs. We have the following lemma:

▶ **Lemma 19.** *Let $\mathcal{A}$ be an algorithm explicitly maintaining a 3-orientation of an n-vertex plane triangulation under diagonal flips. Then the flip operation can be made to spend $\Omega(n)$ update time, even when considering amortised complexity.*

Proof. Consider the following plane triangulation containing a path $s_1, s_2, \ldots, s_k$ of length $k = n - 5 = \Omega(n)$ (see Figure 3). The plane triangulation is 3-degenerate, and hence by Lemma 18, it has a unique 3-orientation: By diagonal flipping the edge uv and subsequently the edge s_1x, one gets a new plane triangulation. It is again 3-degenerate, and hence by Lemma 18 it has a unique 3-orientation. (see Figure 4). By diagonally flipping the same edges in the opposite order, one reclaims the original graph. The new 3-orientation has $\Omega(n)$ edges oriented differently compared to the original 3-orientation, but it only requires a constant number of diagonal flips to go between the two graphs. Hence, a constant number of flips forces $\mathcal{A}$ to change the orientation of $\Omega(n)$ edges, and thus, the update time for the diagonal flip operation must be $\Omega(n)$, even amortised, as one can force this update sequence as many times as desired. ◀

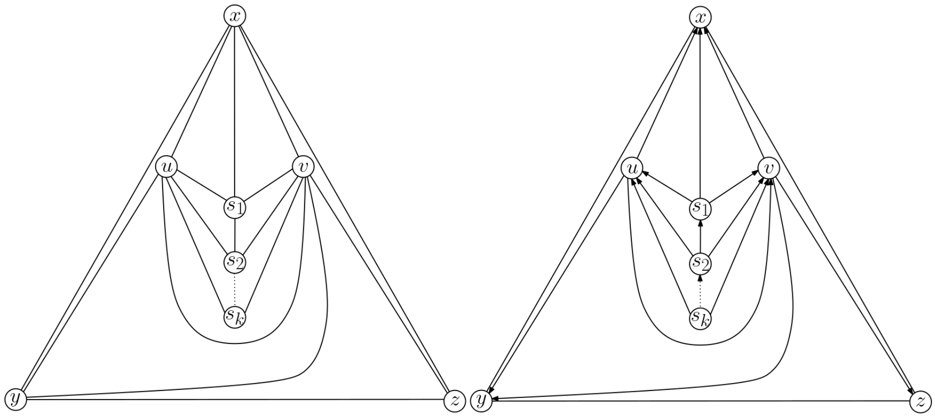


Figure 3 The plane triangulation along with its unique 3-orientation.

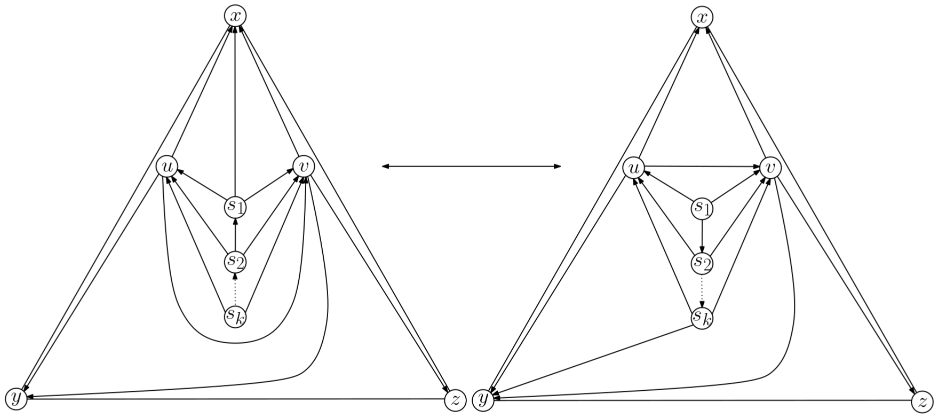


Figure 4 Going between two unique 3-orientations.

3-Bounded Out-Orientations. The goal now is to extend this lower bound to 3-bounded out-degree orientations of planar graphs under insertion/deletion of edges. There are only three things to consider before doing such an extension. Firstly, now we support the operations insertion/deletion of edges and not the diagonal flip. This is however a non-issue since a diagonal flip can be simulated by first deleting the edge and then inserting the other diagonal. Secondly, we now consider 3-bounded out-degree orientations and so outer vertices also have out-edges, and not all inner vertices are required to have out-degree 3. Lastly, the lower bound should apply not only to plane graphs where an embedding is chosen, but also to planar graphs. We deal with the last two points by using at least 13 copies of the graph from above. Then, in at least one of the copies, all inner vertices must have out-degree 3 - both before and after a sequence of updates. Hence, we can do the aforementioned updates in all 13 copies, and this will then ensure that at least one copy has to behave as in Lemma 19. Now, we show Theorem 5:

▶ **Theorem 20** (Identical to Theorem 5). *Let $\mathcal{A}$ be an algorithm explicitly maintaining a 3-bounded out-degree orientation of an n-vertex planar graph under insertion and deletion of edges. Then there exists a sequence of updates taking $\Omega(n)$ amortised time per update.*

Proof. Create a planar graph G by placing 13 copies of the graph P from the proof of Lemma 19 in the plane, and triangulating the outside arbitrarily. Let $n = |V(G)|$. For each copy P_i of P, we let the set I_i resp. O_i be the set of vertices that are inner resp. outer vertices of P_i, if P_i is embedded as in Lemma 19. In particular, for a specific copy of P, say P_i, the corresponding set I_i has size $|I_i| = \frac{n}{13} - 3 = \Omega(n)$. Since G is a plane triangulation, it follows from Euler's Theorem that $|E(G)| = 3n - 6$. This implies that at most 6 vertices of G can have out-degree strictly less than 3. Hence, in at least 7 of the 13 copies of P, every vertex in I has out-degree 3. Since the O vertices of each of these specific copies of P form a triangle, it follows from Lemma 18 that the edges incident to I must have the same orientation as in the plane embedding in Lemma 19 and that this orientation is unique.

Now, we simulate the flip sequence used in Lemma 19 in each copy. Doing this in all 13 copies only requires 26 insertions and 26 deletions in total. After these alterations, it is still the case that in at least 7 of the 13 copies of P every vertex in I has out-degree 3. Furthermore, since the O vertices of each of these specific copies of P form a triangle, it follows from Lemma 18 that the edges incident to I must have the same orientation as in the altered plane embedding in Lemma 19 and that this orientation is unique. At least one copy P_i of P has out-degree 3 at every vertex in I_i in both the orientation before and in the orientation after the update sequence. Consequentially, $\mathcal{A}$ must have reoriented at least $|I_i| - 3 = \Omega(n)$ edges during the update sequence. The update sequence consists of only a constant number of updates, and it can be reversed by first deleting uv and re-inserting it across the opposite diagonal in each copy, and then doing the same for $s_k y$ in every copy of P. Each reversal of the update sequence, requires only a constant number of updates, but forces $\mathcal{A}$ to change at least $\Omega(n)$ edge-orientations. It follows that there exists a sequence of updates taking $\Omega(n)$ amortised time per update. ◀

6 Conclusion

We have shown how to dynamically maintain 4-bounded out-orientations and 4-arboricity decompositions with sublinear update time. We extended these algorithms to compute $\alpha + 1$-bounded out-orientations and arboricity decompositions in α-arboricity bounded dynamic graphs. Finally, we also showed that maintaining a 3-bounded out-orientation of a planar graph explicitly, must take $\Omega(n)$ update time, even amortised. This extends the explicit lower bound of Brodal & Fagerberg to the planar case.

It would be interesting to see, if one can maintain $\alpha + 1$ out-edges or forests in sub-polynomial time. Results of Brodal & Fagerberg [9] and Harris et al. [21] show that the problems of maintaining $\alpha + 1$ out-edges resp. forests have $O(\alpha \log n)$ recourse – how close to this worst-case recourse can one get the update times?

References

1 Oswin Aichholzer, Franz Aurenhammer, and Günter Rote. *Optimal graph orientation with storage applications*. SFB-Report , SFB 'Optimierung und Kontrolle'. TU Graz, 1995. Reportnr.: F003-51.

2 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. `doi:10.1145/1103963.1103966`.

3 Srinivasa Rao Arikati, Anil Maheshwari, and Christos D. Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discret. Appl. Math.*, 78(1-3):1–16, 1997. `doi:10.1016/S0166-218X(97)00007-3`.

4 Niranka Banerjee, Venkatesh Raman, and Saket Saurabh. Fully dynamic arboricity maintenance. In *Computing and Combinatorics – 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2019. `doi:10.1007/978-3-030-26176-4_1`.

5 Edvin Berglin and Gerth Stolting Brodal. A Simple Greedy Algorithm for Dynamic Graph Orientation. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ISAAC.2017.12`.

6 Markus Blumenstock. Fast algorithms for pseudoarboricity. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*, pages 113–126. SIAM, 2016. `doi:10.1137/1.9781611974317.10`.

7 Markus Blumenstock and Frank Fischer. A constructive arboricity approximation scheme. In *SOFSEM 2020: Theory and Practice of Computer Science – 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, volume 12011 of *Lecture Notes in Computer Science*, pages 51–63. Springer, 2020. `doi:10.1007/978-3-030-38919-2_5`.

8 Enno Brehm. 3-orientations and schnyder 3-tree-decompositions, 2000.

9 Gerth Stolting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *In Proc. 6th International Workshop on Algorithms and Data Structures (WADS)*, pages 342–351. Springer-Verlag, 1999.

10 Aleksander B. G. Christiansen. Dynamic algorithms for implicit vertex-colouring of graphs with bounded arboricity. Master's thesis, Technical University of Denmark, Kgs. Lyngby, Denmark, October 2021.

11 Aleksander B. G. Christiansen and Eva Rotenberg. Fully-dynamic $\alpha + 2$ arboricity decompositions and implicit colouring. In *ICALP*, volume 229 of *LIPIcs*, pages 42:1–42:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

12 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci.*, 86(2):243–266, 1991.

13 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October – 1 November 1989*, pages 436–441, 1989. doi:10.1109/SFCS.1989.63515.

14 Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, page 67, 1965.

15 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, August 1994. doi:10.1016/0020-0190(94)90121-X.

16 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 599–608, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644208.

17 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *Journal of Computer and Systems Sciences*, 52(1):3–27, February 1996. doi:10.1006/jcss.1996.0002.

18 Harold Gabow and Herbert Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 407–421, New York, NY, USA, 1988. Association for Computing Machinery.

19 Harold N. Gabow. Algorithms for graphic polymatroids and parametric s-sets. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 88–97, USA, 1995. Society for Industrial and Applied Mathematics.

20 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *Journal of the ACM*, 46(1):28–91, 1999. doi:10.1145/300515.300517.

21 David G. Harris, Hsin-Hao Su, and Hoa T. Vu. On the locality of nash-williams forest decomposition and star-forest decomposition. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 295–305. ACM, 2021. doi:10.1145/3465084.3467908.

22 Meng He, Ganggui Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *ISAAC*, 2014.

23 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020. arXiv:2002.10142.

24 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 167–180. ACM, 2020. doi:10.1145/3357713.3384249.

25 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental spqr-trees: Embeddings, planarity, and triconnectivity. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2378–2397. SIAM, 2020. Full version: arXiv:1910.09005. doi:10.1137/1.9781611975994.146.

26 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. doi:10.1007/3-540-57273-2_57.

27 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.

28 Łukasz Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In *Proceedings of the 17th International Conference on Algorithms and Computation*, ISAAC'06, pages 557–566, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11940128_56.

29 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, May 2007. doi:10.1016/j.ipl.2006.12.006.

30 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in Õ(vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433, 2014. doi:10.1109/FOCS.2014.52.

31 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262, 2013. doi:10.1109/FOCS.2013.35.

32 C. St.J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 1961. doi:10.1112/jlms/s1-36.1.445.

33 C. St.J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, 1964. doi:10.1112/jlms/s1-39.1.12.

34 Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982. doi:10.1002/net.3230120206.

35 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. doi:10.1145/195058.195439.

36 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 114–122, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800076.802464.

37 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3392724.

38 Jeffery 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. doi:10.1007/3-540-55719-9_86.

$\mathsf{LO_v}$-Calculus: A Graphical Language for Linear Optical Quantum Circuits

Alexandre Clément ✉ 🏠
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Nicolas Heurtel ✉
Quandela, 7 Rue Léonard de Vinci, 91300 Massy, France
Université Paris-Saclay, CentraleSupélec, Inria, CNRS, ENS Paris-Saclay,
Laboratoire Méthodes Formelles, 91190 Gif-sur-Yvette, France

Shane Mansfield ✉
Quandela, 7 Rue Léonard de Vinci, 91300 Massy, France

Simon Perdrix ✉ 🏠
Inria Mocqua, LORIA, CNRS, Université de Lorraine, F-54000 Nancy, France

Benoît Valiron ✉ 🏠
Université Paris-Saclay, CentraleSupélec, Inria, CNRS, ENS Paris-Saclay,
Laboratoire Méthodes Formelles, 91190 Gif-sur-Yvette, France

Abstract

We introduce the $\mathrm{LO_v}$-calculus, a graphical language for reasoning about linear optical quantum circuits with so-called vacuum state auxiliary inputs. We present the axiomatics of the language and prove its soundness and completeness: two $\mathrm{LO_v}$-circuits represent the same quantum process if and only if one can be transformed into the other with the rules of the $\mathrm{LO_v}$-calculus. We give a confluent and terminating rewrite system to rewrite any polarisation-preserving $\mathrm{LO_v}$-circuit into a unique triangular normal form, inspired by the universal decomposition of Reck *et al.* (1994) for linear optical quantum circuits.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Axiomatic semantics; Hardware → Quantum computation; Hardware → Quantum communication and cryptography

Keywords and phrases Quantum Computing, Graphical Language, Linear Optical Circuits, Linear Optical Quantum Computing, Completeness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.35

Related Version *Full Version*: `https://arxiv.org/abs/2204.11787`

Funding This work is funded by ANR-17-CE25-0009 SoftQPro, ANR-17-CE24-0035 VanQuTe, PIA-GDN/Quantex, and LUE / UOQ, the PEPR integrated project EPiQ ANR-22-PETQ-0007 part of Plan France 2030, and by *"Investissements d'avenir"* (ANR-15-IDEX-02) program of the French National Research Agency; the European Project NExt ApplicationS of Quantum Computing (NEASQC), funded by Horizon 2020 Program inside the call H2020-FETFLAG-2020-01(Grant Agreement 951821), and the HPCQS European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101018180.

1 Introduction

Quantum computing and information processing promise a variety of advantages over their classical analogues, from the potential for computational speedups (e.g. [33, 51]) to enhanced security and communication (e.g. [7, 28]). By encoding information into the states of physical systems that are quantum rather than classical, one can then process that information by

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 35; pp. 35:1–35:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

evolving and manipulating the systems according to the laws of quantum mechanics. This opens up the possibility of exploiting non-classical behaviours available to quantum systems in order to process information in radically new and potentially advantageous ways.

The development of quantum technologies has proceeded at pace over the past number of years, with a variety of different physical supports for quantum information being pursued. These include matter-based systems like superconducting circuits, cold atoms, and trapped ions, as well as light-based systems, in which information is encoded in photons. Among these, photons have a privileged role in the sense that regardless of hardware choice it will eventually be necessary to network quantum processors, and (as the only sensible support for communicating quantum information) some quantum information will need to be treated photonically. Yet, in their own right, photons also offer viable approaches to quantum computation in the noisy intermediate-scale [40] and large-scale fault-tolerant [6] regimes.

The standard unit of quantum information is the quantum bit or qubit, and photons allow for a rich variety of ways to encode qubits. However it is also interesting to note that treating photons as informational units in their own right can be advantageous. A good example is BosonSampling, originally proposed by Aaronson and Arkhipov [1], a computational task that is $\#P$-hard but which can be efficiently solved by interacting photons in an idealised generic linear-optical circuit in which no qubit encoding need be imposed. At present, along with Random Circuit Sampling [2, 9], this provides one of the two main routes to experimental demonstrations of quantum computational advantage [3, 55, 53, 54], in which quantum devices have been claimed to outperform classical capabilities for specific tasks.

The usual semantics for quantum computation stemming from quantum mechanics is based on unitary matrices (or unitary operators in general) over Hilbert spaces. Although this faithfully models the extensional behaviour of a computation, it fails to address several key aspects that are of interest when considering the design and implementation of quantum algorithms. A first limitation is the intensional description of the computation: an algorithm or quantum computation in general consists of modular components that are composed and combined in specific way, and one wants to keep track of this information. One therefore needs a *language* for coding these. The other important aspect is the need to specify and verify the said code. Indeed, classically simulating a quantum process is a task that is exponentially costly in the size of the system, while running code on physical devices is expensive. If some limited testing techniques are available on quantum systems [29, 43], it is however highly desirable to be able to reason and prove the desired properties of the code upstream, and rely on *formal methods*. If text-based high-level languages oriented towards formal methods have successfully been proposed in the literature [32, 8, 37], we aim in this paper to explore a lower-level, graphical language, making contact with photonic hardware.

Graphical languages for quantum computation have a long history: since Feyman diagrams [30], graphical languages for representing (low-level) quantum processes have been considered as an answer to the limitations of plain unitary matrices. Quantum circuits – the quantum equivalent to classical, boolean circuits – are an obvious candidate for a graphical language, and indeed, several lines of research took them as their main object of study [32, 22, 46, 13]. Quantum circuits in particular form a natural medium for describing the execution flow of a computation. The main problem with the model of quantum circuits is the lack of a satisfactory equational presentation. If several attempts have been made for various subsets [20, 19, 36, 45], none of them provides a complete presentation.

A recent proposal responding to the shortfalls of quantum circuits as a model is the ZX-calculus [21], which, along with its variants [11, 4, 12], have proved to be particularly useful for reasoning about qubit quantum mechanics, for applications such as quantum circuit optimisation [25, 5], verification [26, 31, 35] and representation e.g. for MBQC patterns [27] or error-correction [27, 23]. However, while ZX-calculus is versatile and provides a welcomed formal semantics for quantum computation, it remains at an abstract level.

There is therefore a clear interest in developing a graphical language for quantum photonic processes, especially linear quantum optics, which is closer to photonic hardware and laboratory operations that are easily implementable in bulk optics, fibres, or in integrated photonic circuits. This would provide a more formal counterpart to software frameworks that have been proposed for defining and classically simulating such processes to the extent that it is tractable [39, 34]. The need for such a formal language is also evidenced, for example, by the appeal to diagrams to concisely illustrate equivalent unitaries in recent work in the physics literature [48]. Following on the trend for graphical quantum languages, the PBS-calculus [16, 10, 17] has been proposed as a first step towards an alternative to ZX dedicated to linear quantum optical computation (LOQC). The PBS-calculus makes it possible to reason on a small subset of linear optical components only acting on the polarisation of a photon. While it is enough to describe and analyse non causally-ordered computations, it falls short at expressing other aspects of LOQC typically considered in the physics community, such as the phase. Note that a recent, independent work[1] establishes some connections between the ZX-calculus and the photon preserving fragment of linear optics with multiple photons [24].

Our goal here is to take a more bottom-up approach and to propose a new language which formalises the kinds of diagrammatics that are currently in use in the physics community. In practice this can find many uses including for the design, optimisation, verification, error-correction, and systematic study of linear optical quantum circuits for quantum information.

Contributions. Our main contributions are the following.

- A graphical language for LOQC featuring most of the physical apparatuses used in the physics literature. The language comes equipped with an equational theory that is sound and complete with respect to the standard semantics of LOQC.
- A strongly normalising and globally confluent rewrite system and normal form for the polarisation-preserving fragment, for which we recover the Reck *et al.* [49] decomposition as normal form (modulo 0-angled beam splitters and 0-angled phase shifters) with a novel proof of its uniqueness.

Finally, and maybe more importantly, our language makes it possible to formalise and reason within a common framework on various presentations of LOQC stemming from parallel research paths. Our semantics not only allow us to recover, extend and improve on some key results in LOQC such as the universal decompositions of Reck *et al.* [49] and Clements *et al.* [18], but it also gives a unifying language for the different formalisms from the literature. Furthermore, this result paves the way towards the design of complete equational theories for quantum circuits [14].

Plan of the paper. The article is structured as follows. In Section 2, we present the syntax and the semantics of the LO_v-calculus. The equational theory and its soundness are given in Section 3. In Section 4 we present the strongly normalising and globally confluent rewrite system. This allows us to prove the completeness of the LO_v-calculus in Section 5. Finally, we conclude in Section 6. More complete proofs can be found in the appendix of the technical report [15].

[1] The preprint version of [24] has been upladed to arXiv a few days after the one of the present paper [15].

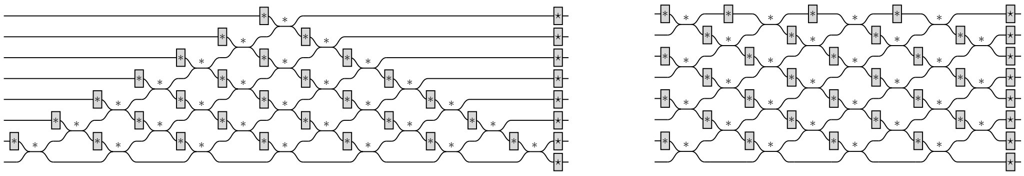
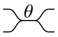
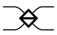
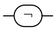
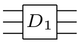
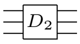
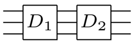
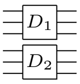

(a) Triangular form [49]. **(b)** Rectangular form [18].

Figure 1 Triangular and rectangular forms for polarisation-preserving circuits.

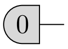
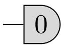
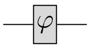
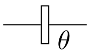
2 Linear Optical Quantum Circuits

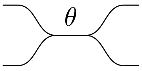
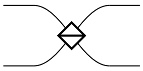
A linear optical quantum computation [42, 41] (LOQC) consists of spatial modes through which photons pass – which may be physically instantiated by optical fibers, waveguides in integrated circuits, or simply by paths in free space (bulk optics) – and operations that act on the spatial and polarisation degrees of freedom of the photons, including in particular *beam splitters* (), *polarising beam splitters* (), *phase shifters* ( ), *wave plates* ( ), *pola-negations* () and finally the *vacuum state sources* and *detectors* (and). Their action and the semantics are described in Section 2.2.

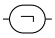
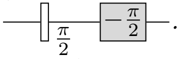
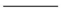
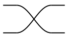
2.1 Syntax

In order to formalise linear optical quantum circuits, we use the formalism of PROPs [44]. A PRO is a strict monoidal category whose monoid of objects is freely generated by a single X: the objects are all of the form $X \oplus ... \oplus X$,[2] and simply denoted by n, the number of occurrences of X. PROs are typically represented graphically as circuits: each copy of X is represented by a wire and morphisms by boxes on wires, so that $\oplus$ is represented vertically and morphism composition "$\circ$" is represented horizontally. For instance, D_1 and D_2 represented as  and  can be horizontally composed as $D_2 \circ D_1$, represented by  , and vertically composed as $D_1 \oplus D_2$, represented by  . A PROP is the symmetric monoidal analogue of PRO, so it is equipped with a swap .

▶ **Definition 1.** **LO$_v$** *is the PROP of* LO$_v$*-circuits generated by*

 $: 0 \to 1$  $: 1 \to 0$  $: 1 \to 1$  $: 1 \to 1$

 $: 2 \to 2$  $: 2 \to 2$

where $\theta, \varphi \in \mathbb{R}$. When the parameters θ and φ are omitted we take them to be equal to $\pi/4$. We write  *as a shortcut notation for*  *. The tensor of the monoidal structure is denoted with $\oplus$, and the identity, swap and empty circuit (unit of $\oplus$) are denoted as follows:*  $: 1 \to 1$,  $: 2 \to 2$, $: 0 \to 0$.

▶ **Example 2.** An example of a linear optical quantum circuit using all of the connectives presented in Definition 1 is shown in Figure 2.

[2] Here we denote the monoidal product as $\oplus$ rather than $\otimes$ in order to better correspond to the semantics of LO$_v$-circuits (see Section 2.2).

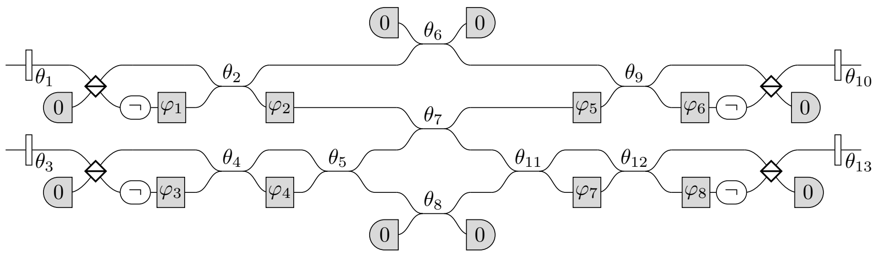


Figure 2 LO$_\text{v}$-circuit implementing a variational quantum eigensolver [47], an algorithm with applications including calculation of ground-state energies in quantum chemistry.

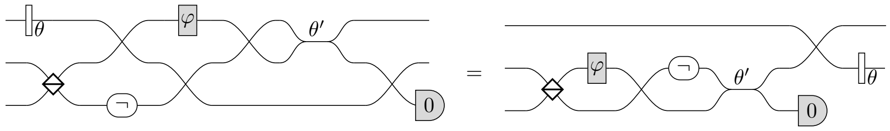
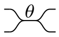


Figure 3 Two equivalent representations of the same LO$_\text{v}$-circuit.

▶ Remark 3. The axioms of PROPs guarantee that linear optical quantum circuits are defined up to deformations: Figure 3 shows two equivalent circuits under the equations of PROPs.

Among the generators, the beam splitters and phase shifters are known to preserve the polarisation of the photons, as a consequence, we define a *polarisation-preserving* sub-PRO of LO$_\text{v}$ as follows.

▶ **Definition 4. LO$_\text{PP}$** *is the PRO of* polarisation-preserving circuits *generated by beam splitters*  *and phase shifters* *.*

Notice that we define polarisation-preserving circuits as a PRO rather than a PROP, thus they do not include swaps.

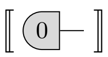
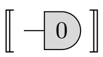
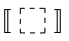
2.2 Single-Photon Semantics

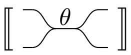
We will characterise photons by their spatial and polarisation modes. Spatial modes refer to position, and polarisation can be horizontal (H) or vertical (V). Note that the quantum formalism admits (normalised complex) superpositions of both spatial and polarisation modes. For any $n \in \mathbb{N}$, let $M_n = \{\mathrm{V}, \mathrm{H}\} \times [n]$, where $[n] = \{0, \dots n-1\}$, be the set of states (spatial and polarisation modes). The elements of M_n are denoted c_p with $c \in \{\mathrm{V}, \mathrm{H}\}$ and $p \in [n]$. The state space of a single photon is $\mathbb{C}^{M_n} = span(|\mathrm{V}_i\rangle, |\mathrm{H}_i\rangle \mid i \in [n])$. Notice that $\mathbb{C}^{M_0} = \mathbb{C}^{\emptyset} = \{0\}$ is the Hilbert space of dimension 0. For instance, on 2 spatial modes (i.e. 2 wires), there are four possible basis states: $\mathrm{H}_0, \mathrm{H}_1, \mathrm{V}_0, \mathrm{V}_1$. Indeed, a photon can be on one of the two wires, while in the horizontal or vertical polarisation. The state space is then a 4-dimensional Hilbert space. The semantics of a LO$_\text{v}$-circuit is defined as follows.

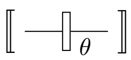
▶ **Definition 5.** *For any* LO$_\text{v}$*-circuit* $D : n \to m$*, let* $[\![D]\!] : \mathbb{C}^{M_n} \to \mathbb{C}^{M_m}$ *be the linear map inductively defined by Table 1*[3]*, and by* $[\![D_2 \circ D_1]\!] = [\![D_2]\!] \circ [\![D_1]\!]$, $[\![D_1 \oplus D_2]\!] = [\![D_1]\!] \oplus [\![D_2]\!]$*, where for all* $f \in \mathbb{C}^{M_n} \to \mathbb{C}^{M_m}$ *and* $g \in \mathbb{C}^{M_{n'}} \to \mathbb{C}^{M_{m'}}$, $(f \oplus g)(|c_k\rangle) = f(|c_k\rangle)$ *if* $k < n$ *and* $S_{m,m'}(g(|c_{k-n}\rangle))$ *if* $k \geq n$*, with* $S_{m,m'} : \mathbb{C}^{M_{m'}} \to \mathbb{C}^{M_{m+m'}} = |c_k\rangle \mapsto |c_{k+m}\rangle$ *a shift of the positions by* m*.*

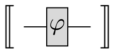
[3] There are many possible conventions for beam splitters. We have chosen this one as it is a symmetric

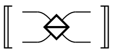
Table 1 Semantics of LO$_\mathrm{v}$-circuits.

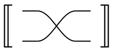
 $= 0$  $= 0$  $= 0$

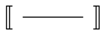
 $= |c_p\rangle \mapsto \cos(\theta)|c_p\rangle + i\sin(\theta)|c_{1-p}\rangle$

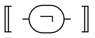
 $= \begin{cases}|\mathrm{V}_0\rangle \mapsto \cos(\theta)|\mathrm{V}_0\rangle + i\sin(\theta)|\mathrm{H}_0\rangle\\ |\mathrm{H}_0\rangle \mapsto \cos(\theta)|\mathrm{H}_0\rangle + i\sin(\theta)|\mathrm{V}_0\rangle\end{cases}$

 $= |c_0\rangle \mapsto e^{i\varphi}|c_0\rangle$

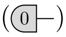
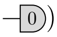
 $= \begin{cases}|\mathrm{V}_p\rangle \mapsto |\mathrm{V}_p\rangle\\ |\mathrm{H}_p\rangle \mapsto |\mathrm{H}_{1-p}\rangle\end{cases}$

 $= |c_p\rangle \mapsto |c_{1-p}\rangle$

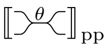
 $= |c_0\rangle \mapsto |c_0\rangle$

▶ **Example 6.** The negation inverts polarisation:  : $|\mathrm{V}_0\rangle \mapsto |\mathrm{H}_0\rangle$ and $|\mathrm{H}_0\rangle \mapsto |\mathrm{V}_0\rangle$.

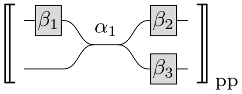
▶ Remark 7. The semantics of the circuits is sound with respect to the axioms of PROPs. In other words two circuits that are equal up to deformation have the same semantics. More formally, $[\![.]\!] : \mathbf{LO_v} \to (\mathbf{Hilb}, \oplus, 0)$ is a monoidal functor where **Hilb** is the category of state spaces $\mathbb{C}^{M_n}$ and linear maps.

▶ Remark 8. All the generators of the LO$_\mathrm{v}$-circuits are photon preserving, even the vacuum state sources () and detectors (). Indeed the vacuum state source produces no photons, whereas the semantics of the detector corresponds to a postselection on the case where no photons are detected.

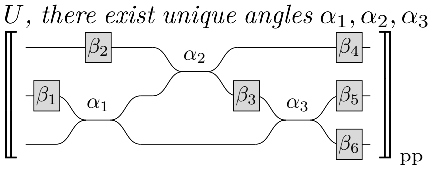
▶ **Definition 9.** *For any* LO$_\mathrm{PP}$*-circuit* $D : n \to n$*, we define* $[\![D]\!]_\mathrm{pp} : \mathbb{C}^n \to \mathbb{C}^n$ *as the unique linear map such that* $[\![.]\!] \circ \iota = \iota \circ [\![.]\!]_\mathrm{pp}$ *where* $\iota : \mathbb{C}^n \to \mathbb{C}^{M_n} = |k\rangle \mapsto |\mathrm{H}_k\rangle$.

For instance  $= \begin{pmatrix}\cos(\theta) & i\sin(\theta)\\ i\sin(\theta) & \cos(\theta)\end{pmatrix}$.

Polarisation-preserving circuits are universal for unitary transformations, this is a direct consequence of the result of Reck *et al.* [49]. Unitary transformations can actually be uniquely represented by LO$_\mathrm{PP}$-circuits, as illustrated by the following two cases on 2 and 3 modes, the general case being proved in Section 4.

▶ **Lemma 10.** *For any unitary* 2×2 *matrix* U*, there exist unique* $\beta_1, \alpha_1 \in [0, \pi)$ *and* $\beta_2, \beta_3 \in [0, 2\pi)$ *such that*  $= U$*, and* $\alpha_1 \in \{0, \frac{\pi}{2}\} \Rightarrow \beta_1 = 0$.

Proof. The proof is given in [15]. ◀

▶ **Lemma 11.** *For any unitary* 3×3 *matrix* U*, there exist unique angles* $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3 \in [0, \pi)$ *and* $\beta_4, \beta_5, \beta_6 \in [0, 2\pi)$ *such that*  $= U$ *where* $\forall i \in \{1, 2, 3\}, \alpha_i \in \{0, \frac{\pi}{2}\} \Rightarrow \beta_i = 0$*, and where* $\alpha_2 = 0 \Rightarrow \alpha_1 = 0$.

Proof. The existence of such a canonical form is shown in [49]. The uniqueness can then be derived by analysing the possible cases (See [15]). ◀

operation with good composition properties. The convention for the wave plate has been chosen for similar reasons.

$\mathrm{LO_v}$-circuits are more expressive than $\mathrm{LO_{PP}}$-ones, they not only act on the polarisation but the use of detectors and sources allow the representation of non-unitary evolutions: For any $\mathrm{LO_v}$-circuit $D : n \to m$, $[\![D]\!]$ is sub-unitary[4]. $\mathrm{LO_v}$-circuits are actually universal for sub-unitary transformations:

▶ **Theorem 12** (Universality of $\mathrm{LO_v}$). *For every sub-unitary map $U : \mathbb{C}^{M_n} \to \mathbb{C}^{M_m}$ (i.e. such that $U^\dagger U \sqsubseteq I$) there exists a diagram $D : n \to m$ s.t. $[\![D]\!] = U$.*

Proof. The proof given in [15] relies on the normal forms developed in Section 5. ◀

3 Equational Theory

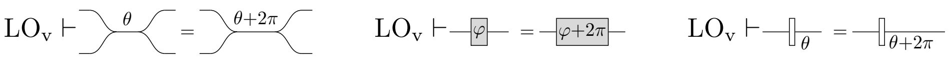
Two distinct $\mathrm{LO_v}$-circuits may represent the same quantum evolution: for instance, composing two negations is equivalent to the identity. In order to characterise equivalences of $\mathrm{LO_v}$-circuits, we introduce a set of equations, shown in Figure 4. They capture basic properties of $\mathrm{LO_v}$-circuits, such as: detectors and sources essentially absorbing the other generators (Equations (9) to (12)); parameters forming a monoid (Equations (1) and (2)); and various commutation properties (Equations (15), (16)). Notice that there are two equations acting on 3 modes: Equation (6) and Equation (18). Equation (6) is a variant of the Yang-Baxter Equation [38], whereas Equation (18) is a property of decompositions into Euler angles. Indeed, in 3-dimensional space, the two sides of this equation correspond to two distinct decompositions in elementary rotations.

▶ **Definition 13** ($\mathrm{LO_v}$-calculus). *Two $\mathrm{LO_v}$-circuits D_1, D_2 are equivalent according to the rules of the* $\mathrm{LO_v}$-calculus, *denoted $\mathrm{LO_v} \vdash D_1 = D_2$, if one can transform D_1 into D_2 using the equations given in Figure 4. More precisely, $\mathrm{LO_v} \vdash \cdot = \cdot$ is defined as the smallest congruence which satisfies the equations of Figure 4 in addition to the axioms of PROP.*

▶ **Proposition 14** (Soundness). *For any two $\mathrm{LO_v}$-circuits D_1 and D_2, if $\mathrm{LO_v} \vdash D_1 = D_2$ then $[\![D_1]\!] = [\![D_2]\!]$.*

Proof. Since semantic equality is a congruence it suffices to check that for every equation of Figure 4 both sides have the same semantics, which follows from Definition 5 and Lemma 11. ◀

▶ **Proposition 15.** *The rules of the $\mathrm{LO_v}$-calculus imply that the parameters are 2π-periodic, i.e. for any $\theta, \varphi \in \mathbb{R}$:*



Proof. The proof is given in [15]. ◀

We now state one of our main results: the completeness of the $\mathrm{LO_v}$-calculus.

▶ **Theorem 16** (Completeness). *For any two $\mathrm{LO_v}$-circuits D_1 and D_2, if $[\![D_1]\!] = [\![D_2]\!]$ then $\mathrm{LO_v} \vdash D_1 = D_2$.*

The proof of Theorem 16 is given in Section 5. As a step towards proving the theorem, we first consider the fragment of the $\mathrm{LO_{PP}}$-circuits.

[4] U is sub-unitary (see for instance [50]) iff $U^\dagger U \sqsubseteq I$, where $\sqsubseteq$ is the Löwner partial order, i.e. $I - U^\dagger U$ is a positive semi-definite.

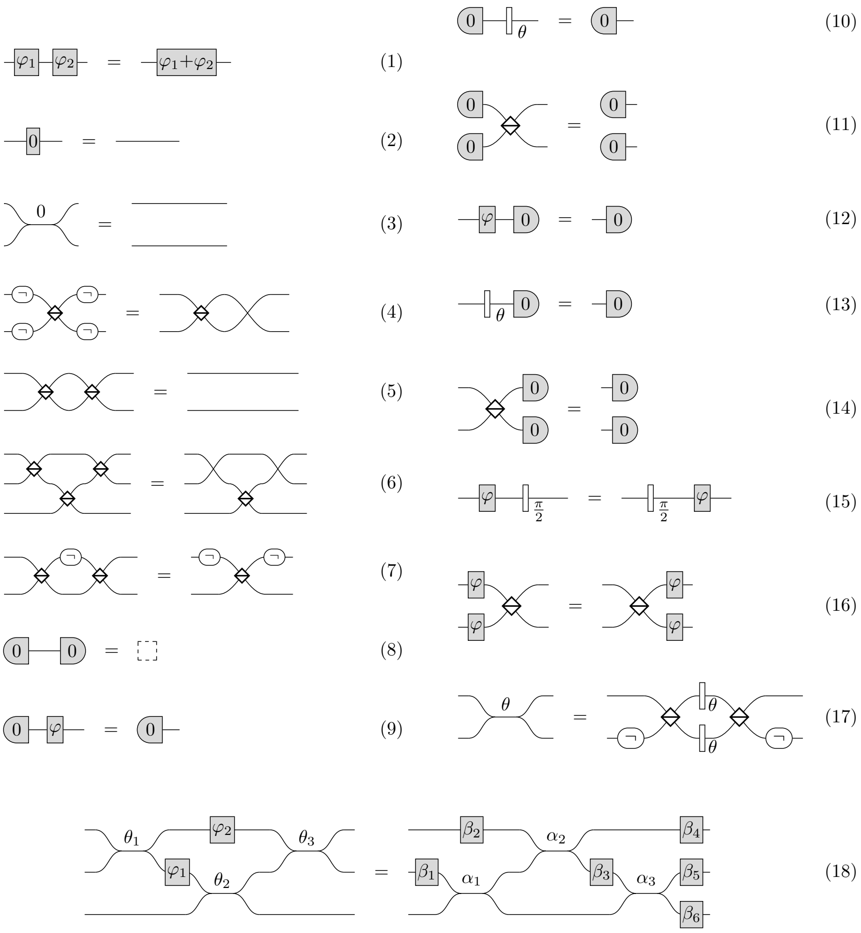

φ_1 φ_2 = $\varphi_1+\varphi_2$ (1)
0 = (2)
0 = (3)
¬ = (4)
= (5)
= (6)
¬ = ¬ (7)
0 0 = (8)
0 φ = 0 (9)
0 θ = 0 (10)
0 0 = 0 0 (11)
φ 0 = 0 (12)
θ 0 = 0 (13)
0 0 = 0 0 (14)
φ $\frac{\pi}{2}$ = $\frac{\pi}{2}$ φ (15)
φ φ = φ φ (16)
θ = ¬ θ θ ¬ (17)
θ_1 φ_2 θ_3 φ_1 θ_2 = β_2 α_2 β_4 β_1 α_1 β_3 α_3 β_5 β_6 (18)

Figure 4 Axioms of the LO$_v$-calculus. The equations are valid for arbitrary parameters $\varphi, \varphi_i, \theta, \theta_i \in \mathbb{R}$. In Equation (18), the angles on the left-hand side can take any value while the right-hand side is given by Lemma 11 (where U is the $[\![.]\!]_{\text{pp}}$-semantics of the left-hand side of the equation).

4 Polarisation-Preserving Circuits

This section gives a universal normal form for any LO$_{\text{PP}}$-circuit. We prove the uniqueness of that form by introducing a strongly normalising and confluent polarisation-preserving rewrite system: PPRS.

▶ **Definition 17.** *The rewrite system* PPRS *is defined on* LO$_{\text{PP}}$*-circuits with the rules of Figure 5.*

▶ **Lemma 18.** *If D_1 rewrites to D_2 using the* PPRS *rewrite system then* LO$_v \vdash D_1 = D_2$.

Proof. The proof is given in [15]. ◀

▶ **Theorem 19.** *The rewrite system* PPRS *is strongly normalising.*

Proof. The proof is done by defining a lexicographic order on six distinct values: numbers of beam splitters of various angle ranges, count of specific patterns, numbers and positions of phase shifters. The order is shown to be decreasing with respect to the rewrite rules of PPRS. The complete proof is given in [15]. ◀

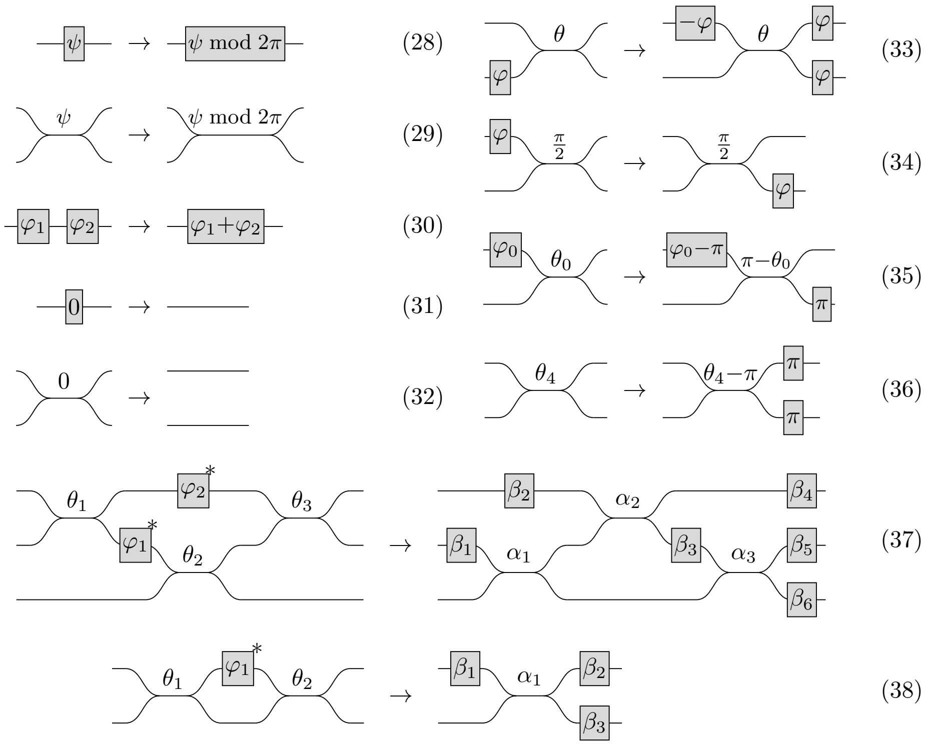


Figure 5 Rewriting rules of PPRS. $\psi \in \mathbb{R} \setminus [0, 2\pi)$, $\varphi, \varphi_1, \varphi_2 \in (0, 2\pi)$, $\varphi_0, \theta_4 \in [\pi, 2\pi)$, $\theta, \theta_0, \theta_1, \theta_2, \theta_3 \in (0, \pi)$, and $\theta_0 \neq \frac{\pi}{2}$. $-\boxed{\varphi}^{*}-$ denotes either $-\boxed{\varphi}-$ or ———. In Rules (37) and (38), the angles on the left-hand side can take any value while the right-hand side is given by Lemma 11 and Lemma 10 respectively.

As PPRS is terminating, we can therefore derive the existence of normal forms. The next step is to show that these normal forms are unique: this is derived from Theorem 20.

▶ **Theorem 20.** PPRS *is globally confluent.*

Proof. PPRS is locally confluent. Indeed, one can show by case analysis that the non-trivial peaks all use at most three wires. Each peak can be closed since for any polarisation-preserving LO$_\text{v}$-circuit of size $n \in \{1, 2, 3\}$, PPRS terminates to a specific unique normal form: when $n = 1$, a simple phase-shift; when $n = 2$, the form shown in Lemma 10; when $n = 3$, the form shown in Lemma 11. See [15] for details. Finally, using Theorem 19, global confluence is deduced from Newman's lemma [52]. ◀

▶ **Definition 21.** *A* PPRS triangular normal form *is a circuit with a triangular shape similar to Figure 1a, but with all* 0*-angled generators replaced with identities and with additional conditions on the angles, as described in Figure 6.*

Figure 7 shows an example: the figure on the left is the “full” circuit with 0-angled beam splitters while on the right is the corresponding PPRS triangular normal form.

▶ **Lemma 22.** *Any irreducible* LO$_\text{PP}$*-circuit is a PPRS triangular normal form.*

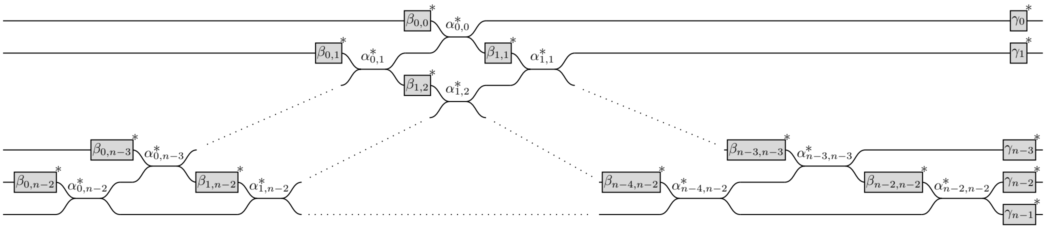


Figure 6 General scheme of a PPRS triangular normal form. The stars mean that any phase shifter or beam splitter with angle 0 is replaced by the identity. The conditions on the angles are the following: $\alpha_{i,j}, \beta_{i,j} \in [0,\pi)$; $\gamma_i \in [0,2\pi)$; $\alpha_{i,j} = 0 \Rightarrow \forall j' > j, \alpha_{i,j'} = 0$; $\alpha_{i,j} \in \{0, \frac{\pi}{2}\} \Rightarrow \beta_{i,j} = 0$.

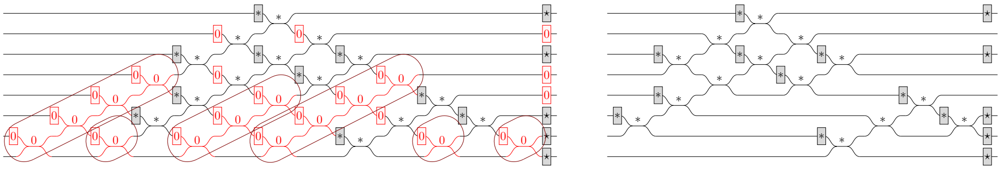


Figure 7 An example of a PPRS triangular normal form. In the figure on the left, the beam splitters and phase shifters with angle 0 in the corresponding triangular form are shown in red. In the figure on the right, they are replaced with identities.

Proof. This property can be proven by induction. First, we lay out the properties of any irreducible circuit that can be directly deduced from the PPRS rules of Figure 5. Then, we give two more properties characterising the PPRS triangular normal forms. By induction, we prove that any irreducible circuit respects those two properties, so that any irreducible circuit is a PPRS triangular normal form. See [15] for more details. ◀

▶ **Theorem 23.** *Any* LO$_{\text{PP}}$*-circuit, with the rules of* PPRS*, converges to a unique* PPRS *triangular normal form.*

Proof. PPRS is globally confluent and terminating: normal forms are unique. From Lemma 22, PPRS triangular normal forms are the only irreducible forms. Therefore, any polarisation-preserving circuit terminates to such a unique normal form. ◀

▶ Remark 24. In particular by using Equation (18) and by adding 0-angled beam splitters if necessary, one can turn any circuit in PPRS triangular normal form into a circuit in the rectangular form of [18] shown in Figure 1b. A schematic example of such a transformation is shown in [15].

We can now prove the completeness of the polarisation-preserving fragment.

▶ **Theorem 25.** *For any* LO$_{\text{PP}}$*-circuits* C_1,C_2 *such that* $[\![C_1]\!]_{\text{pp}} = [\![C_2]\!]_{\text{pp}}$*, their normal forms are equal, i.e.* $N_1 = N_2$*, where* N_1 *(resp.* N_2*) is the unique normal form of* C_1 *(resp.* C_2*) given by Theorem 23.*

Proof. As the rewrite system preserves the semantics, it is sufficient to prove that $[\![N_1]\!]_{\text{pp}} = [\![N_2]\!]_{\text{pp}} \Rightarrow N_1 = N_2$. First, we can show by induction that $[\![N]\!]_{\text{pp}} = [\![I_n]\!]_{\text{pp}} \Rightarrow N = I_n$. Indeed, to have the semantics as the identity, we can show the upper beam splitter and phase shifters are necessarily 0-angled. The proof follows from induction, details are given in [15]. Let P be an inverse circuit of N_1 and N_2, that is, a polarisation-preserving circuit such that $[\![P]\!]_{\text{pp}} = [\![N_1]\!]_{\text{pp}}^{-1}$. The existence of such a circuit follows from [49]. As $[\![N_1P]\!]_{\text{pp}} = [\![PN_2]\!]_{\text{pp}} = [\![I_n]\!]_{\text{pp}}$, the term N_1PN_2 can both be reduced to N_1 (by reducing PN_2 first) and N_2 (by reducing N_1P first). By Theorem 23, $N_1 = N_2$. ◀

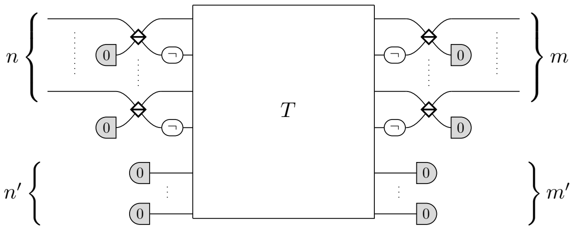


Figure 8 Shape of a circuit in normal form as of Definition 27.

▶ **Proposition 26** (Universality and uniqueness in the polarisation-preserving fragment)**.** *For any unitary* $U\colon \mathbb{C}^n \to \mathbb{C}^n$*, there exists a unique circuit* T *in* PPRS *triangular normal form such that* $[\![T]\!]_{\mathrm{pp}} = U$*.*

Proof. This follows directly from [49], Theorems 23 and 25 and the fact that all PPRS triangular normal forms are irreducible. ◀

5 Completeness of the $\mathrm{LO_v}$-Calculus

To prove the completeness of the $\mathrm{LO_v}$-Calculus (Theorem 16), we introduce the following notion of normal form.

▶ **Definition 27** (Normal form)**.** *A circuit in normal form* $N : n \to m$ *is a circuit of the form shown in Figure 8, where* T *is a PPRS triangular normal form (Definition 21). If* $n' = m' = 0$*, then* N *is said to be in* pure normal form.

▶ **Lemma 28** (Uniqueness of the pure normal form)**.** *If two circuits* N_1 *and* N_2 *in pure normal form are such that* $[\![N_1]\!] = [\![N_2]\!]$*, then* $N_1 = N_2$*.*

Proof. Let T_1 (resp. T_2) be the $\mathrm{LO_{PP}}$-circuit associated with N_1 (resp N_2) as in Figure 8. Notice that $[\![T_i]\!]_{\mathrm{pp}} \circ \mu = \mu \circ [\![N_i]\!]$ where $\mu : \mathbb{C}^{M_n} \to \mathbb{C}^{2n}$ is the isomorphism $|\mathrm{V}_k\rangle \mapsto |2k\rangle$ and $|\mathrm{H}_k\rangle \mapsto |2k+1\rangle$. Thus $[\![N_1]\!] = [\![N_2]\!]$ implies $[\![T_1]\!]_{\mathrm{pp}} = [\![T_2]\!]_{\mathrm{pp}}$ so that the result follows from Theorem 23. ◀

▶ **Lemma 29.** *For any circuit* D *without vacuum state sources or detectors there exists a circuit in pure normal form* N *such that* $\mathrm{LO_v} \vdash D = N$*.*

Proof. The proof is given in [15]. ◀

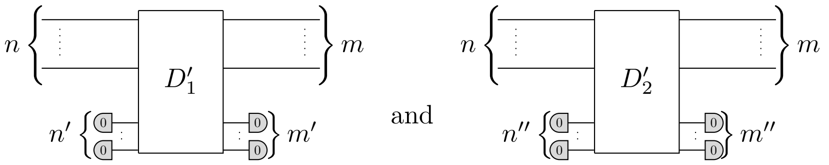
Completeness for circuits without vacuum state sources or detectors follows directly from Lemmas 28 and 29:

▶ **Proposition 30.** *Given any two circuits* D_1 *and* D_2 *without any* (0)– *or* –(0)*, if* $[\![D_1]\!] = [\![D_2]\!]$ *then* $\mathrm{LO_v} \vdash D_1 = D_2$*.*

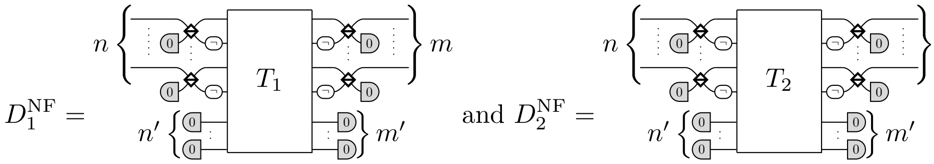
Proof. By Lemma 29, there exist two circuits in pure normal form N_1 and N_2 such that $\mathrm{LO_v} \vdash D_1 = N_1$ and $\mathrm{LO_v} \vdash D_2 = N_2$. By Proposition 14, one has $[\![N_1]\!] = [\![D_1]\!] = [\![D_2]\!] = [\![N_2]\!]$, so that by Lemma 28, $N_1 = N_2$. The result follows by transitivity. ◀

Proof of Theorem 16

We now have the required material to to finish the proof of Theorem 16. Let $D_1, D_2 : n \to m$ be any two LO$_\text{v}$-circuits such that $[\![D_1]\!] = [\![D_2]\!]$. By deformation, we can write them as



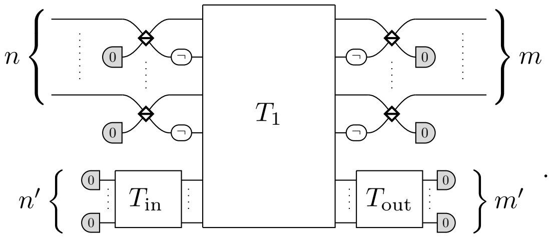
where D_1', D_2' do not contain (0)— or —(0). Up to using Equation (8), we can assume that $n'' = n'$. Since circuits without vacuum state sources and detectors necessarily have the same number of input wires as of output wires, this implies that $m'' = m'$. By Lemma 29, we can put D_1' and D_2' in pure normal form. Then by using Equations (9)–(14), we get two circuits in normal form



with T_1 and T_2 in PPRS triangular normal form.

$[\![D_1]\!] = [\![D_2]\!]$ implies that $\pi \circ [\![T_1]\!]_{\text{pp}} \circ \iota = \pi \circ [\![T_2]\!]_{\text{pp}} \circ \iota$ where $\iota : \mathbb{C}^{2n} \to \mathbb{C}^{2n+n'}$ is the injection $|k\rangle \mapsto |k\rangle$ and $\pi : \mathbb{C}^{2m+m'} \to \mathbb{C}^{2m}$ is the projector s.t. $\pi|k\rangle = |k\rangle$ when $k < 2m$ and $\pi|k\rangle = 0$ otherwise. Thus there exists two unitaries Q, Q' s.t. $[\![T_2]\!]_{\text{pp}} = (I \oplus Q') \circ [\![T_1]\!]_{\text{pp}} \circ (I \oplus Q)$ (see [15]).

By Proposition 26, there exist two circuits T_{in} and T_{out} in PPRS triangular normal form such that $[\![T_{\text{in}}]\!]_{\text{pp}} = Q$ and $[\![T_{\text{out}}]\!]_{\text{pp}} = Q'$. Using the equational theory we can then make T_{in} and T_{out} appear, turning D_1^{NF} into



Since by construction, the middle part has the same single-photon semantics as T_2, by Proposition 30 we can transform it into T_2 using the axioms of the LO$_\text{v}$-calculus, which means transforming D_1^{NF} into D_2^{NF}. The result follows by transitivity. ◀

6 Conclusion

In this paper, we presented the LO$_\text{v}$-calculus, a graphical language for LOQC capturing most of the components typically considered in the physics community for linear optical quantum circuits. The language comes equipped with a sound and complete semantics, and we discussed how it provides a unifying framework for many of the existing approaches in the literature. We explained how several existing results can be ported in the LO$_\text{v}$ framework.

An obvious direction for future work is to extend the language to allow for sources and detectors of a non-zero number of photons. A more exploratory research avenue is to add support for features such as squeezed states or continuous variables.

References

1 Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. *Theory of Computing*, 9(4):143–252, 2013. doi:10.4086/toc.2013.v009a004.

2 Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments, 2016. arXiv:1612.05903.

3 Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A. Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

4 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. *Electronic Proceedings in Theoretical Computer Science*, 287:23–42, 2019.

5 Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski, and John van de Wetering. There and back again: A circuit extraction tale. *Quantum*, 5:421, 2021.

6 Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Terry Rudolph, and Chris Sparro. Fusion-based quantum computation, 2021. arXiv:2101.09310.

7 Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, 2014. Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84. doi:10.1016/j.tcs.2014.05.025.

8 Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin T. Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI'20*, pages 286–300. ACM, 2020. doi:10.1145/3385412.3386007.

9 Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 15(2):159–163, 2019. doi:10.1038/s41567-018-0318-2.

10 Cyril Branciard, Alexandre Clément, Mehdi Mhalla, and Simon Perdrix. Coherent control and distinguishability of quantum channels via PBS-diagrams. In Filippo Bonchi and Simon J. Puglisi, editors, *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPIcs*, pages 22:1–22:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.22.

11 Titouan Carette, Dominic Horsman, and Simon Perdrix. SZX-calculus: Scalable graphical quantum reasoning. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.55.

12 Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of graphical languages for mixed states quantum mechanics. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 108:1–108:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.108.

13 Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. An automated deductive verification framework for circuit-building quantum programs. In Nobuko Yoshida, editor, *Programming Languages and Systems*, volume 12648 of *Lecture Notes in Computer Science (LNCS)*, pages 148–177, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-72019-3_6.

14 Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. A complete equational theory for quantum circuits, 2022. doi:10.48550/ARXIV.2206.10577.

15 Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. LOv-calculus: A graphical language for linear optical quantum circuits, 2022. arXiv:2204.11787.

16 Alexandre Clément and Simon Perdrix. PBS-calculus: A graphical language for coherent control of quantum computations. In Javier Esparza and Daniel Kráľ, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:14, Dagstuhl, Germany, August 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.24.

17 Alexandre Clément and Simon Perdrix. Minimising resources of coherently controlled quantum computations, 2022. Accepted at MFCS 2022. arXiv:2202.05260.

18 William R. Clements, Peter C. Humphreys, Benjamin J. Metcalf, W. Steven Kolthammer, and Ian A. Walmsley. Optimal design for universal multiport interferometers. *Optica*, 3(12):1460–1465, December 2016. doi:10.1364/OPTICA.3.001460.

19 Robin Cockett and Cole Comfort. The category TOF. In Peter Selinger and Giulio Chiribella, editors, *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018*, volume 287 of *EPTCS*, pages 67–84, 2019.

20 Robin Cockett, Cole Comfort, and Priyaa Srinivasan. The category CNOT. In Peter Selinger and Giulio Chiribella, editors, *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018*, volume 287 of *EPTCS*, pages 258–293, 2019. doi:10.4204/EPTCS.266.18.

21 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.* Cambridge University Press, 2017. doi:10.1017/9781316219317.

22 Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: Classical, probabilistic, and quantum effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 833–845, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009859.

23 Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery. *Quantum*, 4:218, 2020. arXiv:1704.08670.

24 Giovanni de Felice and Bob Coecke. Quantum linear optics via string diagrams, 2022. arXiv:2204.12985.

25 Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. Graph-theoretic simplification of quantum circuits with the ZX-calculus. *Quantum*, 4:279, 2020.

26 Ross Duncan and Maxime Lucas. Verifying the Steane code with Quantomatic. In Bob Coecke and Matty J. Hoban, editors, *Proceedings of the 10th International Workshop on Quantum Physics and Logic, QPL 2013*, volume 171 of *EPTCS*, pages 33–49, 2013. doi:10.4204/EPTCS.171.4.

27 Ross Duncan and Simon Perdrix. Rewriting measurement-based quantum computations with generalised flow. In *International Colloquium on Automata, Languages, and Programming*, pages 285–296. Springer, 2010.

28 Artur K. Ekert. Quantum cryptography based on Bell's theorem. *Physical Review Letters*, 67:661–663, August 1991. doi:10.1103/PhysRevLett.67.661.

29 Yuan Feng, Ernst Moritz Hahn, Andrea Turrini, and Lijun Zhang. QPMC: a model checker for quantum programs and protocols. In Nikolaj Bjørner and Frank S. de Boer, editors, *Proceedings of the 20th International Symposium on Formal Methods (FM 2015)*, volume 9109 of *Lecture Notes in Computer Science*, pages 265–272. Springer, 2015. doi:10.1007/978-3-319-19249-9_17.

30 Richard P. Feynman and Albert R. Hibbs. *Quantum Mechanics and Path Integrals.* McGraw-Hill Publishing Company, 1965.

31 Liam Garvie and Ross Duncan. Verifying the smallest interesting colour code with quantomatic. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 266:147–163, 2018.

32 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In Hans-Juergen Boehm and Cormac Flanagan, editors, *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13*, pages 333–342. ACM, 2013. doi:10.1145/2491956.2462177.

33 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237866.

34 Nicolas Heurtel, Andreas Fyrillas, Grégoire de Gliniasty, Raphaël Le Bihan, Sébastien Malherbe, Marceau Pailhas, Boris Bourdoncle, Pierre-Emmanuel Emeriau, Rawad Mezher, Luka Music, et al. Perceval: A software platform for discrete variable photonic quantum computing, 2022. arXiv:2204.00602.

35 Anne Hillebrand. *Quantum Protocols involving Multiparticle Entanglement and their Representations in the zx-calculus.* PhD thesis, University of Oxford, 2011.

36 Christian Hutslar, Jacques Carette, and Amr Sabry. A library of reversible circuit transformations (work in progress). In Jarkko Kari and Irek Ulidowski, editors, *Proceedings of the 10th International Conference on Reversible Computation, RC 2018*, volume 11106 of *Lecture Notes in Computer Science*, pages 339–345. Springer, 2018. doi:10.1007/978-3-319-99498-7_24.

37 Ali Javadi-Abhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45:2–17, 2015. doi:10.1016/j.parco.2014.12.001.

38 Michio Jimbo. Introduction to the Yang-Baxter equation. *International Journal of Modern Physics A*, 4(15):3759–3777, 1989.

39 Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. Strawberry fields: A software platform for photonic quantum computing. *Quantum*, 3:129, 2019.

40 Emanuel Knill, Raymond Laflamme, and Gerald J. Milburn. A scheme for efficient quantum computation with linear optics. *Nature*, 409(6816):46–52, 2001. doi:10.1038/35051009.

41 Pieter Kok and Brendon W. Lovett. *Introduction to Optical Quantum Information Processing.* Cambridge University Press, 2010.

42 Pieter Kok, William J. Munro, Kae Nemoto, Timothy C. Ralph, Jonathan P. Dowling, and Gerald J. Milburn. Linear optical quantum computing with photonic qubits. *Reviews of Modern Physics*, 79:135–174, January 2007. doi:10.1103/RevModPhys.79.135.

43 Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. Projection-based runtime assertions for testing and debugging quantum programs. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):150:1–150:29, 2020. doi:10.1145/3428218.

44 Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106, 1965. doi:bams/1183526392.

45 Justin Makary, Neil J. Ross, and Peter Selinger. Generators and relations for real stabilizer operators. In Chris Heunen and Miriam Backens, editors, *Proceedings of hte 18th International Conference on Quantum Physics and Logic, QPL 2021*, volume 343 of *EPTCS*, pages 14–36, 2021. doi:10.4204/EPTCS.343.2.

46 Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17*, pages 846–858. ACM, 2017. doi:10.1145/3009837.3009894.

47 Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.

48 Mathias Pont, Riccardo Albiero, Sarah E. Thomas, Nicolò Spagnolo, Francesco Ceccarelli, Giacomo Corrielli, Alexandre Brieussel, Niccolo Somaschi, Hêlio Huet, Abdelmounaim Harouri, Aristide Lemaître, Isabelle Sagnes, Nadia Belabas, Fabio Sciarrino, Roberto Osellame, Pascale Senellart, and Andrea Crespi. Quantifying n-photon indistinguishability with a cyclic integrated interferometer, 2022. arXiv:2201.13333.

49 Michael Reck, Anton Zeilinger, Herbert J. Bernstein, and Philip Bertani. Experimental realization of any discrete unitary operator. *Physical Review Letters*, 73:58–61, July 1994. doi:10.1103/PhysRevLett.73.58.

50 Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004. doi:10.1017/S0960129504004256.

51 Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi:10.1109/SFCS.1994.365700.

52 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

53 Yulin Wu, Wan-Su Bao, Sirui Cao, Fusheng Chen, Ming-Cheng Chen, Xiawei Chen, Tung-Hsun Chung, Hui Deng, Yajie Du, Daojin Fan, et al. Strong quantum computational advantage using a superconducting quantum processor. *Physical Review Letters*, 127(18):180501, 2021.

54 Han-Sen Zhong, Yu-Hao Deng, Jian Qin, Hui Wang, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Dian Wu, Si-Qiu Gong, Hao Su, et al. Phase-programmable Gaussian boson sampling using stimulated squeezed light. *Physical Review Letters*, 127(18):180502, 2021.

55 Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

Resource Optimisation of Coherently Controlled Quantum Computations with the PBS-Calculus

Alexandre Clément
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Simon Perdrix
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

Coherent control of quantum computations can be used to improve some quantum protocols and algorithms. For instance, the complexity of implementing the permutation of some given unitary transformations can be strictly decreased by allowing coherent control, rather than using the standard quantum circuit model. In this paper, we address the problem of optimising the resources of coherently controlled quantum computations. We refine the PBS-calculus, a graphical language for coherent control which is inspired by quantum optics. In order to obtain a more resource-sensitive language, it manipulates abstract gates – that can be interpreted as queries to an oracle – and more importantly, it avoids the representation of useless wires by allowing unsaturated polarising beam splitters. Technically the language forms a coloured PROP. The language is equipped with an equational theory that we show to be sound, complete, and minimal.

Regarding resource optimisation, we introduce an efficient procedure to minimise the number of oracle queries of a given diagram. We also consider the problem of minimising both the number of oracle queries and the number of polarising beam splitters. We show that this optimisation problem is NP-hard in general, but introduce an efficient heuristic that produces optimal diagrams when at most one query to each oracle is required.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Axiomatic semantics; Theory of computation → Categorical semantics

Keywords and phrases Quantum computing, Graphical language, Coherent control, Completeness, Resource optimisation, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.36

Related Version *Preprint Version*: `https://arxiv.org/abs/2202.05260` [13]

Funding This work is funded by ANR-17-CE25-0009 SoftQPro, ANR-17-CE24-0035 VanQuTe, PIA-GDN/Quantex, and LUE/UOQ, the PEPR integrated project EPiQ ANR-22-PETQ-0007 part of Plan France 2030, and by *"Investissements d'avenir"* (ANR-15-IDEX-02) program of the French National Research Agency; the European Project NExt ApplicationS of Quantum Computing (NEASQC), funded by Horizon 2020 Program inside the call H2020-FETFLAG-2020-01 (Grant Agreement 951821), and the HPCQS European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101018180.

1 Introduction

Most models of quantum computation (like quantum circuits) and most quantum programming languages are based on the *quantum data/classical control* paradigm. In other words, based on a set of quantum primitives (e.g. unitary transformations, quantum measurements), the way these primitives are applied on a register of qubits is either fixed or classically controlled.

However, quantum mechanics offers more general control of operations: for instance in quantum optics it is easy to control the trajectory of a system, like a photon, based on its polarisation using a *polarising beam splitter*. One can then position distinct quantum primit-

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 36; pp. 36:1–36:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

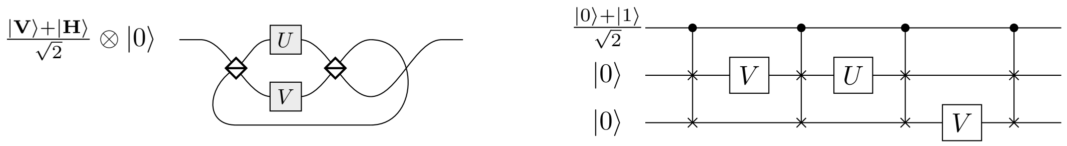
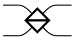

$\frac{|\mathbf{V}\rangle+|\mathbf{H}\rangle}{\sqrt{2}} \otimes |0\rangle$
U
V
$\frac{|0\rangle+|1\rangle}{\sqrt{2}}$
$|0\rangle$
$|0\rangle$
V
U
V

Figure 1 [*Left*] Coherently controlled quantum computation for solving the commuting problem. Only two queries are used: one query to U and one query to V. [*Right*] Optimal circuit for solving the commuting problem, where the 3-qubit gate is a control-swap. Notice that three queries are necessary in the quantum circuit model.

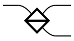
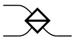
ives on the distinct trajectories. Since the polarisation of a photon can be in superposition, it achieves some form of quantum control, called coherent control: the quantum primitives are applied in superposition depending on the state of another quantum system. Coherent control is not only a subject of interest for foundations of quantum mechanics [21, 26, 33], it also leads to advantages in solving computational problems [18, 4, 14, 27] and in designing more efficient protocols [19, 9, 1, 17, 20].

Indeed, some problems can be solved more efficiently by using coherent control rather than the usual quantum circuits. This separation has been proved in a multi-oracle model where the measure of complexity is the number of queries to (a single or several distinct) oracles, which are generally unitary maps. The simplest example is the following problem [9]: given two oracles U and V with the promise that they are either commuting or anti-commuting, decide whether U and V are commuting or not. This problem can be solved using the so-called quantum switch [10] which can be implemented using only two queries by means of coherent control, whereas solving this problem requires at least 3 queries (e.g. two queries to U and one query to V) in the quantum circuit model (see Figure 1).

In this paper, we address the problem of optimising resources of coherently controlled quantum computations. To do so, we first refine the framework of the PBS-calculus – a graphical language for coherently controlled quantum computation – to make it more resource-sensitive. Then, we consider the problem of optimising the number of queries, and also the number of polarising beam splitters, of a given coherently controlled quantum computation, described as a PBS-diagram.

PBS-calculus. The PBS-calculus is a graphical language that has been introduced [12] to represent and reason about quantum computations involving coherent control of quantum operations. Inspired by quantum optics [11], the polarising beam splitter (PBS for short), denoted  is at the heart of the language: when a photon enters the PBS, say from the top left, it is reflected (and hence outputted on the top right) if its polarisation is vertical; or transmitted (and hence outputted on the bottom right) if its polarisation is horizontal. If the polarisation is a superposition of vertical and horizontal, the photon is outputted in a superposition of two positions. As a consequence, the trajectory of a particle, say a photon, will depend on its polarisation. The second main ingredient of the PBS-calculus are the gates, denoted $-\boxed{U}-$ which applies some transformation U on a data register. Notice that the gates never act on the polarisation of the particle.

PBS-diagrams, which form a traced symmetric monoidal category (more precisely a traced prop [24]), are equipped with an equational theory that allows one to transform a diagram. The equational theory has been proved to be sound, complete, and minimal [12].

Notice that a PBS-diagram may have some useless wires, like in the example of the "half quantum switch", see Figure 2 (left). We refine the PBS-calculus in order to allow one to remove these useless wires, leading to unsaturated PBS (or 3-leg PBS) like  or .

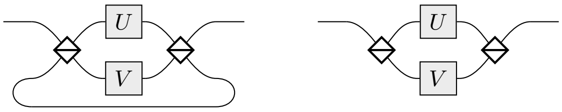
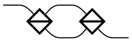


Figure 2 A coherent control of U and V, also called a half quantum switch: when the initial polarisation is vertical ($\mathbf{V}$), U is applied on the data register, when the polarisation is horizontal ($\mathbf{H}$), V is applied. Whatever the polarisation is, the particle always goes out of the top port of the second beam splitter. On the right-hand side the diagram is made of beam splitters with a missing leg, whereas on the left-hand side standard beam splitters are used, and a useless trace is added.

To avoid ill-formed diagrams like , a typing discipline is necessary. To this end, we use the framework of coloured props: each wire has 3 possible colours: black, red and blue which can be interpreted as follows: a photon going through a blue (resp. red) wire must have a horizontal (resp. vertical) polarisation.

The introduction of unsaturated polarising beam splitters requires to revisit the equational theory of the PBS-calculus. The heart of the refined equational theory is the axiomatisation of the 3-leg polarising beam splitters, together with some additional equations which govern how 4-leg polarising beam splitters can be decomposed into 3-leg ones. To show the completeness of the refined equational theory, we introduce normal forms and show that any diagram can be put in normal form. Finally, we also show the minimality of the equational theory by proving that none of the equations can be derived from the other ones.

Resource Optimisation. The PBS-calculus, thanks to its refined equational theory, provides a way to detect and remove dead-code in a diagram. We exploit this property to address the crucial question of resource optimisation. We introduce a specific form of diagrams that minimises the number of gates, more precisely the number of queries to oracles, with an appropriate modelisation of oracles. We provide an efficient procedure to transform any diagram into this specific form. We then focus on the problem of optimising both the number of queries and the number of polarising beam splitters. We refine the previous procedure, leading to an efficient heuristic. We show that the produced diagrams are optimal when every oracle is queried at most once, but might not be optimal in general. We actually show that the general optimisation problem is NP-hard using a reduction from the *maximum Eulerian cycle decomposition problem* [8].

Related work. Several languages have been designed to represent coherently controlled quantum computation: some of them are extensions of quantum circuits, and other diagrammatic languages [30, 5, 31, 29]; others are based on abstract programming languages [2, 32, 16, 15, 6]. While there are numerous works on resource-optimisation of quantum computation, in particular for quantum circuits [23, 3, 25], there was, up to our knowledge, no procedure for resource optimisation of coherently controlled quantum computation.

All omitted proofs can be found in the preprint version of the paper [13].

2 Coloured PBS-diagrams

We use the formalism of traced coloured props (i.e. small traced symmetric strict monoidal categories whose objects are freely spanned by the elements of a set of colours) to represent coherently controlled quantum computations. We are going to use the “colours” $\mathbf{v}$, $\mathbf{h}$, $\top$, to denote respectively vertical, horizontal or possibly both polarisations.

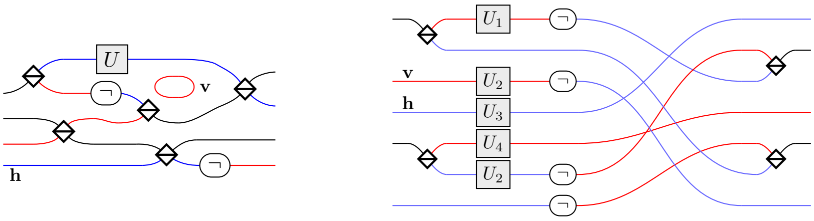
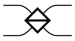
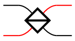
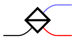
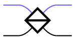
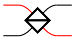
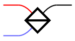
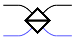
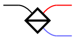
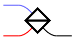
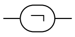
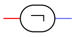
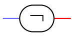
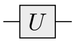
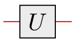
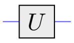
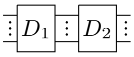
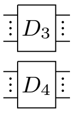
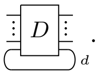


Figure 3 (*Left*) An example of diagram of type $\top \oplus \top \oplus \mathbf{v} \oplus \mathbf{h} \to \top \oplus \mathbf{h} \oplus \top \oplus \mathbf{v}$. (*Right*) An example of a diagram of type $\top \oplus \mathbf{v} \oplus \mathbf{h} \oplus \top \oplus \mathbf{h} \to \mathbf{h} \oplus \top \oplus \mathbf{v} \oplus \top \oplus \mathbf{h}$, in a particular form that we will call *normal form* (see Definition 15).

▶ **Definition 1.** *Given a monoid* M*, let* $\mathbf{Diag}^{\mathsf{M}}$ *be the traced coloured prop with colours* $\{\mathbf{v}, \mathbf{h}, \top\}$ *freely generated by the following generators, for any* $U \in \mathsf{M}$*:*

	: $\top \oplus \top \to \top \oplus \top$			: $\top \oplus \mathbf{v} \to \mathbf{v} \oplus \top$			: $\top \to \mathbf{h} \oplus \mathbf{v}$
	: $\mathbf{h} \oplus \top \to \mathbf{h} \oplus \top$			: $\mathbf{v} \oplus \top \to \top \oplus \mathbf{v}$			: $\mathbf{v} \oplus \mathbf{h} \to \top$
	: $\top \oplus \mathbf{h} \to \top \oplus \mathbf{h}$			: $\top \to \mathbf{v} \oplus \mathbf{h}$			: $\mathbf{h} \oplus \mathbf{v} \to \top$
	: $\top \to \top$			: $\mathbf{v} \to \mathbf{h}$			: $\mathbf{h} \to \mathbf{v}$
	: $\top \to \top$			: $\mathbf{v} \to \mathbf{v}$			: $\mathbf{h} \to \mathbf{h}$

The morphisms of $\mathbf{Diag}^{\mathsf{M}}$ are called M-diagrams or simply diagrams when M is irrelevant or clear from the context. Intuitively, the diagrams are inductively obtained by composition of the generators from Definition 1 using the sequential composition $D_2 \circ D_1$, the parallel composition $D_3 \oplus D_4$, and the trace $Tr_d(D)$ which are respectively depicted as follows:    . Notice that these compositions should type-check, i.e. $D_1 : a \to b$, $D_2 : b \to c$ and $D : a \oplus d \to b \oplus d$ with $d \in \{\top, \mathbf{v}, \mathbf{h}\}$. The axioms of the traced coloured prop guarantee that the diagrams are defined up to deformation: two diagrams whose graphical representations are isomorphic are equal.

Regarding notations, we use actual colours for wires: blue for $\mathbf{h}$-wires, red for $\mathbf{v}$-wires, and black for $\top$-wires. We also add labels on the wires, which are omitted when clear from the context, so that there is no loss of information in the case of a colour-blind reader or black and white printing. Two examples of diagrams are given in Figure 3.

Unless specified, the unit of M is denoted I and its composition is $\cdot$ which will be generally omitted (VU rather than $V \cdot U$). The main two examples of monoids we consider in the rest of the paper are:

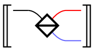
- The monoid $\mathcal{U}(\mathcal{H})$ of isometries of a Hilbert space $\mathcal{H}$ with the usual composition. When $\mathcal{H}$ is of finite dimension, the elements of $\mathcal{U}(\mathcal{H})$ are unitary maps. With a slight abuse of notations, the corresponding traced coloured prop of diagrams is denoted $\mathbf{Diag}^{\mathcal{H}}$.
- The free monoid $\mathcal{G}^*$ on some set $\mathcal{G}$. The gates, when the monoid is freely generated, can be interpreted as queries to oracles (each element of $\mathcal{G}$ corresponds to an oracle): the gates implement a priori arbitrary operations with no particular structures. We use the term *abstract diagram* when the underlying monoid is freely generated. Notice that the free monoid case can also be seen as an extension of the *bare diagrams* [7] whose gates are labelled with *names*.

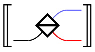
3 Semantics

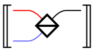
The input of a diagram is a single particle, which has a polarisation, a position and a data register. A basis state for the polarisation is either vertical or horizontal, and a basis state for the position is an integer which corresponds to the wire on which the particle is located. The type of a diagram restricts the possible input/output configurations: if $D : \mathbf{v} \oplus \top \to \mathbf{h} \oplus \mathbf{h} \oplus \mathbf{v}$ then the possible input (resp. output) configurations are the following polarisation-position pairs: $\{(\mathbf{V}, 0), (\mathbf{V}, 1), (\mathbf{H}, 1)\}$ (resp. $\{(\mathbf{H}, 0), (\mathbf{H}, 1), (\mathbf{V}, 2)\}$). More generally for any object a, let $[a]$ be the set of possible configurations, and $|a|$ be its size, inductively defined as follows: $|I| = 0$, $|a \oplus \top| = |a \oplus \mathbf{v}| = |a \oplus \mathbf{h}| = |a| + 1$, and $[I] = \emptyset$, $[a \oplus \mathbf{v}] = [a] \cup \{(\mathbf{V}, |a|)\}$, $[a \oplus \mathbf{h}] = [a] \cup \{(\mathbf{H}, |a|)\}$ and $[a \oplus \top] = [a] \cup \{(\mathbf{V}, |a|), (\mathbf{H}, |a|)\}$.

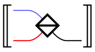
The semantics of a M-diagram $D : a \to b$ is a map $[a] \to [b] \times \mathsf{M}$ which associates with an input configuration (c, p), an output configuration (c', p') and a side effect $U_k \ldots U_1 \in \mathsf{M}$ which represents the action applied on a data register of the particle. Thus the semantics of a diagram can be formulated as follows:

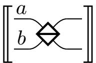
▶ **Definition 2.** *Given an M-diagram $D : a \to b$, let $\llbracket D \rrbracket : [a] \to [b] \times \mathsf{M}$ be inductively defined as: $\forall D_1 : a \to b, D_2 : b \to d, D_3 : d \to e, D_4 : a \oplus f \to b \oplus f$, where $f \in \{\top, \mathbf{v}, \mathbf{h}\}$:*

$\llbracket$  $\rrbracket = \begin{cases} (\mathbf{V}, 0) \mapsto ((\mathbf{V}, 0), I) \\ (\mathbf{H}, 0) \mapsto ((\mathbf{H}, 1), I) \end{cases}$

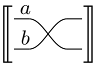
$\llbracket$  $\rrbracket = \begin{cases} (\mathbf{V}, 0) \mapsto ((\mathbf{V}, 1), I) \\ (\mathbf{H}, 0) \mapsto ((\mathbf{H}, 0), I) \end{cases}$

$\llbracket$  $\rrbracket = \begin{cases} (\mathbf{V}, 0) \mapsto ((\mathbf{V}, 0), I) \\ (\mathbf{H}, 1) \mapsto ((\mathbf{H}, 0), I) \end{cases}$

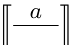
$\llbracket$  $\rrbracket = \begin{cases} (\mathbf{V}, 1) \mapsto ((\mathbf{V}, 0), I) \\ (\mathbf{H}, 0) \mapsto ((\mathbf{H}, 0), I) \end{cases}$

$\llbracket$ 

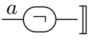
$\rrbracket = (c, p) \mapsto \begin{cases} ((c, p), I) & \text{if } c = \mathbf{V} \\ ((c, 1 - p), I) & \text{if } c = \mathbf{H} \end{cases}$

$\llbracket$ 

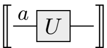
$\rrbracket = (c, p) \mapsto ((c, 1 - p), I)$

$\llbracket$ 

$\rrbracket = (c, 0) \mapsto ((c, 0), I)$

$\llbracket$ 

$\rrbracket = \begin{cases} (\mathbf{V}, 0) \mapsto ((\mathbf{H}, 0), I) \\ (\mathbf{H}, 0) \mapsto ((\mathbf{V}, 0), I) \end{cases}$

$\llbracket$ 

$\rrbracket = (c, 0) \mapsto ((c, 0), U)$

$$\llbracket D_1 \oplus D_3 \rrbracket = (c, p) \mapsto \begin{cases} \llbracket D_1 \rrbracket (c, p) & \textit{if } p < |a| \\ S_a(\llbracket D_3 \rrbracket (c, p - |a|)) & \textit{otherwise} \end{cases} \qquad \llbracket D_2 \circ D_1 \rrbracket = \llbracket D_2 \rrbracket \circ \llbracket D_1 \rrbracket$$

$$\llbracket Tr_f(D_4) \rrbracket = (c, p) \mapsto \begin{cases} \llbracket D_4 \rrbracket (c, p) \quad \textit{if } \pi_{pos}(\llbracket D_4 \rrbracket (c, p)) < |b| \\ \llbracket D_4 \rrbracket \circ S_{a-b}(\llbracket D_4 \rrbracket (c, p)) \\ \qquad \textit{if } \pi_{pos}(\llbracket D_4 \rrbracket \circ S_{a-b}(\llbracket D_4 \rrbracket (c, p))) < |b| \leq \pi_{pos}(\llbracket D_4 \rrbracket (c, p)) \\ \llbracket D_4 \rrbracket \circ S_{a-b}(\llbracket D_4 \rrbracket \circ S_{a-b}(\llbracket D_4 \rrbracket (c, p))) \quad \textit{otherwise} \end{cases}$$

where the composition is: $g \circ f(c, p) = ((c'', p''), U'U)$ with $f(c, p) = ((c', p'), U)$ and $g(c', p') = ((c'', p''), U')$; $\pi_{pos} : [a] \times \mathsf{M} \to \mathbb{N} = ((c, p), U) \mapsto p$ is the projector on the position, and $S_a : [b] \times \mathsf{M} \to [a \oplus b] \times \mathsf{M} = ((c, p), U) \mapsto ((c, p + |a|), U)$ and $S_{a-b} : [b] \times \mathsf{M} \to [a] \times \mathsf{M} = ((c, p), U) \mapsto ((c, p + |a| - |b|), U)$ shift the position.

Given $D : a \to b$ and $(c, p) \in [a]$, we denote respectively by $c^D_{c,p}$, $p^D_{c,p}$ and $U^D_{c,p}$ the polarisation, the position and the element of M, such that $\llbracket D \rrbracket (c, p) = ((c^D_{c,p}, p^D_{c,p}), U^D_{c,p})$. In the case where M is the free monoid $\mathcal{G}^*$, its elements can be seen as words, so we will use the notation $w^D_{c,p}$ instead of $U^D_{c,p}$.

Notice that the semantics of the trace is not defined as a fixed point but as a finite number of unfoldings. Indeed, like for PBS-diagrams, one can show that any wire of a diagram is used at most twice, each time with a distinct polarisation.

▶ **Proposition 3.** $[\![.]\!]$ *is well defined, i.e. the axioms of the traced coloured prop are sound and the semantics of the trace is well defined.*

3.1 Quantum semantics

Any diagram whose underlying monoid consists of linear maps, admits a *quantum semantics* defined as follows:

▶ **Definition 4** (Quantum semantics). *Given a monoid* M *of linear maps (with the standard composition) on a complex vector space* $\mathcal{V}$*, for any* M*-diagram* $D : a \to b$ *the quantum semantics of* D *is the linear map* $V_D : \mathbb{C}^{[a]} \otimes \mathcal{V} \to \mathbb{C}^{[b]} \otimes \mathcal{V} = |c,p\rangle \otimes |\phi\rangle \mapsto \left|c^D_{c,p}, p^D_{c,p}\right\rangle \otimes U^D_{c,p}\,|\phi\rangle$.

The diagrams in $\mathbf{Diag}^{\mathcal{H}}$ are valid by construction, in the sense that their semantics are valid quantum evolutions:

▶ **Proposition 5.** *For any* $D \in \mathbf{Diag}^{\mathcal{H}}$*,* $V_D : \mathbb{C}^{[a]} \otimes \mathcal{H} \to \mathbb{C}^{[b]} \otimes \mathcal{H}$ *is an isometry.*

Note that $[\![D]\!] = [\![D']\!]$ implies $V_D = V_{D'}$; the converse is true if and only if $0 \notin \mathsf{M}$:

▶ **Proposition 6.** *Given a monoid* M *of complex linear maps, we have* $\forall D, D', [\![D]\!] = [\![D']\!] \Leftrightarrow V_D = V_{D'}$*, if and only if* $0 \notin \mathsf{M}$.

In particular, two diagrams in $\mathbf{Diag}^{\mathcal{H}}$ have the same action semantics if and only if they have the same quantum semantics.

3.2 Interpretation

Given a monoid homomorphism $\gamma : \mathsf{M} \to \mathsf{M}'$, one can transform any M-diagram into a M'-diagram straightforwardly, by applying γ on each gate of the diagram:

▶ **Definition 7.** *Given a* M*-diagram* $D : a \to b$ *and a monoid homomorphism* $\gamma\colon \mathsf{M} \to \mathsf{M}'$*, we define its* γ*-interpretation* $\gamma(D) : a \to b$ *as the* M'*-diagram obtained by applying* γ *to each gate of* D*. It is defined inductively as:* $\gamma(\overset{a}{-}\boxed{U}- : a \to a) = \overset{a}{-}\boxed{\gamma(U)}- : a \to a$*, for any other generator* g*,* $\gamma(g) = g$*,* $\gamma(D_2 \circ D_1) = \gamma(D_2) \circ \gamma(D_1)$*,* $\gamma(D_1 \oplus D_2) = \gamma(D_1) \oplus \gamma(D_2)$*, and* $\gamma(Tr_e(D)) = Tr_e(\gamma(D))$.

▶ **Proposition 8.** *Any* M*-diagram is the interpretation of an abstract diagram.*

It is easy to see that the action of monoid homomorphisms on diagrams is well-behaved with respect to the semantics:

▶ **Proposition 9.** *Given any* M*-diagram* $D : a \to b$ *and any monoid homomorphism* $\gamma\colon \mathsf{M} \to \mathsf{M}'$*, for any configuration* $(c,p) \in [a]$*, if* $[\![D]\!]\,(c,p) = ((c',p'),U)$ *then* $[\![\gamma(D)]\!]\,(c,p) = ((c',p'),\gamma(U))$.

As a consequence, given two abstract diagrams $D_1, D_2 \in \mathbf{Diag}^{\mathcal{G}^*}$, if $[\![D_1]\!] = [\![D_2]\!]$ then for any homomorphism $\gamma : \mathcal{G}^* \to \mathsf{M}$, $[\![\gamma(D_1)]\!] = [\![\gamma(D_2)]\!]$. The converse is not true in general. Notice that in the framework of graphical languages an equation holds in graphical languages for traced symmetric (resp. dagger compact closed) monoidal categories if and only if it holds in finite-dimensional vector (resp. Hilbert) spaces [22, 28]. We prove a similar result by showing that interpreting abstract diagrams using 2-dimensional Hilbert spaces is enough to completely characterise their semantics:

▶ **Proposition 10.** *Given a Hilbert space $\mathcal{H}$ of dimension at least 2 and a set $\mathcal{G}$, $\forall D_1, D_2 \in$ $\mathbf{Diag}^{\mathcal{G}^*}$, there exists a monoid homomorphism $\gamma\colon \mathcal{G}^* \to \mathcal{U}(\mathcal{H})$ s.t. $[\![D_1]\!] = [\![D_2]\!] \Leftrightarrow [\![\gamma(D_1)]\!] =$ $[\![\gamma(D_2)]\!]$.*

A stronger result, where the homomorphism γ is independent of the diagrams, is also true, assuming the axiom of choice:

▶ **Proposition 11.** *Given a Hilbert space $\mathcal{H}$ of dimension at least 2, and a set $\mathcal{G}$ of cardinality at most the cardinality of $\mathcal{U}(\mathcal{H})$, there exists a monoid homomorphism $\gamma\colon \mathcal{G}^* \to \mathcal{U}(\mathcal{H})$ s.t. $\forall D_1, D_2 \in \mathbf{Diag}^{\mathcal{G}^*}$, $[\![D_1]\!] = [\![D_2]\!] \Leftrightarrow [\![\gamma(D_1)]\!] = [\![\gamma(D_2)]\!]$.*

▶ Remark 12. Notice that the cardinality of $\mathcal{U}(\mathcal{H})$ is $\max(2^{\aleph_0}, 2^{\dim(\mathcal{H})})$ (where $2^{\aleph_0}$ is the cardinality of $\mathbb{R}$).

4 Equational theory

In this section, we introduce an equational theory which allows one to transform any M-diagram into an equivalent one. Indeed, all the equations we present in this section preserve the semantics of the diagrams (see Proposition 14).

These equations are summarised in Figure 4. They form what we call the CPBS-calculus:

▶ **Definition 13** (CPBS-calculus)**.** *Two* M*-diagrams D_1, D_2 are equivalent according to the rules of the* CPBS*-calculus, denoted* CPBS $\vdash D_1 = D_2$*, if one can transform D_1 into D_2 using the equations given in Figure 4. More precisely,* CPBS $\vdash \cdot = \cdot$ *is defined as the smallest congruence which satisfies equations of Figure 4 in addition to the axioms of coloured traced prop.*

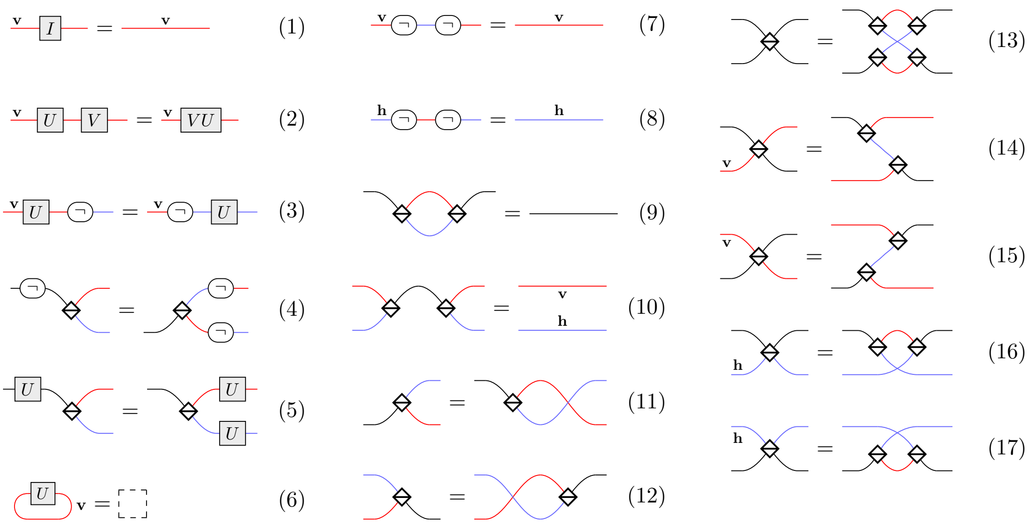


Figure 4 Axioms of the CPBS-calculus. $U, V \in$ M. Equations (1) and (2) reflect the monoid structure of M; Equations (3) to (5) show how the three generators commute; Equation (6) means that a disconnected diagram (with no inputs/outputs) can be removed; Equations (7) to (10) witness the fact that the negation and the 3-leg PBS are invertible; Equations (11) and (12) are essentially topological rules; Equations (13) to (17) show how 4-leg PBS can be decomposed into 3-leg PBS. Notice in particular that the other rules do not use 4-leg PBS, as a consequence one could define the language using 3-leg PBS only and see the 4-leg PBS as syntactic sugar.

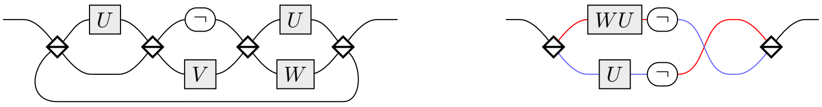


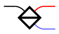
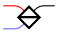
Figure 5 An example of a CPBS-diagram (*left*) and its equivalent diagram in normal form (*right*).

Notice that the CPBS-calculus subsumes the PBS-calculus: the fragment of monochromatic (black) $\mathcal{H}$-diagrams of the CPBS-calculus coincides with the set of PBS-diagrams, moreover, for any two PBS-diagrams D_1, D_2, PBS $\vdash D_1 = D_2$ if and only if CPBS $\vdash D_1 = D_2$.

▶ **Proposition 14** (Soundness). *For any two* M*-diagrams D_1 and D_2, if* CPBS $\vdash D_1 = D_2$ *then* $[\![D_1]\!] = [\![D_2]\!]$.

We introduce normal forms, that will be useful to prove that the equational theory is complete, and will also play a role in optimising the number of gates in a diagram in Section 5.

▶ **Definition 15.** *A diagram is said to be in* normal form *if it is of the form $M \circ P \circ F \circ G \circ S$, where:*

- *S is of the form $b_1 \oplus \cdots \oplus b_n$, where each b_i is either* $\overset{\mathbf{v}}{—}$, $\overset{\mathbf{h}}{—}$ *or* 
- *G is of the form $g_1 \oplus \cdots \oplus g_k$, where each g_i is either* $\overset{\mathbf{v}}{—}$, $\overset{\mathbf{h}}{—}$, *or* $\overset{\mathbf{v}}{—}\boxed{U_i}—$ *or* $\overset{\mathbf{h}}{—}\boxed{U_i}—$ *with $U_i \neq I$*
- *F is of the form $n_1 \oplus \cdots \oplus n_k$, where each n_i is either* $\overset{\mathbf{v}}{—}$, $\overset{\mathbf{h}}{—}$, $\overset{\mathbf{v}}{—}\boxed{\neg}—$ *or* $\overset{\mathbf{h}}{—}\boxed{\neg}—$
- *P is a permutation of the wires, that is, a trace-free diagram in which all generators are identity wires or swaps*
- *M is of the form $w_1 \oplus \cdots \oplus w_m$, where each w_i is either* $\overset{\mathbf{v}}{—}$, $\overset{\mathbf{h}}{—}$ *or*  .

For example, the diagram shown in Figure 3 (right) is in normal form.

▶ **Theorem 16.** *For any* M*-diagram D, there exists a* M*-diagram in normal form N such that* CPBS $\vdash D = N$.

Note that the structure of the normal form as well as the proof of Theorem 16 use in an essential way the removal of useless wires made possible by the use of colours, and in particular Equation (10), which has no equivalent in the monochromatic PBS-calculus of [12]. An example of CPBS-diagram and its normal form are given in Figure 5.

Now we use the normal form to prove the completeness of the CPBS-calculus:

▶ **Lemma 17** (Uniqueness of the normal form). *For any two diagrams in normal form N and N', if $[\![N]\!] = [\![N']\!]$ then $N = N'$.*

▶ **Theorem 18** (Completeness). *Given any two* M*-diagrams D_1 and D_2, if $[\![D_1]\!] = [\![D_2]\!]$ then* CPBS $\vdash D_1 = D_2$.

Finally, each equation of Figure 4 is necessary for the completeness:

▶ **Theorem 19** (Minimality). *None of the equations of Figure 4 is a consequence of the others.*

5 Resource optimisation

We show in this section that the equational theory of the CPBS-calculus can be used for resource optimisation.

5.1 Minimising the number of oracle queries

We consider the problem of minimising the number of oracle queries: given a set $\mathcal{G}$ of (distinct) oracles and a $\mathcal{G}^*$-diagram D, the objective is to find a diagram D' equivalent to D (i.e. $[\![D]\!] = [\![D']\!]$) such that D' is using a minimal number of queries to each oracle. Since there are several oracles, the definition of the optimal diagrams should be made precise.

First, we define the number of queries to a given oracle:

▶ **Definition 20.** *Given a $\mathcal{G}^*$-diagram D, for any $U \in \mathcal{G}$, let $\#_U(D)$ be the number of queries to U in D, inductively defined as follows: $\#_U(\overset{a}{-}\boxed{w}-) = |w|_U$, $\#_U(g) = 0$ for all the other generators, and $\#_U(D_1 \oplus D_2) = \#_U(D_2 \circ D_1) = \#_U(D_1) + \#_U(D_2)$, $\#_U(Tr_a(D)) = \#_U(D)$, where $|w|_U$ is the number of occurrences of U in the word $w \in \mathcal{G}^*$.*

We can now define a query-optimal diagram as follows:

▶ **Definition 21.** *A $\mathcal{G}^*$-diagram D is query-optimal if $\forall D' \in \mathbf{Diag}^{\mathcal{G}^*}$, $\forall U \in \mathcal{G}$, $[\![D]\!] = [\![D']\!]$ implies $\#_U(D) \leq \#_U(D')$.*

Notice that given a diagram, it is not a priori guaranteed that there exists an equivalent diagram which is query-optimal, if for instance, all the diagrams which minimise the number of queries to some oracle U do not minimise the number of queries to another oracle V. We actually show (Proposition 23) that any diagram can be turned into a query-optimal one. To this end, we first need a lower-bound on the number of queries to a given oracle:

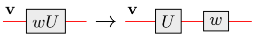
▶ **Proposition 22** (Lowerbound)**.** *For any $\mathcal{G}^*$-diagram $D : a \to b$ and any $U \in \mathcal{G}$, $\#_U(D) \geq \left\lceil \sum_{(c,p)\in X} \frac{|w^D_{c,p}|_U}{2} \right\rceil$ where $w^D_{c,p} \in \mathcal{G}^*$ is such that $[\![D]\!]\,(c,p) = (c', p', w^D_{c,p})$.*

Notice that Proposition 22 provides a lower bound on the minimal number of queries to U one can reach in optimising a diagram since the right-hand side of the inequality only depends on the semantics of the diagram.

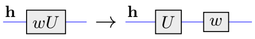
We are now ready to introduce an optimisation procedure that transforms any diagram into an equivalent query-optimal one:

Query optimisation procedure of a $\mathcal{G}^*$-diagram D.

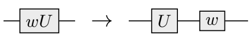
1. Transform D into its normal form D_{NF}. A recursive procedure for doing this can easily be deduced from the proof of Theorem 16.
2. Split all gates into elementary gates (that is, gates whose label is a single letter), using the following variants of Equation (2), which are consequences of the equations of Figure 4 (see [13]): $\forall U \in \mathcal{G}$, $\forall w \in \mathcal{G}^*$, $w \neq I$:



(18)

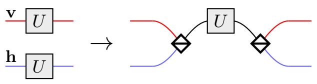


(19)

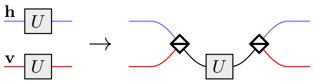


(20)

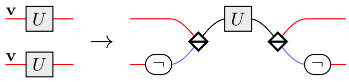
3. As long as the diagram contains two nonblack gates with the same label, merge them. To do so, deform the diagram to put one over the other, and apply one of the following equations, which are also consequences of the equations of Figure 4 :



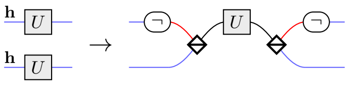
(21)



(22)



(23)



(24)

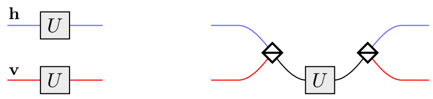


Figure 6 Two equivalent diagrams: the diagram on the left is optimal in terms of number of polarising beam splitters, the diagram on the right is optimal in terms of queries. Notice there is no equivalent diagram with no polarising beam splitter and at most a single query.

An example of query-optimised diagram is given in Figure 8. The query-optimisation procedure transforms any diagram into an equivalent query-optimal one:

▶ **Proposition 23.** *The diagram D_0 output by the query optimisation procedure is query-optimal: for any U and any D' s.t. $[\![D']\!] = [\![D_0]\!]$, one has $\#_U(D_0) \leq \#_U(D')$.*

Notice that the query-optimisation procedure is efficient: one can naturally define the size $|D|$ of a diagram $D \in \mathbf{Diag}^{\mathcal{G}^*}$ as follows: $|\overset{a}{-}\boxed{w}-| = |w|$, $|g| = 1$ for all the other generators, and $|D_1 \oplus D_2| = |D_2 \circ D_1| = |D_1| + |D_2|$, $|Tr_a(D)| = |D| + 1$. Step 1 of the procedure, which consists in putting the diagram in normal form, can be done using a number of elementary equations of Figure 4 which is quadratic in the size of the diagram, the other two steps being linear. Notice that here we only count the number of basic equations, but it requires also some diagrammatic transformations, which can be handled efficiently using appropriate data structures.

5.2 Optimising both queries and PBS

We refine the resource optimisation of a diagram by considering not only the number of queries but also the number of instructions, and in particular the number of polarising beam splitters. Notice that the number of beam splitters and the number of queries cannot be minimised independently, in the sense that there might not exist a diagram that is both query-optimal and PBS-optimal (see such an example in Figure 6). As the implementation of an oracle is a priori more expensive than the implementation of a single PBS, we optimise the number of queries and then the number of PBS in this order, i.e. the measure of complexity is the lexicographic order number of queries, number of polarising beam splitters.

▶ **Definition 24.** *A diagram D is query-PBS-optimal if D is query-optimal and for any query-optimal diagram D' equivalent to D (i.e. $[\![D]\!] = [\![D']\!]$), $\#_{\mathrm{PBS}}(D) \leq \#_{\mathrm{PBS}}(D')$, where $\#_{\mathrm{PBS}}(D)$ be the number of PBS of D.*

We introduce an efficient heuristic, called *PGT procedure* that, when applied on a query-optimal diagram D_0, preserves the number of queries. Moreover, the produced diagram, called in PGT form (see Figure 7), is query-PBS-optimal when there is at most one query to each oracle:

▶ **Theorem 25.** *Any query-optimal diagram in PGT form that does not contain two queries to the same oracle (i.e. $\forall U \in \mathcal{G}, \#_U(D) \leq 1$) is query-PBS-optimal.*

The procedure relies on equations of Figure 4, together with easy to derive variants of these equations. The procedure, with all steps detailed, more pictures and explicit statement of the variants of the equations, is given in the preprint version of the paper [13].

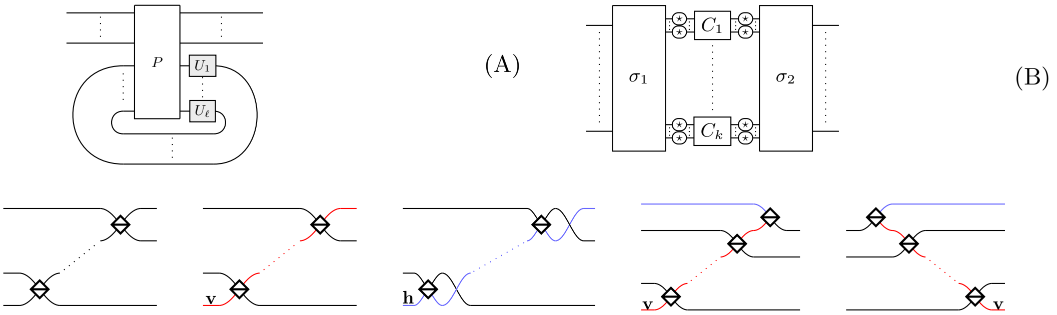
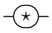
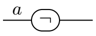
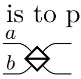
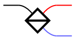
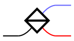
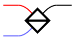
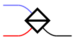


Figure 7 Schematic description of a diagram in PGT form (for Permutation, Gates and Traces). A diagram is in PGT form if it is of the form (A), with P of the form (B), and the C_i of the forms depicted on the second line.  denotes either or  with $a \in \{\mathbf{v}, \mathbf{h}\}$, and σ_1, σ_2 are permutations of the wires.

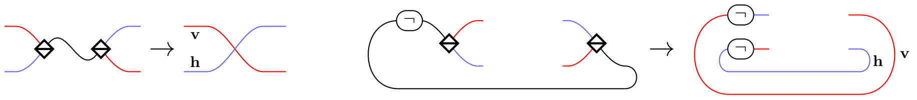
PGT procedure. Given a query optimal diagram D_0:

0. During all the procedure, every time there are two consecutive negations, we remove them using Equation (7), (8) or their all-black version.

1. Deform the gate-optimal diagram D_0 to put it in the form (A) with P gate-free. The goal of the following steps is to put P in stair form.

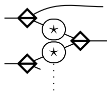
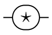
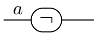
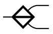
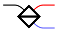
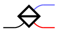
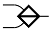
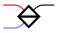
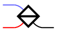
2. Split all PBS of the form  into combinations of , ,  and , using Equations (13) to (17).

3. As long as there are two PBS connected by a black wire, with possibly a black negation on this wire, push the possibly remaining negation out using Equation (4), and cancel the PBS together using Equation (10) and its variants. For example:



When there are not two such PBS anymore, all black wires are connected to at least one side of P (possibly through negations), and the PBS are connected together with red and blue wires with possibly negations on them.

4. Remove all isolated loops. Note that since D_0 is query-optimal, there cannot be loops containing gates at this point.

5. Deform P to put it in the form (B) with the C_i of the form  and σ_1 and σ_2 being wire permutations, where  is either or  with $a \in \{\mathbf{v}, \mathbf{h}\}$,  is either  or  and  is either  or .

6. Remove the negations in the middle of the C_i by pushing them to the bottom by means of variants of Equation (4).

7. Transform each C_i, which is now, up to deformation, a ladder of PBS without negations, into one of the five kinds of stairs depicted in Figure 7, depending on its type. To do so, deform it and apply Equations (11) and (12) appropriately, and repeatedly apply the appropriate equation among (14), (15), (16), and a variant of (13). This gives us D_1.

An example of diagram produced by the PGT procedure is given in Figure 8.

Since the PGT procedure consists in putting a subdiagram of D_0 in stair form (except Step 1 which is just deformation and does not change the number of PBS), this procedure does not increase the number of PBS in D_0:

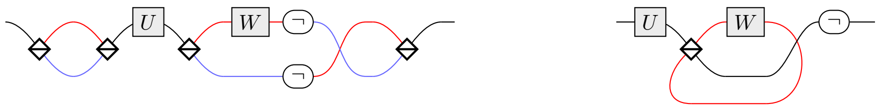


Figure 8 The diagram on the left is the obtained by applying the query-optimisation procedure on the example of Figure 5. The diagram on the right is (up to deformation) obtained by applying the PGT procedure to the diagram on the left. Notice that this diagram is both query- and PBS-optimal.

▶ **Proposition 26.** *The diagram D_1 output by the PGT procedure contains at most as many PBS as the initial diagram D_0.*

This also implies that given any diagram D, there exists an equivalent query-PBS-optimal diagram in PGT form. Indeed, by Proposition 23, there exist query-optimal diagrams equivalent to D, and among these diagrams, some of them have minimal number of PBS and are therefore query-PBS-optimal. Finally, applying the PGT procedure to one of these diagrams gives us an equivalent diagram in PGT form, which, since the PGT procedure does not change the gates or increase the number of PBS, is still query-PBS-optimal.

Applying the PGT procedure after the query optimisation procedure produces an interesting heuristic: the output diagram is necessarily query-optimal and can even be query-PBS-optimal when it does not contain two queries to the same oracle.

Notice that, like the query optimisation procedure, the PGT procedure is efficient, i.e. it can be done using a number of elementary graphical transformations (those of Figure 4) which is linear in the size of the diagram. Moreover, it also requires some diagrammatic transformations, which can be handled using appropriate data structures, leading to a quadratic algorithm.

5.3 Hardness

We show in this section that the query-PBS optimisation problem is actually NP-hard.

▶ **Theorem 27.** *The problem of, given an abstract diagram, finding an equivalent query-PBS-optimal diagram, is NP-hard.*

The proof, given in [13], is based on a reduction from the maximum Eulerian cycle decomposition problem (MAX-ECD) which is known to be NP-hard [8]. The MAX-ECD problem consists, given a graph, in finding a partition of its set of edges into the maximum number of cycles. Intuitively, the reduction goes as follows: given an Eulerian graph $G = (V = \{v_0, \dots v_{n-1}\}, E)$, let σ be a permutation of the vertices of the graph s.t. $\forall i, (v_i, \sigma(v_i)) \in E$ (such a σ exists since G is Eulerian), we construct a V^*-diagram D such that the number of occurrences of each v_i in D is half its degree in G; and such that $\forall i$, $[\![D]\!]\,(\mathbf{V}, i) = ((\mathbf{V}, i), v_i)$ and $[\![D]\!]\,(\mathbf{H}, i) = ((\mathbf{H}, i), \sigma(v_i))$. Roughly speaking, we show that the edge-partitions of G into cycles correspond to the possible implementations of D, and that a partition with a maximal number of cycles leads to an implementation with a minimal number of PBS.

In the following, we explore a few variants of the problem, which remain NP-hard.

First, query-PBS optimisation is still hard when restricted to negation-free diagrams:

▶ **Corollary 28.** *The problem of, given a negation-free abstract diagram, finding an equivalent diagram which is query-PBS-optimal among negation-free diagrams, is NP-hard.*

Additionally, it is also hard, in a query-optimal diagram, to optimise the gates and the negations together by, respectively, defining a cost function (at least in the case where the negation cost is not less than the PBS), prioritising the negations over the PBS, and prioritising the PBS over the negations. Note that the NP-hardness is clear in the third case since the considered problem is a refinement of the query-PBS-optimisation problem addressed in Theorem 27.

▶ **Corollary 29.** *For any $\alpha \geq 1$, the problem of, given an abstract diagram D, finding an equivalent query-optimal diagram D' such that $\#_{\text{PBS}}(D') + \alpha\#_{\neg}(D')$ is minimal, is NP-hard, where $\#_{\neg}(D)$ is the number of negations in D.*

▶ **Corollary 30.** *The problem of, given an abstract diagram D, finding an equivalent query-¬-PBS-optimal*[1] *diagram is NP-hard.*

6 Discussions and Future Work

The power and limits of quantum coherent control is an intriguing question. Maybe surprisingly, we have proved that coherently controlled quantum computations, when expressed in the PBS-calculus, can be efficiently optimised: any PBS-diagram can be transformed in polynomial time into a diagram that is optimal in terms of oracle queries. We have refined the procedure to also decrease the number of polarising beam splitters. It leads to an optimal diagram when each oracle is queried only once, but the corresponding optimisation problem is NP-hard in general. We leave to future work an experimental evaluation of the PGT procedure when each oracle is not necessarily queried only once.

To perform the resource optimisation, we have introduced a few add-ons to the framework of the PBS-calculus. First, we have refined the syntax in order to allow the representation of unsaturated (or 3-leg) polarising beam splitters. They are essential ingredients for resource optimisation, as they provide a way to decompose a diagram into elementary components and then remove the useless ones. However, notice that one can perform resource optimisation of vanilla PBS-diagrams, using the refined one only as an intermediate language. Indeed, given a vanilla PBS-diagram (where all wires are black), one can apply the optimisation procedures described in this paper. The resulting optimised PBS-diagram may contain some unsaturated PBS, but all these 3-leg PBS can be saturated by adding useless traces and then one can make the diagram monochromatic. The resulting vanilla PBS-diagram keeps the same number of queries and PBS.

We have also generalised the gates of the diagrams, by considering arbitrary monoids. This is a natural abstraction that allows one to consider various examples and in particular the one of the free monoid which is appropriate to model the oracle queries. The query complexity is a convenient model to prove lower bounds, but note that the optimisation procedures described in this paper can be applied with any arbitrary monoid (for instance using Proposition 8). However, there is no guarantee of minimality with an arbitrary monoid.

Another direction of research is to consider resource optimisation in a more expressive language for quantum control. Indeed, the polarisation of a particle can only be flipped within a PBS-diagram. The PBS-calculus is well suited for most applications of coherent control in quantum computing, by allowing the description of superpositions of classical controls (in particular superposition of causal orders) since the input particle can be in any superposition of polarisations. However, it would be interesting to develop resource optimisation techniques for quantum computation involving arbitrary quantum control.

[1] A diagram is query-¬-PBS-optimal if it is optimal according to the lexicographic order: the number of queries then the number of negations and finally the number of polarising beam splitters.

References

1 Alastair A. Abbott, Julian Wechs, Dominic Horsman, Mehdi Mhalla, and Cyril Branciard. Communication through coherent control of quantum channels. *Quantum*, 4:333, September 2020. doi:10.22331/q-2020-09-24-333.

2 Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 249–258. IEEE, 2005.

3 Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

4 Mateus Araújo, Fabio Costa, and Časlav Brukner. Computational advantage from quantum-controlled ordering of gates. *Physical Review Letters*, 113(25):250402, 2014. doi:10.1103/PhysRevLett.113.250402.

5 Pablo Arrighi, Christopher Cedzich, Marin Costes, Ulysse Rémond, and Benoît Valiron. Addressable quantum gates. *arXiv preprint*, 2021. arXiv:2109.08050.

6 Costin Badescu and Prakash Panangaden. Quantum alternation: Prospects and problems. In Chris Heunen, Peter Selinger, and Jamie Vicary, editors, *Proceedings 12th International Workshop on Quantum Physics and Logic, QPL 2015, Oxford, UK, July 15-17, 2015*, volume 195 of *EPTCS*, pages 33–42, 2015. doi:10.4204/EPTCS.195.3.

7 Cyril Branciard, Alexandre Clément, Mehdi Mhalla, and Simon Perdrix. Coherent control and distinguishability of quantum channels via PBS-diagrams. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:20, Dagstuhl, Germany, August 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.22.

8 Alberto Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999. doi:10.1137/S089548019731994X.

9 Giulio Chiribella. Perfect discrimination of no-signalling channels via quantum superposition of causal structures. *Physical Review A*, 86(4):040301, 2012. doi:10.1103/PhysRevA.86.040301.

10 Giulio Chiribella, Giacomo Mauro D'Ariano, Paolo Perinotti, and Benoît Valiron. Quantum computations without definite causal structure. *Physical Review A*, 88:022318, August 2013. doi:10.1103/PhysRevA.88.022318.

11 Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. LOv-calculus: A graphical language for linear optical quantum circuits. Accepted at MFCS'22, 2022. arXiv:2204.11787.

12 Alexandre Clément and Simon Perdrix. PBS-calculus: A graphical language for coherent control of quantum computations. In Javier Esparza and Daniel Kráľ, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:14, Dagstuhl, Germany, August 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.24.

13 Alexandre Clément and Simon Perdrix. Minimising resources of coherently controlled quantum computations, 2022. doi:10.48550/ARXIV.2202.05260.

14 Timoteo Colnaghi, Giacomo Mauro D'Ariano, Stefano Facchini, and Paolo Perinotti. Quantum computation with programmable connections between gates. *Physics Letters A*, 376(45):2940–2943, 2012. doi:10.1016/j.physleta.2012.08.028.

15 Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. Realizability in the unitary sphere. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019.

16 Gilles Dowek and Pablo Arrighi. Lineal: A linear-algebraic lambda-calculus. *Logical Methods in Computer Science*, 13, 2017.

17 Daniel Ebler, Sina Salek, and Giulio Chiribella. Enhanced communication with the assistance of indefinite causal order. *Physical Review Letters*, 120(12):120502, March 2018. doi:10.1103/PhysRevLett.120.120502.

18 Stefano Facchini and Simon Perdrix. Quantum circuits for the unitary permutation problem. In *International Conference on Theory and Applications of Models of Computation*, pages 324–331. Springer, 2015. doi:10.1007/978-3-319-17142-5_28.

19 Adrien Feix, Mateus Araújo, and Časlav Brukner. Quantum superposition of the order of parties as a communication resource. *Physical Review A*, 92(5):052326, 2015. doi:10.1103/PhysRevA.92.052326.

20 Philippe Allard Guérin, Adrien Feix, Mateus Araújo, and Časlav Brukner. Exponential communication complexity advantage from quantum superposition of the direction of communication. *Physical Review Letters*, 117(10):100502, 2016. doi:10.1103/PhysRevLett.117.100502.

21 Lucien Hardy. Probability theories with dynamic causal structure: a new framework for quantum gravity. *arXiv preprint gr-qc/0509120*, 2005.

22 Masahito Hasegawa, Martin Hofmann, and Gordon Plotkin. Finite dimensional vector spaces are complete for traced symmetric monoidal categories. In *Pillars of computer science*, pages 367–385. Springer, 2008.

23 Vadym Kliuchnikov and Dmitri Maslov. Optimization of Clifford circuits. *Physical Review A*, 88(5):052307, 2013.

24 Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106, 1965.

25 Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):1–12, 2018.

26 Ognyan Oreshkov, Fabio Costa, and Časlav Brukner. Quantum correlations with no causal order. *Nature communications*, 3(1):1–8, 2012.

27 Martin J. Renner and Časlav Brukner. Reassessing the computational advantage of quantum-controlled ordering of gates. *Physical Review Research*, 3(4):043012, 2021.

28 Peter Selinger. Finite dimensional hilbert spaces are complete for dagger compact closed categories. *Electronic Notes in Theoretical Computer Science*, 270(1):113–119, 2011.

29 Augustin Vanrietvelde, Hlér Kristjánsson, and Jonathan Barrett. Routed quantum circuits. *Quantum*, 5:503, 2021.

30 Julian Wechs, Hippolyte Dourdent, Alastair A. Abbott, and Cyril Branciard. Quantum circuits with classical versus quantum control of causal order. *arXiv preprint*, 2021. arXiv:2101.08796.

31 Matt Wilson and Giulio Chiribella. A diagrammatic approach to information transmission in generalised switches. *arXiv preprint*, 2020. arXiv:2003.08224.

32 Mingsheng Ying, Nengkun Yu, and Yuan Feng. Defining quantum control flow. *arXiv preprint*, 2012. arXiv:1209.4379.

33 Magdalena Zych, Fabio Costa, Igor Pikovski, and Časlav Brukner. Bell's theorem for temporal order. *Nature communications*, 10(1):1–10, 2019.

A Complexity Approach to Tree Algebras: the Polynomial Case

Thomas Colcombet
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Arthur Jaquard
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Abstract

In this paper, we consider infinitely sorted tree algebras recognising regular language of finite trees. We pursue their analysis under the angle of their asymptotic complexity, i.e. the asymptotic size of the sorts as a function of the number of variables involved.

Our main result establishes an equivalence between the languages recognised by algebras of polynomial complexity and the languages that can be described by nominal word automata that parse linearisation of the trees. On the way, we show that for such algebras, having polynomial complexity corresponds to having uniformly boundedly many orbits under permutation of the variables, or having a notion of bounded support (in a sense similar to the one in nominal sets).

We also show that being recognisable by an algebra of polynomial complexity is a decidable property for a regular language of trees.

2012 ACM Subject Classification Theory of computation → Tree languages; Theory of computation → Regular languages

Keywords and phrases Tree algebra, nominal automata, language theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.37

Funding *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).

1 Introduction

Among the many different approaches to language theory, the algebraic one focalises toward the understanding of the expressive power of regular languages based on the properties of algebraic recognizers. The first work in this direction [11] characterized star-free languages, and initiated a very fruitful branch of research. While algebraic theories for word languages (both finite and inifinite) are already well developed, the corresponding picture remains incomplete for e.g. languages of trees (both finite or infinite) or graphs. Finding an effective characterization of the regular languages of trees definable in first order logic remains for instance a long standing open problem.

When designing algebras for tree languages or graph languages, one is naturally inclined to consider infinitely sorted algebras. The case of tree algebras (such as preclones, ω-hyperclones, operads [8, 1]) is typical: plugging a subtree into another one requires a mechanism for identifying the leaf/leaves in which the substitution has to be performed. Notions such as variables, hole types, or colors are used for that. Another example is the one of graphs (HR- and VR- algebras [7]) in which basic operations (a) glue graphs together using a set of colors (sometimes called ports) for identifying the glue-points, or (b) add all possible edges between vertices of fixed given colors. In these examples, the algebras are naturally sliced into infinitely many sorts based on the number of variables/hole types/colors that are used simultaneously.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 37; pp. 37:1–37:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, a technical difficulty arises immediately when using such algebras. Even when all sorts are finite (what we call a finite algebra), these algebras are not really finite due to the infinite number of sorts. This forbids, for instance, to explicitely describe the whole algebra in a finite way, which is of course a problem for designing algorithms. This hurdle to handle infinitely sorted algebras can, arguably, be seen as one of the causes of the many years that it took before having a good definition of an algebra for infinite trees [2], or the time that it took before it was possible to characterize logically the expressiveness of recognizable properties of graphs under bounded tree-width hypothesis [4].

Complexity of algebras. To cope with this difficulty, the notion of complexity for infinitely sorted algebras was introduced in [5]. In each of the above cases, the sorts are naturally indexed by a natural number parameter: the number of variables, or hole types, or colors. Hence an algebra $\mathcal{A}$ would have a carrier of the form

$$(A_n)_{n\in\mathbb{N}}$$

together with suitable operations that depend on the particular algebra type. Such an algebra is called finite if all the A_n are finite and, in this case, the complexity map $c_{\mathcal{A}}\colon \mathbb{N} \to \mathbb{N}$ of the algebra is defined as:

$$c_{\mathcal{A}}(n) = |A_n| \ , \qquad \text{for all } n \in \mathbb{N}.$$

This approach gives rises to the classification of finite algebras depending on the asymptotic growth of $c_{\mathcal{A}}$: algebras can have bounded complexity, polynomial complexity, etc. It is then possible to study the expressive power of algebras in a prescribed complexity class.

In all of the mentioned examples of algebras, there is a natural operation that performs a renaming of the variables/hole types/colors. This renaming is parameterized by a bijection over variables/hole types/colors, and this permutation acts on the corresponding sort. Thus, there is an action of the symmetric group over n elements, $\mathbf{Sym}(n)$ over A_n. The orbit-complexity map $c^\circ\colon \mathbb{N} \to \mathbb{N}$ is then naturally defined as:

$$c^\circ_{\mathcal{A}}(n) = |A_n/\mathbf{Sym}(n)| \ , \qquad \text{for all } n \in \mathbb{N}.$$

The notions of bounded orbit-complexity, polynomial orbit-complexity, etc. are then defined accordingly.

Along with the definitions of complexity and orbit-complexity, a precise description of the languages recognized by tree algebras of bounded complexity was given in [5][1]. In this article we endeavor to study tree algebras of polynomial complexity.

Tree algebras. The notion of tree algebra that we presented above is a bit unpractical, because the variables/hole types/colors are unnamed (we will simply call them variables from now on). We instead consider tree algebras with a carrier of the form

$$(A_X)_{X \text{ finite set of variables}}$$

for which the notions of complexity and orbit-complexity can be easily adapted. Given a finite X, a tree is then seen as an element of A_X whenever all the variables on the leaves of the tree are in X. This notion of tree algebras may have different flavors:

[1] Let us emphasize that the algebras used in [5] are different, since all variables are furthermore required to appear at least once. This apparently inocuous modification has consequences on the results.

- *Unrestrained tree algebras* in which there is no additional constraint, as in this paper.
- *Affine tree algebras*: all variables must appear on at most one leaf of the tree [2].
- *Relevant tree algebras*: all variables must appear on at least one leaf of the tree [5].
- *Linear tree algebras*: all variables must appear on exactly one leaf of the tree [8, 9].

We could also consider variants in which the variables are ordered. For instance, starting from linear tree algebras and adding the condition that, when read from left to right, the variables are monotone, corresponds to preclones [8]. The expressive power of relevant tree algebras of bounded complexity was precisely described in [5].

Contributions of the article. In this article, we study unrestrained tree algebras (or simply tree algebras from now on) of polynomial complexity.

We introduce a way to encode finite trees into data words, and thus to encode languages of trees into languages of data words (we say that the language of trees is described by the language of data words).

We then establish that tree algebras of polynomial complexity and of bounded orbit-complexity have the same expressive power, thus answering a question from [5] (for unrestrained tree algebras). Moreover, the languages that they recognize are exactly those described by regular languages over our data alphabet (in which the notion of regular language is defined using orbit-finite nominal automata [10, 3], a mild generalization of register automata that fit nicely into our algebraic setting).

Our main result is the following Theorem 1. It states the above described equivalences, and add a third item that will be formalised in the body of the paper.

▶ **Theorem 1.** *For a regular language of finite trees, the following properties are equivalent:*

1. *Being recognized by a finite tree algebra of polynomial complexity.*
2. *Being recognized by a finite tree algebra of bounded orbit complexity.*
3. *Being recognized by a finite tree algebra that has a bounded and stable system of supports.*
4. *Being described by a coding automaton.*

Our second theorem, Theorem 22, establishes the decidability of this class.

Structure of the paper. In Section 2, we recall some classical definitions, and introduce the notion of algebras. In Section 3, we explain our encoding of trees into data words, and prove Proposition 16 corresponding to the implication from Item 4 to Items 1 and 2 of Theorem 1. In Section 4, we look in detail at the properties of tree algebras of polynomial complexity and bounded orbit-complexity. Doing this, we prove Propositions 19 and 20, that correspond to proving implications from Items 1 and 2 to Item 3, and from Item 3 to Item 4 of Theorem 1. We also address the decidability question an prove Theorem 22. Section 5 concludes.

2 Definitions

We denote by $\mathbb{N}$ the set of all non-negative integers. Given $n \in \mathbb{N}$, we write $[n] = \{0, 1, ..., n-1\}$. The symmetric group (resp. alternating group) of a set X is denoted $\mathbf{Sym}(X)$ (resp. $\mathbf{Alt}(X)$). We fix a finite *ranked alphabet* Σ; the *arity* of a *symbol* $a \in \Sigma$ is denoted $\mathrm{ar}(a)$. It is a *constant* if $\mathrm{ar}(a) = 0$, and is *unary* if $\mathrm{ar}(a) = 1$. For $k \in \mathbb{N}$, we set $\Sigma_k = \{a \in \Sigma \mid \mathrm{ar}(a) = k\}$. A^* is the set of finite words over A, and $A^+ = A^* \setminus \{\varepsilon\}$.

2.1 Trees

In this section, we introduce notions and notations for trees.

We fix a countable set of *variables* $\mathcal{V}$. Given a finite set of variables X, a Σ, X-tree is, informally, a tree in which nodes are labelled by elements of Σ and leaves also possibly by variables of X. Formally, a *Σ, X-tree* is a partial map $t \colon \mathbb{N}^* \to \Sigma \uplus X$ such that $\mathrm{dom}(t)$ is non-empty and prefix-closed, and furthermore, for every $u \in \mathrm{dom}(t)$ there exists $n \in \mathbb{N}$ such that $\{i \mid ui \in \mathrm{dom}(t)\} = [n]$, and

- either $t(u) \in \Sigma_n$ (*symbol node*), or
- $t(u) \in X$ and $n = 0$ (*variable node*). Note that a variable node is always a leaf.

$\Sigma, \emptyset$-trees are simply called *Σ-trees.* The elements in $\mathrm{dom}(t)$ are called *nodes.* The prefix relation over nodes is called the *ancestor relation.* The node ε is called the *root* of the tree. The tree t is *finite* if it has finitely many nodes. A *branch* of a tree t is a maximal set of nodes ordered under the ancestor relation. Let $\mathrm{Trees}(\Sigma, X)$ bet the set of finite Σ, X-trees, for every finite set of variables X.

▶ Remark 2. Note that Σ, X-trees are also Σ, Y-trees whenever $X \subseteq Y$. This is in contrast with [5] in which all variables were assumed to appear at least once.

Building trees. We introduce now some operations on trees. See Fig. 1.

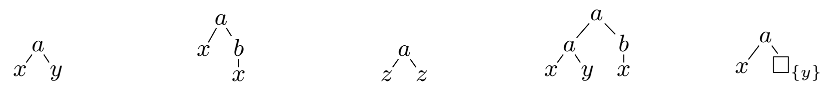
- ε_x, where x is a variable, denotes the $\Sigma, \{x\}$-tree consisting of a single root node labelled x.
- $a(x_0, \dots, x_{n-1})$, for $x_0, \dots, x_{n-1}$ variables and $a \in \Sigma_n$, denotes the $\Sigma, \{x_0, \dots, x_{n-1}\}$-tree consisting of a root labelled a, and children $0, \dots, n-1$ labelled with variables $x_0, \dots, x_{n-1}$ respectively.
- $s \cdot_x t$, for two trees $s \in \mathrm{Trees}(\Sigma, X)$, $t \in \mathrm{Trees}(\Sigma, Y)$ and a variable $x \in \mathcal{V}$, is the $\Sigma, (X \setminus \{x\}) \cup Y$-tree s in which t is substituted for every occurrence of the variable x, which may not be present in s at all.
- $\widetilde{\sigma}(t)$, for a tree $t \in \mathrm{Trees}(\Sigma, X)$ and a map $\sigma \colon X \to Y$, is the Σ, Y-tree obtained as t in which variable $\sigma(x)$ has been substituted to x for every $x \in X$. Note that $\widetilde{\sigma} \circ \widetilde{\tau} = \widetilde{\sigma \circ \tau}$.
- $t[x_0 \leftarrow t_0, \dots, x_{n-1} \leftarrow t_{n-1}]$ denotes the tree of sort $X \setminus \{x_0, \dots, x_{n-1}\} \cup \bigcup_i Y_i$ obtained from t by simultaneously substituting the tree t_i for the variable x_i for every $i \in [n]$, where t is a tree of sort X, $x_0, \dots, x_{n-1} \in \mathcal{V}$, and $t_0, \dots, t_{n-1}$ are trees of sort Y_i for every $i \in [n]$. Note that this operation is equivalent to a combination of the previous ones.
- $a(t_0, ..., t_{n-1})$, for $a \in \Sigma_n$, denotes the tree of root a and children $t_0, \dots, t_{n-1}$ at respective positions $0, \dots, n-1$. Again, this operation is equivalent to a combination of the previous ones.

▶ **Example 3.** Throughout this paper, we use the map $\mathrm{create}_x^X : X \to X \cup \{x\}$, acting as the identity, in which X is a finite set of variables and $x \in \mathcal{V}$. $\widetilde{\mathrm{create}_x^X}$ is thus a mapping from $\mathrm{Trees}(\Sigma, X)$ to $\mathrm{Trees}(\Sigma, X \cup \{x\})$ that maps every tree to itself. Most of the time, we will simply write $\mathrm{create}_x(t)$ when X is clear from the context.

▶ **Lemma 4.** *All finite trees can be obtained from the trees of the form ε_x and $a(x_0, \dots, x_{n-1})$ using the operations "$\cdot$".*

Expressions denoting finite trees. For X a finite set of variables, a *tree-expression of sort X (over the alphabet Σ)* is an expression built inductively as follows:

- ε_x is a tree-expression of sort $\{x\}$ for every variable x,
- $a(x_0, \dots, x_{n-1})$ is a tree-expression of sort $\{x_0, \dots, x_{n-1}\}$ for every symbol $a \in \Sigma_n$,



$t = a(x, y)$ $t \cdot_y b(x)$ $\tilde{\sigma}(t)$ $t[x \leftarrow t, y \leftarrow b(x)]$ $a(x, \square_{\{y\}})$

Figure 1 Trees and contexts with their notations. Here $\sigma(x) = \sigma(y) = z$.

- $S \cdot_x T$ is a tree-expression of sort $X \setminus \{x\} \cup Y$ for all tree-expressions S of sort X, all tree-expressions T of sort Y, and all variables $x \in \mathcal{V}$ (*substitution*),
- $\tilde{\sigma}(T)$ is a tree-expression of sort Y for all tree-expressions T of sort X, and map $\sigma \colon X \to Y$ (*renaming*). Note that σ needs not be bijective here.

For a tree-expression T of sort X, $[\![T]\!]$ denotes its evaluation into a finite Σ, X-tree using the operations of substitution and renaming.

Contexts. We define now contexts, which are terms with a specific leaf called the hole. Since we work in a multi-sorted algebra, the hole itself has a sort. Essentially, to a hole of sort X will be substituted a term of sort X. Formally, for fixed finite set of variables Y, a *context of sort X with hole of sort Y* (or simply a context) is defined inductively as a tree expression of sort X, using the extra construction $\square_Y$ (the *hole of sort* Y) which is a context of sort Y with hole of sort Y. This new construction must appear exactly once in a context.

For C a context of sort X with hole of sort Y, $[\![C]\!] \colon \mathrm{Trees}(\Sigma, Y) \to \mathrm{Trees}(\Sigma, X)$ is the function which to a tree of sort Y t associates the tree of sort X obtained by evaluating the operations as above, interpreting $\square_Y$ as t.

2.2 Finite tree algebras

Our notion of tree algebra is the natural notion associated to finite trees equipped with the above operations. We give here a more formal definition, though the detail of identities is more for reference. What matters is that it is defined such that the free algebra coincides with finite trees. A *tree algebra* $\mathcal{A}$ consists of an infinite collection of carrier sets A_X indexed by finite sets of variables X, together with operations:

- $\varepsilon_x^{\mathcal{A}} \in A_{\{x\}}$ for every variable x,
- $a(x_0, \ldots, x_{n-1})^{\mathcal{A}} \in A_{\{x_0, \ldots, x_{n-1}\}}$ for all $a \in \Sigma_n$ and variables $x_0, \ldots, x_{n-1}$,
- $\cdot_x^{\mathcal{A}} \colon A_X \times A_Y \to A_{X \setminus \{x\} \cup Y}$ for all finite sets of variables X, Y and $x \in \mathcal{V}$,
- $\sigma^{\mathcal{A}} \colon A_X \to A_Y$ for every renaming $\sigma \colon X \to Y$,

that satisfy the expected identities, i.e. the ones guaranteeing that several ways to describe the same tree yield the same evaluation in the algebra. Formally, for all s, t, u that belong to A_X, A_Y, A_Z respectively,

- $\varepsilon_x \cdot_x t = t$ for every $x \in \mathcal{V}$,
- $t \cdot_x \varepsilon_x = t$ for every $x \in Y$,
- $(s \cdot_x^{\mathcal{A}} t) \cdot_y^{\mathcal{A}} u = s \cdot_x^{\mathcal{A}} (t \cdot_y^{\mathcal{A}} u)$ for all $x \in X$ and $y \in X \setminus Y$ (horizontal associativity), and
- $(s \cdot_x^{\mathcal{A}} t) \cdot_y^{\mathcal{A}} u = s \cdot_x^{\mathcal{A}} (t \cdot_y^{\mathcal{A}} u)$ for all $x \in X$ and $y \in Y \setminus X \cup \{x\}$ (vertical associativity),

for all s, t that belong to A_X, A_Y, $x \in X$ and renaming $\sigma \colon X \to Y$,

- $\sigma^{\mathcal{A}}(s \cdot_x^{\mathcal{A}} t) = \sigma^{\mathcal{A}}(s) \cdot_x^{\mathcal{A}} t$ if $\sigma^{-1}(\sigma(x)) = \{x\}$ and $\sigma(y) = y$ for every $y \in X \cap Y \setminus \{x\}$,
- $\sigma^{\mathcal{A}}(s \cdot_x^{\mathcal{A}} t) = s \cdot_x \sigma^{\mathcal{A}}(t)$ if $\sigma(y) = y$ for every $y \in X \cap Y \setminus \{x\}$,

for all maps $\sigma \colon X \to Y$ and $\tau \colon Y \to Z$, $(\tau \circ \sigma)^{\mathcal{A}} = \tau^{\mathcal{A}} \circ \sigma^{\mathcal{A}}$, and for all maps $\sigma \colon \{x_0, \ldots x_{n-1}\} \to Y$ and $a \in \Sigma_n$, $\sigma^{\mathcal{A}}(a(x_0, \ldots, x_{n-1})^{\mathcal{A}}) = a(\sigma(x_0), \ldots, \sigma(x_{n-1}))^{\mathcal{A}}$. In practice, we shall not explicitly use these identities, and simply write two elements of the algebra equal as soon as they obviously come from expressions denoting the same trees. A tree algebra is *finite* if the A_X's are all finite.

A *morphism of tree algebras* from $\mathcal{A}$ to $\mathcal{B}$ is a family of maps $\alpha_X\colon A_X \to B_X$ for every finite sets of variables X which preserves all operations, i.e. $\alpha_Y(\sigma^{\mathcal{A}}(s)) = \sigma^{\mathcal{B}}(\alpha_X(s))$ for every map $\sigma\colon X \to Y$, $\alpha(a(x_0,\dots,x_{n-1})^{\mathcal{A}}) = a(x_0,\dots,x_{n-1})^{\mathcal{B}}$, and $\alpha_{X\setminus\{x\}\cup Y}(s \cdot_x^{\mathcal{A}} t) = \alpha_X(s) \cdot_x^{\mathcal{B}} \alpha_Y(t)$ for all $s \in A_X$, $t \in A_Y$ and $x \in \mathcal{V}$.

The $\mathrm{Trees}(\Sigma, X)$ sets equipped with the operations of substitution and renaming form a tree algebra (it is the free tree algebra generated by $\emptyset$). For $\mathcal{A}$ a tree algebra, its associated *evaluation morphism* is the unique morphism from $\mathrm{Trees}(\Sigma)$ to $\mathcal{A}$.

A *congruence* $\sim$ over a tree algebra $\mathcal{A}$ is a family $\sim$ of equivalence relations over the A_X's (each denoted $\sim$) such that, for any $a \sim b \in A_X$, $c \sim d \in A_Y$, $y \in Y$ and $\sigma\colon X \to Y$: $c \cdot_y a \sim c \cdot_y b$; $c \cdot_y a \sim d \cdot_y a$ and $\widetilde{\sigma}(a) \sim \widetilde{\sigma}(b)$. From such a congruence, one can define the *quotient algebra* $\mathcal{A}/\sim$ in the natural way.

2.3 Languages and syntactic algebras

A *language of finite* Σ*-trees* L is a set of Σ-trees. It is *recognized by* a tree algebra $\mathcal{A}$ if there is a set $P \subseteq A_\emptyset$ such that $L = \alpha^{-1}(P)$ in which α is the evaluation morphism of $\mathcal{A}$.

The *syntactic congruence* $\sim_L$ *of a language* L of finite Σ-trees is defined in the following way $s \sim_L t$ for s, t finite Σ-trees if, for every context C, $[\![C]\!](s) \in L$ if and only if $[\![C]\!](t) \in L$. It is easy to prove that $\sim_L$ is indeed a congruence. The quotient algebra $\mathrm{Trees}(\Sigma)/_{\sim_L}$ is called the *syntactic algebra* of L, and this algebra recognizes L.

▶ **Example 5.** The language of all finite trees in which the symbol a appears on the leftmost branch has for syntactic tree algebra the algebra with sorts $A_X = \{\bot, \top\} \uplus X$ for every finite set of variables X. Let t be a Σ, X-tree, we define α as follows: $\alpha_X(t) = \top$ if there is an a on the leftmost branch of t, $\alpha_X(t) = x$ if the leftmost branch of t ends with an x but contains no a, and $\alpha_X(t) = \bot$ otherwise. The operations of the algebra are defined so that α becomes the evaluation morphism.

2.4 Complexity

Following [5], we define the notion of complexity and of orbit-complexity of a tree algebra $\mathcal{A}$.

Complexity. We start by highlighting the fact that any bijection σ between finite sets of variables X and Y induces a bijection $\widetilde{\sigma}$ between A_X and A_Y. As such, it is meaningful to define the *complexity map* of the algebra $c_{\mathcal{A}}\colon \mathbb{N} \to \mathbb{N}$ as follows:

$$c_{\mathcal{A}}(|X|) = |A_X| \ , \qquad \text{for every finite set of variables } X.$$

A tree algebra $\mathcal{A}$ has *bounded complexity* if $c_{\mathcal{A}}$ is bounded. It has *polynomial complexity* if there is a polynomial P such that $c_{\mathcal{A}}(n) \leqslant P(n)$ for every $n \in \mathbb{N}$.

Orbit-complexity. Similarly, we define the *orbit-complexity map* $c^{\circ}_{\mathcal{A}}\colon \mathbb{N} \to \mathbb{N}$ as:

$$c^{\circ}_{\mathcal{A}}(|X|) = |A_X/\mathbf{Sym}(X)| \ , \qquad \text{for every finite set of variables } X,$$

in which $A_X/\mathbf{Sym}(X)$ is the set of all orbits of A_X under the action of $\mathbf{Sym}(X)$. A tree algebra $\mathcal{A}$ has *bounded orbit-complexity* if $c^{\circ}_{\mathcal{A}}$ is bounded.

▶ **Example 6.** The tree algebra $\mathcal{A}$ from Example 5 has polynomial complexity and bounded orbit-complexity: $c_{\mathcal{A}}(n) = n + 2$ and $c^{\circ}_{\mathcal{A}}(n) = 3$ for every $n \in \mathbb{N}$.

To check whether a language of trees is recognized by an algebra with a prescribed complexity, one only needs to look at its syntactic algebra.

▶ **Lemma 7.** *If L is a language of Σ-trees recognized by a tree algebra $\mathcal{B}$, then its syntactic tree algebra $\mathcal{A}$ has lower complexity:*

$$c_{\mathcal{A}}(n) \leqslant c_{\mathcal{B}}(n) \quad \textit{and} \quad c^{\circ}_{\mathcal{A}}(n) \leqslant c^{\circ}_{\mathcal{B}}(n)\ ,\ \textit{for all } n \in \mathbb{N}.$$

2.5 Tree automata

A *tree automaton* $\mathcal{B} = (Q, I, (\delta_a)_{a\in\Sigma})$ over Σ has a finite set Q of *states*, a set of *accepting states* $I \subseteq Q$ and a *transition relation* $\delta_a \subseteq Q \times Q^{\mathrm{ar}(a)}$ for every symbol $a \in \Sigma$. A *run of* $\mathcal{B}$ over a finite tree t is a mapping $\rho\colon \mathrm{dom}(t) \to Q$ such that, for any vertex $u \in \mathrm{dom}(t)$ with $t(u) = a \in \Sigma$, $(\rho(u), (\rho(u0), ..., \rho(u(\mathrm{ar}(a) - 1)))) \in \delta_a$. A run is *accepting* if $\rho(\varepsilon) \in I$. A language L of finite trees is called *regular* if it is recognized by a tree automaton $\mathcal{B}$, meaning the trees in L are exactly those for which there is an accepting run in $\mathcal{B}$.

Example 8 below shows the translation from tree automata to tree algebra.

▶ **Example 8** (Automaton algebra)**.** Consider a regular language L of finite trees recognized by the tree automaton $\mathcal{B} = (Q, q_0, (\delta_a)_{a\in\Sigma})$. Consider some finite set of variables X. An *X-run profile* is a tuple $\tau \in Q \times \mathcal{P}(Q)^X$. For a Σ, X-tree t, $\tau = (p, (U_x)_{x\in X})$ is a *run profile over* t if there exists a run ρ of the automaton over Q such that $\rho(\varepsilon) = p$ and for every variables $x \in X$, U_x is the set of states assumed by ρ at leaves labelled x. We define a tree algebra $\mathcal{A}$ that has as elements of sort X sets of X-run profiles. The definition of the operations is natural, and is such that the image of a Σ, X-tree t under the evaluation morphism yields the set of run profiles over t. It naturally recognizes the language L.

Note that this definition yields an algebra of doubly exponential complexity (and hence, this is an upper bound for regular languages). Of course, in practice, one can restrict the algebra to the reachable elements, and this may dramatically reduce the complexity.

The converse translation is also true (it is for instance proved for preclones in [8]), yielding the following result.

▶ **Proposition 9.** *A finite tree language is regular if and only if it is recognized by a finite algebra. Moreover, every regular tree language is recognized by an algebra of doubly exponential complexity.*

2.6 Group actions and orbit-finite sets

To conclude this list of definitions, we recall some notions on group actions.

Group actions. A (left) group action of a group G on a set X is given by a function $\cdot\colon G \times X \to X$ such that $e \cdot x = x$ and $\sigma \cdot (\tau x) = (\sigma\tau) \cdot x$ for all $x \in X$ and $\sigma, \tau \in G$, where e is the neutral element of G. A set X equipped with such an action is called a *G-set.*

Orbits. For every x in a G-set X, the set $G \cdot x = \{\sigma \cdot x \mid \sigma \in G\}$ is called the *orbit* of x. A G-set is partitioned by the orbits of its elements, and it is said to be *orbit-finite* whenever it has only a finite number of different orbits.

Equivariant subsets and relations. Given a G-set X, a subset Y of X is *equivariant* if $\sigma \cdot Y = Y$ for every $\sigma \in G$. Accordingly, a relation $R \subseteq X \times Y$ between G-sets X and Y is equivariant if it is an equivariant subset of the G-set $X \times Y$ equipped with the point-wise action of G. In particular, a function $f\colon X \to Y$ is equivariant exactly when $f(\sigma \cdot x) = \sigma \cdot f(x)$ for all $x \in X$ and $\sigma \in G$.

Support and nominal sets. From now on, we will only be looking at the group $G = \mathbf{Sym}(\mathcal{V})$. Consider then a $\mathbf{Sym}(\mathcal{V})$-set X. A set $S \subseteq \mathcal{V}$ *supports* an element $x \in X$ if $\sigma \cdot x = x$ for every $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus S)$, where $\mathbf{Sym}(\mathcal{V} \setminus S)$ is seen as a subgroup of $\mathbf{Sym}(V)$. A $\mathbf{Sym}(V)$-set is called a *nominal* set if all of its elements are supported by a finite set.

Given an element $x \in X$ that is finitely supported, it admits a least support. The *least support* of an element x from a nominal set X will be denoted $\mathrm{supp}(x)$.

Whenever A and B are $\mathbf{Sym}(\mathcal{V})$-sets, the set of all functions from A to B may be equipped with the action that maps $f\colon A \to B$ to $\sigma \cdot f$ defined for every $a \in A$ by $(\sigma \cdot f)(a) = \sigma \cdot f(\sigma^{-1} \cdot a)$. When seen in this way as a $\mathbf{Sym}(\mathcal{V})$-set, a function $f\colon A \to B$ is supported by $X \subseteq \mathcal{V}$ whenever $f(\sigma \cdot x) = \sigma \cdot f(x)$ for every $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus X)$.

3 Nominal word automata for tree languages

We start our presentation by showing how nominal word automata can be used to describe regular languages of finite trees. To this end, we first explain in Section 3.1, how to encode trees into data words. In Section 3.2, we exploit this to interpret data languages as tree languages, using the notion of coding automata. In Section 3.3 we establish Proposition 16 stating that languages of trees that are described by coding automata are recognized by tree algebras of polynomial complexity and bounded orbit complexity (thus proving the implication from Item 4 to Items 1 and 2 of Theorem 1).

3.1 Coding languages

In this section, we show how to encode finite trees into data words.

Coding alphabet. We begin by defining the nominal alphabet used for this encoding:

- Let $C_{\mathcal{V}}$ be the set of elements $[x]$ for $x \in \mathcal{V}$.
- Let $C_{\mathcal{V},\Sigma}$ be the set of elements $[\cdot_x a(x_0, ..., x_{n-1})]$ for $a \in \Sigma_n$ and $x, x_0, ..., x_{n-1} \in \mathcal{V}$.
- Let $C = C_{\mathcal{V}} \uplus C_{\mathcal{V},\Sigma}$. It is called the *coding alphabet.*

The coding alphabet is naturally made into a $\mathbf{Sym}(V)$-set, by defining, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma[x] = [\sigma(x)]$ and $\sigma[\cdot_x a(x_0, ..., x_{n-1})] = [\cdot_{\sigma(x)} a(\sigma(x_0), ..., \sigma(x_{n-1}))]$ for all $[x]$ and $[\cdot_x a(x_0, ..., x_{n-1})]$ in the coding alphabet. It is obviously both orbit-finite and nominal.

Tree coding. A *coding* is a word in $\mathrm{Codings} = C_{\mathcal{V}} C_{\mathcal{V},\Sigma}{}^*$. We shall describe now how codings can be evaluated to trees. This is natural since, forgetting the bracket notation, codings can be seen as tree expressions. The *evaluation* $T(c)$ of a coding c is defined as follows: $T([x]) = x$, where $[x] \in C_{\mathcal{V}}$ and $T(c[\cdot_x a(x_0, ..., x_{n-1})]) = \mathrm{create}_x(T(c)) \cdot_x a(x_0, ..., x_{n-1})$, where c is a tree coding and $[\cdot_x a(x_0, ..., x_{n-1})] \in C_{\mathcal{V},\Sigma}$.

▶ **Example 10.** According to the definition of evaluation, $T([x][\cdot_x a(x, y)]) = x \cdot_x a(x, y) = a(x, y)$. Similarly, $T([x][\cdot_x a(x, y)][\cdot_x c][\cdot_x d]) = ((x \cdot_x a(x, y)) \cdot_x c) \cdot_y d = a(c, d)$.

We will be particularly interested in codings c that evaluate to trees without variables (meaning $T(c) \in \text{Trees}(\Sigma, \emptyset)$). Let $\text{Codings}_\emptyset$ be the set of these codings that evaluate to variable-less trees.

Describing languages of trees. A language of codings K is said to *describe* a language L of trees without free variables if, for every coding $c \in \text{Codings}_\emptyset$, $c \in K$ if and only if $T(c) \in L$. The crucial point in this definition is that the language K may also contain codings that do not evaluate to a tree without free variables.

▶ **Example 11.** Let L be a language of trees without free variables, and let K be the language of all codings that evaluate to trees in L. Then both K and $K' = K \cup \{[x][\cdot_x a(x, y)]\}$ describe L. That is because $[x][\cdot_x a(x, y)] \notin \text{Codings}_\emptyset$.

However, not all languages of codings describe tree languages.

▶ **Example 12.** Let $\Sigma = \Sigma_0 \cup \Sigma_2$ with $\Sigma_0 = \{c\}$ and $\Sigma_2 = \{a, b\}$, and consider the language K of codings in which the third letter is $[\cdot_x c]$ for any choice of $x \in \mathcal{V}$.

Let $c = [x][\cdot_x a(x, y)][\cdot_y c][\cdot_x a(y, y)][\cdot_y c]$ and $c' = [x][\cdot_x a(x, y)][\cdot_x a(y, y)][\cdot_y c]$, then c is in K but not c'. Note however that $T(c) = T(c') = a(a(c, c), c)$, and thus K does not describe a language of trees.

3.2 Coding automata

Let us now look at acceptance of coding languages by automata. We start by recalling the notion of nominal automata [10], which is used to recognize data languages over orbit-finite nominal alphabets.

Nominal automaton. A *deterministic G-automaton* is given by

- an orbit-finite G-set A (the alphabet),
- a G-set Q (the states),
- an empty supported $q_0 \in Q$ called the *initial state*,
- an equivariant subset $F \subseteq Q$ of *final states*,
- and an equivariant function $\delta \colon Q \times A \to Q$ called the *transition function*.

Acceptance of a word $w \in A^*$ is then defined in the standard way. A deterministic G-automaton is called *orbit-finite* whenever Q is. It is called *nominal* when $G = \mathbf{Sym}(V)$ and when A and Q are both nominal sets. In this paper, we only consider *orbit-finite nominal automata.*

Coding automata. An orbit-finite nominal automaton $\mathcal{A}$ over the coding alphabet is called a *coding automaton* if it recognizes a language that describes a tree language. We also assume that there is no transition toward the initial state. The *tree language described* by a coding automaton is the language L of trees without free variables described by the language K recognized by $\mathcal{A}$. In other words, it is the language

$$L = \{T(c) \mid c \in \text{Codings}_\emptyset,\ \mathcal{A} \text{ accepts } c\} .$$

▶ **Example 13.** Let $\Sigma = \Sigma_0 \cup \Sigma_2$ with $\Sigma_0 = \{c\}$ and $\Sigma_2 = \{a, b\}$, and consider the language L of trees such that the total number of nodes labelled a appearing on the leftmost branch and the rightmost branch is even. We give a coding automaton that describes L. Its set of states is $Q = \{q_0\} \uplus \{\varepsilon(x) \mid x \in \mathcal{V}\} \uplus \{i(x, y) \mid i \in [2], x, y \in \mathcal{V} \cup \{*\}\}$, and the action of $\mathbf{Sym}(V)$ on Q is the one naturally obtained by permuting the variables. Its transitions are defined in the natural way so that:

- q_0 is the initial state,
- $\varepsilon(x)$ is reached from q_0 by reading $[x]$,
- $\delta(q_0, c) = i(x, y)$, in which i is the total number of nodes labelled a on the leftmost and rightmost branches of t modulo 2, and x (resp. y) is the variable on the leaf of the leftmost (resp. rightmost) branch ($*$ if there is no such variable), for $t = T(c)$.

Setting the only accepting state to be $0(*, *)$, this automaton describes L.

A coding automaton is thus a device that takes as input a top-down description of a tree (a coding), and that decides its belonging to a language while remembering only boundedly many variables. Intuitively, such a device can only remember what happens along boundedly many branches. The subtlety of the model is that it must always yield the same result for a given tree, nonwithstanding the actual coding that was provided.

Minimization. Coding automata can be minimized, in the sense that a coding automaton can be effectively turned into another one that describes the same tree language and is minimal. This property turns out to be key to prove Theorem 1. The construction, though similar to the classical one for minimizing nominal automata recognizing a word language, is not the same. One subtlety is that in our case, there are states in the automaton that may accept both codings in $\mathrm{Codings}_\emptyset$ and codings outside of $\mathrm{Codings}_\emptyset$. Standard constructions are not able to cope with such a phenomenon.

We start by defining the *Myhill-Nerode relation* (or *congruence*) of a tree language L over codings by $c \equiv_L c'$, for codings c, c', when

$$T(cv) \in L \Leftrightarrow T(c'v) \in L \quad \text{for all } v \in {C_{\mathcal{V},\Sigma}}^* \text{ such that } cv \in \mathrm{Codings}_\emptyset \text{ and } c'v \in \mathrm{Codings}_\emptyset \ .$$

Note here that the trees coded by c and c' may have a different set of free variables. This is the only subtlety in the proof of the following expected statement.

▶ **Lemma 14.** *The Myhill-Nerode relation of a tree language L is an equivariant congruence in the sense that it is an equivalence and $cv \equiv_L c'v$ for all codings c, c' such that $c \equiv_L c'$, and every $v \in {C_{\mathcal{V},\Sigma}}^*$.*

It is now standard (see e.g. [10]) to define the *minimal automaton* $\mathrm{Min}_L = (Q, q_0, F, \delta)$ of a tree language L as follows:

- $Q = \{q_0\} \uplus \mathrm{Codings}/_{\equiv_L}$, in which q_0 is the (fresh) initial state,
- $F = \{[c]_{\equiv_L} \mid [c]_{\equiv_L} \subseteq L\}$,
- the transition function δ is given by $\delta(q_0, [x]) = [\,[x]\,]_{\equiv_L}$ and $\delta([c]_{\equiv_L}, v) = [cv]_{\equiv_L}$,

where $[c]_{\equiv_L}$ is the $\equiv_L$-class of $c \in \mathrm{Codings}$.

▶ **Lemma 15.** *For L a a tree language described by a coding automaton,* Min_L *is effectively a coding automaton which describes L.*

3.3 From coding automata to tree algebras

Our next result is Proposition 16 below, which corresponds to the implication from Item 4 to Items 1 and 2 of Theorem 1.

▶ **Proposition 16.** *Every tree language L described by a coding automaton is also recognized by a tree algebra that has both polynomial complexity and bounded orbit-complexity.*

This is proved by transforming the minimal automaton $\mathrm{Min}_L = (Q, q_0, F, \delta)$ that describes a language L, into a tree algebra $\mathcal{A}$ that recognizes the same language and has both polynomial complexity and bounded orbit-complexity. We only outline this construction, which is similar to the monoid of transitions of a word automaton.

Let Q_- be $Q \setminus \{q_0\}$. Given a state $q \in Q_-$, a variable x and a tree t possibly with free variables, we define:

$$\delta_t(q, x) := \delta(q, v)$$

in which $v \in C_{\mathcal{V},\Sigma}{}^*$ is such that $T([x]v) = t$ and does not contain a letter of the form $[\cdot_y a(z_0, ..., z_{\mathrm{ar}(a)-1})]$ in which $y \in \mathrm{supp}(q) \setminus \{x\}$. This definition is shown to be meaningful, in the sense that it does not depend on the choice of v.

The elements of $\mathcal{A}$ are then defined to be the δ_t's, and the operations of the algebra are defined so that the image of a tree t under the evaluation morphism is δ_t. This algebra recognizes L and we show, this is the difficult part of the proof, that it has both polynomial complexity and bounded orbit-complexity.

4 Tree algebras

We now tackle the study of tree algebras of polynomial complexity and bounded orbit-complexity, and prove the remaining implications of Theorem 1. We first define in Section 4.1 the notion of system of supports of a tree algebra that appears in Theorem 1. In Section 4.2, we make use of this notion to prove Theorem 1. Finally we state Theorem 22, our decidability result, in Section 4.3.

4.1 Syntactic tree algebras and systems of supports

We start this section by defining systems of supports of a tree algebra. We then prove Lemma 17 which states that a syntactic tree algebra always has a minimal system of supports, and that it is stable (this property is mentionned in Item 3 of Theorem 1). Finally, we prove Proposition 19, corresponding to implications from Items 1 and 2 to Item 3 in Theorem 1.

System of supports. We now introduce the notion of system of supports. It is a way to transport the notion of support from $\mathbf{Sym}(V)$-sets to tree algebras, which are a collection of $\mathbf{Sym}(X)$-sets for X finite.

For every finite set of variables X, a subset S_a of X is a *support* of $a \in A_X$ if $\sigma(a) = a$ for every $\sigma \in \mathbf{Sym}(X \setminus S_a)$. A *system of supports* for an algebra $\mathcal{A}$ is a family $(S_a)_{a\in\mathcal{A}}$ such that S_a is a support of a for all finite X and $a \in A_X$.

Stability. The notion of system of supports is however lacking to describe properties of tree algebras, as there is no relation between the supports of the different elements, we introduce different notions of stability to cope with this issue.

We define the following properties for a system of supports $(S_a)_{a\in\mathcal{A}}$:

- Being *stable under renamings* if $S_{\sigma(a)} \subseteq \sigma(S_a)$ for $a \in A_X$, X finite, and renaming σ.
- Being *stable under internal substitutions* if $S_{a\cdot_x b} \subseteq (S_a \setminus \{x\}) \cup S_b$ for all $a \in A_X$, $x \in S_a$ and $b \in A_Y$, for finite X and Y.
- Being *stable under external substitutions* $S_{a\cdot_x b} \subseteq S_a$ for all $a \in A_X$, $x \in X \setminus S_a$ and all $b \in A_Y$, for finite X and Y.

- Being *stable under substitutions* if it is both stable under internal and external substitutions.

Finally, a system of supports is *stable* if it is both stable under renamings and substitutions, and it is *bounded* if there is a bound K such that $|S_a| \leq K$ for every $a \in \mathcal{A}$.

Our main result concerning systems of supports, Lemma 17, states the existence of a canonical support for syntactic tree algebras. Moreover, this system of supports $(S_a)_{a\in\mathcal{A}}$ is *minimal*, meaning that $S_a \subseteq T_a$, for every $a \in \mathcal{A}$ and every $(T_a)_{a\in\mathcal{A}}$ with the same stability properties.

▶ **Lemma 17.** *Let $\mathcal{A}$ be a syntactic tree algebra. Then $\mathcal{A}$ has a minimal stable system of supports.*

The proof of this result is lengthy and relies on the fact that a Σ, X-tree t can be seen as a Σ, Y-tree for all $X \subseteq Y$, allowing one to use standard techniques from nominal set theory. The system of supports introduced in Lemma 17 is the *canonical system of supports* of $\mathcal{A}$.

▶ **Example 18.** We once again take a look at the tree algebra introduced in Example 5. Its canonical system of supports is given, for all finite X, by:

$$S_\top = S_\bot = \emptyset \ , \qquad S_x = \{x\} \text{ for all } x \in X \ .$$

We now establish Proposition 19 below, corresponding to implications from Item 1 to Item 3 and from Item 2 to Item 3 of Theorem 1. According to Lemma 7, it is enough to prove the results for syntactic tree algebras, meaning we only need to prove that the canonical system of supports is bounded.

▶ **Proposition 19.** *Let $\mathcal{A}$ be a finite syntactic tree algebra that has either polynomial complexity or bounded orbit complexity. Then $\mathcal{A}$ has a bounded and stable system of supports.*

4.2 From tree algebras to coding automata

Our next result is Proposition 20 below, corresponding to implication from Item 3 to Item 4 of Theorem 1.

▶ **Proposition 20.** *Let L be recognized by a finite tree algebra with a bounded and stable system of supports. Then there is a coding automaton that describes L.*

The following Lemma 21 shows that we only need to consider syntactic tree algebras.

▶ **Lemma 21.** *Let L be a language of trees. If L is recognized by a tree algebra that has a bounded and stable system of supports, then the syntactic tree algebra of L can also be equipped with such a system of supports.*

From tree algebra to coding automata. Fix a language of trees L, let $\mathcal{A}$ be its syntactic tree algebra, and let $(S_a)_{a\in\mathcal{A}}$ be its canonical system of supports, which is bounded (by say K) and stable. In order to prove Proposition 20, we extract from $\mathcal{A}$ a coding automaton $\text{Auto}_\mathcal{A}$ that describes L. Intuitively, because $(S_a)_{a\in\mathcal{A}}$ is stable and bounded, $\mathcal{A}$ is uniquely determined by the A_X's for $|X| \leq K$. This is used to defined an appropriate automaton, whose set of states is orbit-finite.

This concludes the proof of Proposition 20 and thus the proof of Theorem 1.

4.3 Decidability

Our last result is the following.

▶ **Theorem 22.** *There is an algorithm which, given a regular tree language, decides whether it is recognizable by a tree algebra of polynomial complexity.*

The proof informally consists in constructing the expected coding automaton, dropping when possible the variables that are irrelevant for deciding the membership to the language (in the sense that, whatever tree is plugged in it, it does not change the membership to the language). The crucial point is to prove that a bounded number of variables suffices. Regular cost functions over finite trees provide a straightforward technique for solving this boundedness question [6].

5 Conclusion

We analyzed the expressive power of unrestrained tree algebras of polynomial complexity. Theorem 1 shows a link between combinatorial properties (being recognized by tree algebras of polynomial complexity, or of bounded orbit-complexity) and an algebraic one (being described by a coding automaton). Doing so, it gives a crisp description of the expressive power of this class of tree algebras, which we also proved to be decidable.

Other types of tree algebras. This theory is easily ported to affine tree algebras, for which we get the same equivalence and decidability results. In the case of relevant and linear tree algebras, however, the breaking point is the existence of a canonical system of supports. At the cost of a much more technical proof, it remains possible to prove the equivalence of Items 1, 3, and 4 in Theorem 1. We conjecture Theorem 1 to still hold.

Future work. A natural extension to this work is to study the expressive power of tree algebras of exponential complexity, which we conjecture to be the same as the one of tree algebras of polynomial orbit-complexity. Such tree algebras are much more expressive than those studied in this article. It is for instance easy to check that both of these classes subsume the one of languages recognized by top-down deterministic tree automata.

References

1 Achim Blumensath. Recognisability for algebras of infinite trees. *Theoretical Computer Science*, 412(29):3463–3486, 2011. doi:10.1016/j.tcs.2011.02.037.

2 Achim Blumensath. Regular Tree Algebras. *Logical Methods in Computer Science*, Volume 16, Issue 1, February 2020. doi:10.23638/LMCS-16(1:16)2020.

3 Mikołaj Bojanczyk. Slightly infinite sets, 2016. A draft of a book available at https://www.mimuw.edu.pl/~bojan/paper/atom-book.

4 Mikolaj Bojanczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16, New York, NY, USA, July 5-8, 2016*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.

5 Thomas Colcombet and Arthur Jaquard. A complexity approach to tree algebras: the bounded case. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

6 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 70–79. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.36.

7 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

8 Zoltán Ésik and Pascal Weil. Algebraic recognizability of regular tree languages. *Theor. Comput. Sci.*, 340(1):291–321, 2005. doi:10.1016/j.tcs.2005.03.038.

9 Zoltán Ésik and Pascal Weil. Algebraic characterization of logically defined tree languages. *Int. J. Algebra Comput.*, 20(2):195–239, 2010. doi:10.1142/S0218196710005595.

10 Sławomir Lasota, Bartek Klin, and Mikołaj Bojańczyk. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10, 2014.

11 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

Enumeration Classes Defined by Circuits

Nadia Creignou
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Arnaud Durand
Université Paris Cité, CNRS, IMJ-PRG, Paris, France

Heribert Vollmer
Leibniz Universität Hannover

Abstract

We refine the complexity landscape for enumeration problems by introducing very low classes defined by using Boolean circuits as enumerators. We locate well-known enumeration problems, e.g., from graph theory, Gray code enumeration, and propositional satisfiability in our classes. In this way we obtain a framework to distinguish between the complexity of different problems known to be in **DelayP**, for which a formal way of comparison was not possible to this day.

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

Keywords and phrases Computational complexity, enumeration problem, Boolean circuit

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.38

Related Version *Full Version*: https://arxiv.org/abs/2205.00539

1 Introduction

In computational complexity theory, most often decision problems are studied that ask for the existence of a solution to some problem instance, e.g., a satisfying assignment of a given propositional formula. In contrast, enumeration problems ask for a list of all solutions, e.g., all satisfying assignments. In many application areas these are the more "natural" kind of problems – let us just mention database queries, web search, diagnosis, data mining, bioinformatics, etc.

The notion of *tractability* for enumeration problems requires a new approach, simply because there may be a large number of solutions, exponential in the input size. Widely studied is the class **DelayP** ("polynomial delay"), containing all enumeration problems where, for a given instance x, (i) the time to compute the first solution, (ii) the time between producing any two consecutive solutions, and (iii) the time to detect that no further solution exists, are all polynomially bounded in the length of x. Also the class **IncP** ("incremental polynomial time"), where we allow the time to produce the next solution and to signal that no further solution exists to grow by a polynomial bounded in the size of the input *plus* the number of already computed solutions. These classes were introduced in 1988 in [14], and since then, an immense number of membership results have been obtained. Recently, also intractable enumeration problems have received some attention. Reducibilities, a completeness notion and a hierarchy of intractable enumeration problems, analogous to the well-known polynomial hierarchy, were defined and studied in [9].

In this paper we will look for notions of tractability for enumeration stricter than the above two. More specifically, we will introduce a refinement of the existing classes based on the computation model of Boolean circuits. The main new class in our framework is the class $\mathbf{Del}\cdot\mathbf{AC}^0$. An enumeration problem belongs to this class if there is a family of $\mathbf{AC}^0$ circuits, i.e., a family of Boolean circuits of constant depth and polynomial size with unbounded fan-in gates, that (i) given the input computes the first solution, (ii) given input

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 38; pp. 38:1–38:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and a solution computes the next solution (in any fixed order of solutions), and (iii) given input and the last solution, signals that no further solution exists. Still using $\mathbf{AC}^0$ circuits we then consider extended classes by allowing

- precomputation of different complexity (typically, polynomial time precomputation) and/or
- memory to be passed on from the computation of one solution to the next (from a constant to a polynomial number of bits)

By this, we obtain a hierarchy of classes within **DelayP/IncP** shown in Fig. 1.

The main motivation behind our work is the wish to be able to compare the complexity of different tractable enumeration problems by classifying them in a fine hierarchy within **DelayP**, and to obtain lower bounds for enumeration tasks. From different application areas such as graph problems, Gray code enumeration and satisfiability, we identify natural problems, all belonging to **DelayP**, some of which can be enumerated in $\mathbf{Del}\cdot\mathbf{AC}^0$, some cannot, but allowing precomputation or a certain number of bits of auxiliary memory they can. We would like to mention in particular the maybe algorithmically most interesting contribution of our paper, the case of enumeration for satisfiability of 2-CNF (Krom) formulas. While it is known that counting satisfying assignments for formulas from this fragment of propositional logic is #**P**-complete [18], we exhibit a $\mathbf{Del}_\mathbf{P}\cdot\mathbf{AC}^0$ algorithm (i.e. $\mathbf{Del}\cdot\mathbf{AC}^0$ with polynomial time precomputation but no memory), for enumeration, thus placing the problem in one of the lowest class in our framework. This means that surprisingly satisfying assignments of Krom formulas can be enumerated very efficiently (only $\mathbf{AC}^0$ is needed to produce the next solution) after a polynomial time precomputation before producing the first solution.

Building on well-known lower bounds (in particular for the parity function [12, 1]) we prove (unconditional) separations among (some of) our classes and strict containment in **DelayP**, and building on well-known completeness results we obtain conditional separations, leading to the inclusions and non-inclusions depicted in Fig. 1.

Another refinement of **DelayP** that has received considerable attention in the past, in particular in the database community, is the class $\mathbf{CD}\circ\mathbf{lin}$ of problems that can be enumerated on RAMs with constant delay after linear time preprocessing [11] (see also the surveys [16, 10]). A consequence of the widely believed assumption that Boolean matrix multiplication cannot be computed in time linear in the number m of non-zero entries of the matrices A, B and AB (the so called BMM conjecture, see e.g. [5]), is that enumerating the one-entries in AB is not in $\mathbf{CD}\circ\mathbf{lin}$ [3], but we will see that it is in $\mathbf{Del}\cdot\mathbf{AC}^0$. On the other hand, the familiar lower bound for parity [1, 12] easily leads to an enumeration problem not in $\mathbf{Del}\cdot\mathbf{AC}^0$ but in $\mathbf{CD}\circ\mathbf{lin}$; hence we see that $\mathbf{CD}\circ\mathbf{lin}$ and $\mathbf{Del}\cdot\mathbf{AC}^0$ are incomparable classes; thus our approach provides a novel way to refine polynomial delay (see Section 3.3).

This paper is organized as follows. After some preliminaries, we introduce our new classes in Sect. 3. In Sect. 4 we present a number of upper and lower bounds for example enumeration problems from graph theory, Gray code enumeration and propositional satisfiability. Depending whether we allow or disallow precomputation steps, we obtain further conditional or unconditional separation results between classes in Sect. 5. Finally we conclude with a number of open problems.

Because of space limitations, some of our proofs are only sketched or even omitted here; full proofs will appear in the final version of this paper.

2 Preliminaries

Since our main computational model will be Boolean circuits, we fix the alphabet $\Sigma = \{0, 1\}$, and use this alphabet to encode graphs, formulas, etc., as usual. Any reasonable encoding will do for all of our results.

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a computable predicate. We say that R is polynomially balanced, if there is a polynomial p such that for all pairs $(x, y) \in R$, we have $|y| \leq p(|x|)$. Now we define the enumeration problem associated to R as follows.

Enum·R
Input: $x \in \Sigma^*$
Output: an enumeration of elements in $\mathrm{Sol}_R(x) = \{y : R(x, y)\}$

We require that R is computable but do not make any complexity assumptions on R. In the enumeration context, it is sometimes stipulated that R is polynomial-time checkable, i.e., membership of (x, y) in R is decidable in time polynomial in the length of the pair [17, 6]. Generally, we do not require this, but we will come back to this point later.

We assume basic familiarity of the reader with the model of Boolean circuits, see, e.g., [20, 7]. We use $\mathbf{AC}^0$ to denote the class languages that can be decided by uniform families of Boolean circuits of polynomial size and constant depth with gates of unbounded fan-in. The class of functions computed by such circuit families is denoted by $\mathbf{FAC}^0$, and for simplicity often again by $\mathbf{AC}^0$. The notation for the corresponding class of languages/functions defined by uniform families of circuits of polynomial size and logarithmic depth with gates of bounded fan-in is $\mathbf{NC}^1$.

The actual type of uniformity used is of no importance for the results of the present paper. However, for concreteness, all circuit classes in this paper are assumed to be uniform using the "standard" uniformity condition, i. e., DLOGTIME-uniformity/$\mathrm{U_E}$-uniformity [4]; the interested reader may also consult the textbook [20].

3 Delay Classes with Circuit Generators

In this section we present the formal definition of our new enumeration classes. As we already said, we will restrict our definition to usual delay classes; classes with incremental delay can be defined analogously, however, we will see that our delay-classes with memory in a sense reflect incremental classes in the circuit model.

The main idea is that the generation of a *next* solution will be done by a circuit from a family; in the examples and lower and upper bounds in the upcoming sections, these families are usually of low complexity like $\mathbf{AC}^0$ or $\mathbf{NC}^1$. The generator will receive the original input word plus the previous solution. Parameters in the definition will be first the complexity of any precomputation before the first solution is output, and second the amount of information passed from the generation of one solution to the next.

3.1 Delay Classes with no Memory

For a family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ of Boolean circuits, circuit C_i will be the circuit in the family with i input gates. When the length of the circuit input is clear from the context, we will usually simply write $C_{|\cdot|}$ to refer to the circuit with appropriate number of input gates. In the subsequent definitions we use $\mathcal{K}$ to denote a complexity classes defined by families of Boolean circuits obeying certain complexity restrictions. Examples are the above mentioned cases $\mathcal{K} = \mathbf{AC}^0$ or $\mathcal{K} = \mathbf{NC}^1$, but any circuit complexity class will do. Our definitions make sense both for uniform and for non-uniform classes.

▶ **Definition 1** ($\mathcal{K}$-delay). *Let R be a polynomially balanced predicate. The enumeration problem* Enum$\cdot R$ *is in* $\mathbf{Del}\cdot\mathcal{K}$ *if there exists a family of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ such that, for all inputs x, there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:*

- $C_{|\cdot|}(x) = y_1 \in \mathrm{Sol}_R(x)$,
- *for all $i < k$:* $C_{|\cdot|}(x, y_i) = y_{i+1} \in \mathrm{Sol}_R(x)$
- $C_{|\cdot|}(x, y_k) = y_k$

Note that by the last requirement, the circuit family signals there is no further solution if the input solution is given again as output. Moreover, we point out that, in the definition above, if x is an input and $y \in \mathrm{Sol}_R(x)$, then $C_{|x|+|y|}$ produces a $z \in \mathrm{Sol}_R(x)$. However, if $y \notin \mathrm{Sol}_R(x)$, nothing is specified about the output z.

Next we consider classes where a precomputation before outputting the first solution is allowed. The ressource bounds of the precomputation are specified by an arbitrary complexity class.

▶ **Definition 2** ($\mathcal{K}$-delay with T-precomputation). *Let R be a polynomially balanced predicate and T be a complexity class. The enumeration problem* Enum$\cdot R$ *is in* $\mathbf{Del}_T\cdot\mathcal{K}$ *if there exists an algorithm M working with resource T and a family of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ such that, for all input x there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:*

- *M computes some value x^*, i.e., $M(x) = x^*$*
- $C_{|\cdot|}(x^*) = y_1 \in \mathrm{Sol}_R(x)$,
- *for all $i < k$:* $C_{|\cdot|}(x^*, y_i) = y_{i+1} \in \mathrm{Sol}_R(x)$
- $C_{|\cdot|}(x^*, y_k) = y_k$

3.2 Delay Classes with Memory

Extending the above model, we now allow each circuit to produce slightly more than the next solution. These additional information is then passed as extra input to the computation of the next solution, in other words, it can serve as an auxiliary memory.

▶ **Definition 3** ($\mathcal{K}$-delay with auxiliary memory). *Let R be a polynomially balanced predicate. The enumeration problem* Enum$\cdot R$ *is in* $\mathbf{Del}^*\cdot\mathcal{K}$ *if there exist two families of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ such that, for all input x there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:*

- $C_{|\cdot|}(x) = y_1^*$ *and* $D_{|\cdot|}(y_1^*) = y_1 \in \mathrm{Sol}_R(x)$,
- *for all $i < k$:* $C_{|\cdot|}(x, y_i^*) = y_{i+1}^*$ *and* $D_{|\cdot|}(y_{i+1}^*) = y_{i+1} \in \mathrm{Sol}_R(x)$,
- $C_{|\cdot|}(x, y_k^*) = y_k^*$,
- *for $1 \leq i \leq k$, y_i is a prefix of y_i^*; i. e., the information y_i^* passed on to the next round consists of the previous solution y_i plus any additional information.*

When there exists a polynomial $p \in \mathbb{N}[x]$ such that $|y_i^*| \leq p(|x|)$, for all $i \leq k$, the class is called $\mathbf{Del}^{\mathrm{P}}\cdot\mathcal{K}$*, $\mathcal{K}$-*delay with polynomial auxiliary memory. *When there exists a constant $c \in \mathbb{N}$ such that $|y_i^*| \leq |y_i| + c$, for all $i \leq k$, the class is called* $\mathbf{Del}^{\mathrm{c}}\cdot\mathcal{K}$*, $\mathcal{K}$-*delay with constant auxiliary memory.

The idea is that the y_i^* will contain the previous solution plus the additional memory. Hence the superscript “c” indicates a bounded auxiliary memory size.

By abuse of expression, we will sometimes say that a problem in some of these classes above can be enumerated with a delay in $\mathcal{K}$ or with a $\mathcal{K}$-delay. When there is no restriction on memory i.e. when considering the class $\mathbf{Del}_T^*\cdot\mathcal{K}$, an incremental enumeration mechanism can be used. Indeed, the memory can then store all solutions produced so far which results in an increase of the expressive power.

Also in the case of memory, we allow possibly precomputation before the first output is made:

▶ **Definition 4** ($\mathcal{K}$-delay with T-precomputation and auxiliary memory)**.** *Let R be a polynomially balanced predicate and T be a complexity class. The enumeration problem* Enum·R *is in* $\mathbf{Del}^*_T{\cdot}\mathcal{K}$ *if there exists an algorithm M working with resource T and two families of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ such that, for all input x there is an enumeration $y_1, ..., y_k$ of* $\mathrm{Sol}_R(x)$ *and:*

- *M computes some value x^*, i.e., $M(x) = x^*$,*
- *$C_{|\cdot|}(x^*) = y^*_1$ and $D_{|\cdot|}(y^*_1) = y_1 \in \mathrm{Sol}_R(x)$,*
- *for all $i < k$: $C_{|\cdot|}(x^*, y^*_i) = y^*_{i+1}$ and $D_{|\cdot|}(y^*_{i+1}) = y_{i+1} \in \mathrm{Sol}_R(x)$,*
- *$C_{|\cdot|}(x^*, y^*_k) = y^*_k$,*
- *for $1 \le i \le k$, y_i is a prefix of y^*_i.*

When there exists a polynomial $p \in \mathbb{N}[x]$ such that $|y^*_i| \le p(|x|)$, for all $i \le k$, the class is called $\mathbf{Del}^{\mathrm{P}}_T{\cdot}\mathcal{K}$, $\mathcal{K}$-delay with T-precomputation and polynomial auxiliary memory*. When there exists a constant $c \in \mathbb{N}$ such that $|y^*_i| \le |y_i| + c$, for all $i \le k$, the class is called $\mathbf{Del}^{\mathrm{c}}_T{\cdot}\mathcal{K}$,* $\mathcal{K}$-delay with T-precomputation and constant auxiliary memory*.*

3.3 Relation to Known Enumeration Classes

All classes we consider in this paper are subclasses of the well-known classes **DelayP** or **IncP**, resp., even if we allow our circuit to be of arbitrary depth (but polynomial size).

▶ **Theorem 5.** *If $\mathcal{K}, T \subseteq \mathbf{P}$, then $\mathbf{Del}^{\mathrm{P}}_T{\cdot}\mathcal{K} \subseteq \mathbf{DelayP}$ and $\mathbf{Del}^*_T{\cdot}\mathcal{K} \subseteq \mathbf{IncP}$.*

Let us briefly clarify the relation between our classes and the class **CD∘lin** of enumeration problems that have a constant delay on a RAM after linear-time precomputation. This class was introduced in [11].

The algorithmic problem, given a graph, to enumerate all pairs of vertices that are connected by a path of length 2 has only a polynomial number of solutions and is trivially in $\mathbf{Del}{\cdot}\mathbf{AC}^0$. Since it is essentially the same as Boolean matrix multiplication, it is not in **CD∘lin**, assuming the beforementioned BMM hypothesis.

On the other hand, note that the enumeration problem Enum-Parity, given as input a sequence of bits with the solution set consisting only of one solution, the parity of the input, is not in $\mathbf{Del}{\cdot}\mathbf{AC}^0$, since the parity function is not in $\mathbf{AC}^0$ [12, 1]. However, since Parity can be computed in linear time, Enum-Parity is trivially in **CD∘lin**.

As we will show in detail in the full version of this paper, the computation of a constant number of time steps of a RAM can be simulated by $\mathbf{AC}^0$ circuits. Hence if we add linear precomputation and polynomial memory to save the configuration of the RAM, we obtain an upper bound for **CD∘lin**. To summarize:

▶ **Theorem 6.** *Assuming the BMM hypothesis, the classes $\mathbf{Del}{\cdot}\mathbf{AC}^0$ and **CD∘lin** are incomparable, and* $\mathbf{CD{\circ}lin} \subsetneq \mathbf{Del}^{\mathrm{P}}_{lin}{\cdot}\mathbf{AC}^0$.

4 Examples

In this section we show that many natural problems, ranging from graph problems, enumeration of Gray codes and satisfiability problems lie in our circuit classes.

4.1 Graph Problems

We first consider the enumeration problem associated with the notion of reachability in a graph.

Enum-Reach
Input: a graph $G = (V, E)$, $s \in V$
Output: an enumeration of vertices reachable from s

▶ **Theorem 7.** Enum-Reach $\in \mathbf{Del}^{\mathrm{P}} \cdot \mathbf{AC}^0$

Proof. At each step multiplication of Boolean matrices gives the set of vertices which are reachable from s with one more step. This can be done in $\mathbf{AC}^0$. The polynomial memory is used to remember all vertices that have been encountered to far. Observe that in each step, more vertices are being produced, until we reach convergence. ◀

Let us now turn to the enumeration of all transversals (i. e., vertex-sets intersecting every edge), not only the minimal ones.

Enum-Transversal
Input: A hypergraph $H = (V, \mathcal{E})$
Output: an enumeration of all transversals of H

▶ **Theorem 8.** Enum-Transversal $\in \mathbf{Del} \cdot \mathbf{AC}^0$.

Proof. Let $\mathcal{E}$ be a set of hyperedges over a set of n vertices. Every binary word $y = y_1 \cdots y_n \in \{0,1\}^n$ can be interpreted as a subset of vertices. We propose an algorithm that enumerates each of these words that corresponds to a transversal of H in lexicographical order with the convention $1 < 0$. The algorithm is as follows:

- As a first step output $1 \ldots 1$, the trivial solution.
- Let H be the input and y be the last output solution.
  - For each prefix $y_1 \ldots y_i$ of y with $y_i = 1$ and $i \leq n$ consider the word of length n, $z^i = y_1 \ldots y_{i-1} 01 \ldots 1$.
  - Check whether at least one of these words z^i is a transversal of H.
  - If yes select the one with the longest common prefix with y, that is the transversal z^i with the largest i and output it as the next solution.
  - Else stop.

First we prove that the algorithm is correct. The transversal that is the successor of y in our lexicographical order (where $1 < 0$), if it exists, has a common prefix with y, then a bit flipped from 1 to 0, and finally is completed by 1's only. Indeed, a successor of y necessarily starts this way, and by monotonicity the first extension of such a prefix into a solution is the one completed by 1's only. As a consequence our algorithm explores all possible candidates and select the next transversal in the lexicographical order.

Now let us prove that this is an $\mathbf{AC}^0$-delay enumeration algorithm that does not require memory. The main observation is that one can check with an $\mathbf{AC}^0$ circuit whether a binary word corresponds to transversal of H. Now, for each i we can use a sub-circuit, which on input (H, y) checks whether $y_i = 1$ and if yes whether z^i is a transversal of H. This circuit can output $(z^i, 1)$ if both tests are positive, and $(y_i, 0)$ otherwise. All these sub-circuits can be wired in parallel. Finally it suffices to use a selector to output z^i with the largest i for which $(z^i, 1)$ is output at the previous step. Such a selector can be implemented by an $\mathbf{AC}^0$ circuit. ◀

It is then easy to show (in a similar way) that enumeration of all dominating sets of a graph can be done in $\mathbf{Del}{\cdot}\mathbf{AC}^0$.

4.2 Gray Code

Given $n \in \mathbb{N}$, a Gray n-code is a ranked list of elements of Σ^n such that between two successive words x, y there exists only one bit such that $x_i \neq y_i$. Since we deal with Boolean circuits, we have to fix $\Sigma = \{0,1\}$, but Gray codes are defined for arbitrary alphabets.

The binary reflected Gray code of length n, denoted G^n, is made of 2^n words: $G^n = [G^n_0, G^n_1, \ldots, G^n_{2^n-1}]$. It is defined recursively as follows: $G^1 = [0,1]$ and, for $n \geq 1$

$$G^n = [0G_0^{n-1}, 0G_1^{n-1}, \ldots, 0G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, \ldots, 1G_1^{n-1}, 1G_0^{n-1}].$$

As an example let us consider the list of pairs $(rank, word)$ for $n = 4$: $(0, 0000)$, $(1, 0001)$, $(2, 0011)$, $(3, 0010)$, $(4, 0110)$, $(5, 0111)$, $(6, 0101)$, $(7, 0100)$, $(8, 1100)$, $(9, 1101)$, $(10, 1111)$, $(11, 1110)$, $(12, 1010)$, $(13, 1011)$, $(14, 1001)$, $(15, 1000)$.

Given n and $r < 2^n$, let $b_{n-1} \cdots b_1 b_0$ be the binary decomposition of r and $G^n_r = a_{n-1} \cdots a_1 a_0 \in \Sigma^n$ be the rth word in the binary reflected code of length n. It is well-known that, for all $j = 0, ..., n-1$,

$$b_j = \sum_{i=j}^{n-1} a_i \bmod 2 \text{ and } a_j = (b_j + b_{j+1}) \bmod 2.$$

Hence computing the rank of a word in the binary reflected code amounts to be able to compute parity. On the other side, computing the word from its rank can easily be done by a circuit.

While it is trivial to enumerate all words of length n in arbitrary or lexicographic order, this is not so clear for Gray code order. Also, given a rank or a first word, to enumerate all words of higher Gray code rank (in arbitrary order) are interesting computational problems.

Enum-Gray-Rank
Input: a binary word r of length n interpreted as an integer in $[0, 2^n[$
Output: an enumeration of words of G^n that are of rank at least r.

Enum-Gray-Word
Input: a word x of length n
Output: an enumeration of words of G^n, that are of rank at least the rank of x.

It turns out that for those problems where the order of solutions is not important, a very efficient enumeration is possible:

▶ **Theorem 9.** *Let n be an integer*

1. *Given 1^n, enumerating all words of length n even in lexicographic ordering is in* $\mathbf{Del}{\cdot}\mathbf{AC}^0$
2. Enum-Gray-Rank $\in \mathbf{Del}{\cdot}\mathbf{AC}^0$
3. Enum-Gray-Word $\in \mathbf{Del}{\cdot}\mathbf{AC}^0$

We next turn to those versions of the above problems, where we require that solutions are given one after the other in Gray code order. For each of them, the computational complexity is provably higher than in the above cases.

▶ **Theorem 10.** *Given 1^n, enumerating all words of length n in a Gray code order is in* $\mathbf{Del}^{\mathrm{c}}{\cdot}\mathbf{AC}^0 \setminus \mathbf{Del}_{\mathbf{P}}{\cdot}\mathbf{AC}^0$

Proof. A classical method to enumerate gray code of length n is the following [15].

- Step 0 : produce the word $0 \cdots 0$ of length n.
- Step $2k+1$: switch the bit at position 0.
- Step $2k+2$: find minimal position i where there is a 1 and switch bit at position $i+1$.

This method can be turned into an $\mathbf{AC}^0$-delay enumeration without precomputation using one bit of memory (to keep trace if the step is an even or odd one all along the computation). This proves the membership in $\mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0$.

For the lower bound, suppose $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ is an $\mathbf{AC}^0$ circuit family enumerating the Gray code of length n after polynomial time precomputation produced by machine M. We will describe how to use $\mathcal{C}$ to construct an $\mathbf{AC}^0$-family computing the parity function, contradicting the lower bound given by [1, 12].

Given is an arbitrary word $w = w_{n-1} \ldots w_0$ of length n, and we want to compute its parity $\left(\sum_{i=0}^{n-1} w_i\right) \bmod 2$. Let $x^* = M(1^n)$. Then, w will appear as a solution somewhere in the enumeration defined by $\mathcal{C}$. Let w' be the next words after w. There exists r such that $G_r^n = w$ and $G_{r+1}^n = w'$. By comparing w and w', one can decide which transformation step has been applied to w to obtain w' and thus if r is odd or even. Note that the parity of w is 1 if and only if r is odd. Hence, one can compute parity by a constant depth circuit operating as follows:

```
Input $w$:
    $n := |w|$;
    $x^* := M(1^n)$;
    $w' := C_{|\cdot|}(x^*, w)$;
    if last bits of $w$ and $w'$ differ then $v := 1$ else $v := 0$;
    output $v$.
```

Note that the computation of x^* does not depend on w but only on the length of w; hence x^* can be hardwired into the circuit family, which, since M runs in polynomial time, will then be P-uniform. But we know from [12, 1] that parity cannot even be computed by non-uniform $\mathbf{AC}^0$ circuit families. ◀

We also consider the problem of enumerating all words starting not from the first one but at a given position, but now in Gray code order. Surprisingly this time the complexity will depend on how the starting point is given, by rank or by word.

Enum-Gray-Rank$_{ord}$
Input: A binary word r of length n interpreted as an integer in $[0, 2^n[$
Output: an enumeration of words of G^n in increasing number of ranks starting from rank r.

Enum-Gray-Word$_{ord}$
Input: A word x of length n
Output: an enumeration of words of G^n in Gray code order that are of rank at least the rank of x.

▶ **Theorem 11.** 1. Enum-Gray-Rank$_{ord} \in \mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0 \setminus \mathbf{Del}_{\mathbf{P}} \cdot \mathbf{AC}^0$.
2. Enum-Gray-Word$_{ord}$ *is in the class* $\mathbf{Del}^{\mathrm{c}}_{\mathbf{P}} \cdot \mathbf{AC}^0$*, but neither in* $\mathbf{Del}_{\mathbf{P}} \cdot \mathbf{AC}^0$ *nor* $\mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0$.

4.3 Satisfiability Problems

Deciding the satisfiability of a CNF-formula is well-known to be $\mathbf{NP}$-complete. Nevertheless the problem becomes tractable for some restricted classes of formulas. For such classes we investigate the existence of an $\mathbf{AC}^0$-delay enumeration algorithm. First we consider monotone formulas.

Enum-Monotone-Sat
Input: A set of positive (resp. negative) clauses Γ over a set of variables V
Output: an enumeration of all assignments over V that satisfy Γ

The following positive result is an immediate corollary of Theorem 8.

▶ **Theorem 12.** Enum-Monotone-Sat $\in \mathbf{Del}\cdot\mathbf{AC}^0$.

If we allow polynomial precomputation, then we obtain an $\mathbf{AC}^0$-delay enumeration algorithm for a class of CNF-formulas, referred to as IHS in the literature (for Implicative Hitting Sets, see [8]), which is larger than the monotone class. A formula in this class consists of monotone clauses (either all positive or all negative) together with implicative clauses.

Enum-IHS-Sat
Input: A set of clauses $\mathcal{C}$ over a set of variables V, with $\mathcal{C} = \mathcal{M} \cup \mathcal{B}$, where $\mathcal{M}$ is a set of positive clauses (resp. negative clauses) and $\mathcal{B}$ a set of clauses of the form $(\neg x)$ or $(x \vee \neg x')$ (resp. of of the form (x) or $(x \vee \neg x')$)
Output: an enumeration of all assignments over V that satisfy Γ

▶ **Theorem 13.** Enum-IHS-Sat $\in \mathbf{Del_P}\cdot\mathbf{AC}^0 \setminus \mathbf{Del}^*\cdot\mathbf{AC}^0$.

Proof sketch. Observe that contrary to the monotone case 1....1 is not a trivial solution. Indeed a negative unary clause $(\neg x)$ in $\mathcal{B}$ forces x to be assigned 0, and this truth value can be propagated to other variables by the implicative clauses of the form $(x \vee \neg x')$. For this reason as a precomputation step, for each variable x we compute $\mathrm{tc}(x)$ the set of all variables that have to be set to 0 in any assignment satisfying Γ in which x is assigned 0. With this information we can use an algorithm that enumerates all truth assignments satisfying Γ in lexicographical order very similar to the one used for enumerating the transversals of a graph (see the proof of Theorem 8).

For the lower bound, consider the st-connectivity problem: given a directed graph $G = (V, A)$ with two distinguished vertices s and t, decide whether there exists a path from s to t. From G, s and t we build an instance of Enum-IHS-Sat as follows. We consider a set a clauses $\mathcal{C} = \mathcal{P} \cup \mathcal{B}$, where $\mathcal{P} = \{(s \vee t)\}$ and $\mathcal{B} = \{(\neg s)\} \cup \{(x \vee \neg y) \mid (x, y) \in A\}$. This is an $\mathbf{AC}^0$-reduction.

Observe that there exists a path from s to t if and only if Γ is unsatisfiable. Suppose that Enum-IHS-Sat $\in \mathbf{Del}\cdot\mathbf{AC}^0$, this means in particular that outputting a first assignment satisfying Γ or deciding there is none is in $\mathbf{AC}^0$. Thus the above reduction shows that st-connectivity is in $\mathbf{AC}^0$, thus contradicting the fact that st-connectivity is known not to be in $\mathbf{AC}^0$ (see [12, 1]). ◀

Surprisingly the enumeration method used so far for satisfiability problems presenting a kind of monotonicity can be used for the enumeration of all assignments satisfying a Krom set of clauses (i.e., a 2-CNF formula) as soon as the literals are considered in an appropriate order.

▶ **Theorem 14.** Enum-Krom-Sat $\in \mathbf{Del_P}\cdot\mathbf{AC}^0 \setminus \mathbf{Del}^*\cdot\mathbf{AC}^0$.

Proof sketch. The proof builds on the algorithm in [2] that decides whether a set of Krom clauses is satisfiable in linear time.

Let Γ be a set of 2-clauses over a set of n variables V. We perform the following precomputation steps:

- Build the associated implication graph, i.e., the directed graph G whose set of vertices is the set of literals $V \cup \{\bar{v} : v \in V\}$. For any 2-clause $(l \vee l')$ in Γ there are two arcs $\bar{l} \to l'$ and $\bar{l'} \to l$ in G.

- For each literal l compute $\text{tc}(l)$ the set of vertices that are reachable from l in G.
- Compute the set of strongly connected components of G. If no contradiction is detected, that is if no strongly connected component contains both a variable x and its negation, then contract each strongly connected component into one vertex. The result of this operation is a DAG, which, by abuse of notation, we also call G.
- Compute a topological ordering of the vertices of G.
- In searching through this topological ordering, build an ordered sequence M of n literals corresponding to the first occurrences of each variable.

If the set of clauses is satisfiable, one can enumerate the satisfying assignments given as truth assignments on M in lexicographic order. The enumeration process is similar in spirit as the one developed in the preceding theorem.

For the lower bound, the proof given in Theorem 13 applies. ◀

We next turn to the special case where clauses are XOR-clauses, i.e., clauses in which the usual "or" connective is replaced by the exclusive-or connective, $\oplus$. Such a clause can be seen as a linear equation over the two elements field $\mathbb{F}_2$.

Enum-XOR-Sat
Input: A set of XOR-clauses Γ over a set of variables V
Output: an enumeration of all assignments over V that satisfy Γ

If we allow a polynomial precomputation step, then we obtain an $\mathbf{AC}^0$-delay enumeration algorithm for this problem that uses constant memory. Interestingly this algorithm relies on the efficient enumeration of binary words in a Gray code order that we have seen in the previous section and contrary to the satisfiability problems studied so far does not provide an enumeration in lexicographic order.

▶ **Theorem 15.** Enum-XOR-Sat $\in \mathbf{Del}^{\mathrm{c}}_{\mathbf{P}}\cdot\mathbf{AC}^0 \setminus \mathbf{Del}^*\cdot\mathbf{AC}^0$.

Proof sketch. Observe that a set of XOR-clauses Γ over a set of variables $V = \{x_1, \ldots x_n\}$ can be seen as a linear system over V on the two elements field $\mathbb{F}_2$. As a consequence enumerating all assignments over V that satisfy Γ comes down to enumerating all solutions of the corresponding linear system.

As a precomputation step we apply Gaussian elimination in order to obtain an equivalent triangular system. If the system has no solution, we indicate this as required by Definition 1. Otherwise we can suppose that the linear system is of rank $n-k$ for some $0 \leq k \leq n-1$, and without loss of generality that $x_1, \ldots, x_k$ are free variables, whose assignment determines the assignment of all other variables in the triangular system. We then compute a first solution s_0 corresponding to $x_1, \ldots, x_k$ assigned $0 \ldots 0$. Next, for each $i = 1, \ldots, k$ compute the solution s_i corresponding to all variables in $x_1, , \ldots x_k$ assigned 0 except x_i which is assigned 1. Compute then the influence list of x_i, $L(x_i) = \{j \mid k+1 \leq j \leq n, s_0(x_j) \neq s_i(x_j)\}$. The influence list of x_i gives the bits that will be changed when going from a solution to another one in flipping only the bit x_i in the prefix corresponding to the free variables. Observe that this list does not depend on the solution (s_0 in the definition) we start from.

With this precomputation we start our enumeration procedure, which uses the enumeration of binary prefixes of length k in a Gray code order as a subprocedure. ◀

5 Separations of Delay Classes

In the previous results we already presented a few lower bounds, but now we will systematically strive to separate the studied classes.

As long as no precomputation is allowed, we are able to separate all delay classes – with the exception of the class with unbounded auxiliary memory. With precomputation, the situation seems to be more complicated. We obtain only a conditional separation of the class with constant memory from the one without memory at all.

5.1 Unconditional Separations for Classes without Precomputation

▶ **Theorem 16.** $\mathbf{Del}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0$

Proof. Let $x \in \{0,1\}^*$, $|x| = n \in \mathbb{N}^*$, $x = x_1 \dots x_n$. We denote by $m = \lceil \log n \rceil + 1$. Let R_L be defined for all $x \in \{0,1\}^*$ as the union of the two following sets A and B:

- $A = \{\, y \in \{0,1\}^* \big| |y| = m, y \neq 0^m, y \neq 1^m \,\}$
- $B = \{\, 1^m \}$ if x has an even number of ones, else $B = \{\, 0^m \}$.

We denote by $z_1, ..., z_t$ an enumeration of elements of A. Clearly, $|R_L(x)| = t + 1$ and $t \geq n$. To show that $R_L \in \mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0$, we use the enumeration of elements of A (which is easy) and one additional memory bit that is transferred from one step to the other to compute Parity. Indeed, we build families of circuits (C_n) and (D_n) according to Definition 3 as follows.

- First $C_{|\cdot|}(x)$ computes $y_1^* = z_1 b^1$ where $b^1 = x_1$, and $D_{|\cdot|}(y_1^*) = z_1$.
- For $1 < i \leq t$, the circuit $C_{|\cdot|}(x, y_{i-1}^*)$ computes y_i^*, where $y_i^* = z_i b^i$ with $b^i = b^{i-1} \oplus x_i$ if $i \leq n$, and $b^i = b^{i-1}$ else, and $D_{|\cdot|}(y_i^*) = z_i$.
- After t steps, the memory bit b^t contains a 0 if and only if the number of ones in x is even. According to this, we either output 1^m or 0^m as last solution.

Note that the size of the solutions is m, the size of the memory words above is $m + 1$, hence we need constant amount of additional memory. The circuit families (C_n) and (D_n) are obviously DLOGTIME-uniform.

Suppose now that $R_L \in \mathbf{Del}\cdot\mathbf{AC}^0$ and let (C_n) be the associated family of enumeration circuits. We construct a circuit family as follows: We compute in parallel all $C_{|\cdot|}(x)$ and $C_{|\cdot|}(x, z_i)$ for $1 \leq i \leq t$. In this way, we will obtain among other solutions either 0^m or 1^m. We accept in the first case. Note that the z_i are the same for all inputs x of the same length. Thus, we obtain an $\mathbf{AC}^0$ circuit family for parity, contradicting [12, 1]. ◀

By extending the above approach, one can prove the following separation:

▶ **Theorem 17.** $\mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0$

The parity problem can be seen as an enumeration problem: given x, one output the unique solution 1 if the number of ones in x is even. One outputs 0 if it is odd. Since as a function problem, parity can not be in $\mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0$ (the fact there is only one solution makes memory useless). It is obviously in **DelayP**. This implies that $\mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{DelayP}$. Putting all the previous results together, we conclude:

▶ **Corollary 18.** $\mathbf{Del}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{DelayP}$.

5.2 Conditional Separation for Classes with Precomputation

If precomputation is allowed, the separation proofs of the previous subsection no longer work; in fact we do not know if the corresponding separations hold. However, under reasonable complexity-theoretic assumptions we can at least separate the classes $\mathbf{Del}_{\mathbf{P}}\cdot\mathbf{AC}^0$ and $\mathbf{Del}^{\mathrm{c}}_{\mathbf{P}}\cdot\mathbf{AC}^0$. Note that in Theorem 10 we already proved a separation of just these two classes, but this concerns only the special case of ordered enumeration, and does not say anything about the general case. We find it interesting that the proof of the result below relies on a characterization of the class **PSPACE** in terms of regular leaf-languages or *serializable computation* [13, 19].

▶ **Theorem 19.** *If* $\mathbf{NP} \neq \mathbf{PSPACE}$*, then* $\mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0 \setminus \mathbf{Del}_{\mathbf{P}}\cdot\mathbf{AC}^0 \neq \emptyset$.

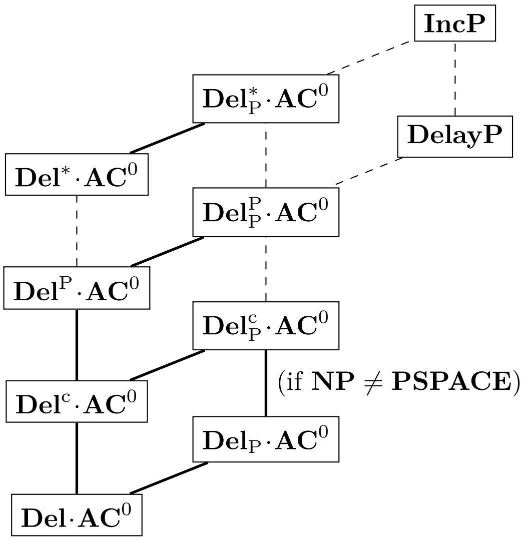

$\mathbf{IncP}$
$\mathbf{Del}^*_{\mathrm{P}}\cdot\mathbf{AC}^0$
$\mathbf{DelayP}$
$\mathbf{Del}^*\cdot\mathbf{AC}^0$
$\mathbf{Del}^{\mathrm{P}}_{\mathrm{P}}\cdot\mathbf{AC}^0$
$\mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0$
$\mathbf{Del}^{\mathrm{c}}_{\mathrm{P}}\cdot\mathbf{AC}^0$
(if $\mathbf{NP} \neq \mathbf{PSPACE}$)
$\mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0$
$\mathbf{Del}_{\mathrm{P}}\cdot\mathbf{AC}^0$
$\mathbf{Del}\cdot\mathbf{AC}^0$

Figure 1 Diagram of the classes. Bold lines denote strict inclusions.

6 Conclusion

The obtained inclusion relations among the classes we introduced are summarized in Fig. 1. We noted earlier that in our context, enumeration problems are defined without a complexity assumption concerning the underlying relation. We should remark that quite often, a polynomial-time upper bound is required, see [17, 6]. All of our results, with the exception of the conditional separations in Sect. 5 also hold under the stricter definition; however, the relation R_L used in the lower bounds in Subsect. 5.2 is based on a **PSPACE**-complete set and therefore, to check whether $y \in R_L(x)$ requires polynomial space w.r.t. the length of x. It would be nice to be able to base these separations on polynomial-time checkable relations, or even better, to separate the classes unconditionally, but this remains open. Moreover, some further inclusions in Fig. 1 are still not known to be strict.

In Subsect. 4.3, we proved that, for several fragments of propositional logic, among them the Krom and the affine fragments, the enumeration of satisfiable assignments is in the class $\mathbf{Del}_{\mathbf{P}}\cdot\mathbf{AC}^0$. This means satisfiable assignments can be enumerated very efficiently, i. e., by an $\mathbf{AC}^0$-circuit family, after some precomputation, which is also efficiently doable (in polynomial time). For another important and very natural fragment of propositional logic, namely the Horn fragment, a **DelayP**-algorithm is known, but it is not at all clear how polynomial-time precomputation can be of any help to produce more than one solution. Since Horn-Sat is **P**-complete, we conclude that Enum-Horn-Sat $\notin \mathbf{Del}^*\cdot\mathbf{AC}^0$, and we conjecture that it is not in $\mathbf{Del}_{\mathbf{P}}\cdot\mathbf{AC}^0$. In fact, we do not see any reasonable better bound than the known **DelayP**.

References

1 Miklós Ajtai. First-order definability on finite structures. *Annals of Pure and Applied Logic*, 45:211–225, 1989.

2 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Computer Science Logic (CSL)*, pages 208–222, 2007.

4 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci. (JCSS)*, 41(3):274–306, 1990.

5 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.

6 Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discret. Appl. Math.*, 268:179–190, 2019. doi:10.1016/j.dam.2018.06.038.

7 Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04943-3.

8 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*, volume 7 of *SIAM monographs on discrete mathematics and applications*. SIAM, 2001.

9 Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. A complexity theory for hard enumeration problems. *Discret. Appl. Math.*, 268:191–209, 2019. doi:10.1016/j.dam.2019.02.025.

10 Arnaud Durand. Fine-grained complexity analysis of queries: From decision to counting and enumeration. 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 331–346. ACM, 2020. doi:10.1145/3375395.3389130.

11 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007. doi:10.1145/1276920.1276923.

12 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

13 Ulrich Hertrampf, Clemans Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207. IEEE Computer Society Press, 1993.

14 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

15 Donald L. Kreher and Douglas Robert Stinson. *Combinatorial Algorithms: generation, enumeration, and search*. CRC Press, 1999.

16 Luc Segoufin. A glimpse on constant delay enumeration (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 13–27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.13.

17 Yann Strozecki. Enumeration complexity. *Bull. EATCS*, 129, 2019. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/596/605.

18 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

19 Heribert Vollmer. A generalized quantifier concept in computational complexity theory. In Jouko A. Väänänen, editor, *Generalized Quantifiers and Computation, 9th European Summer School in Logic, Language, and Information, ESSLLI'97 Workshop, Revised Lectures*, volume 1754 of *Lecture Notes in Computer Science*, pages 99–123. Springer, 1999. doi:10.1007/3-540-46583-9_5.

20 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach.* Springer, Heidelberg, 1999. doi:10.1007/978-3-662-03927-4.

Bounding the Escape Time of a Linear Dynamical System over a Compact Semialgebraic Set

Julian D'Costa
Department of Computer Science, University of Oxford, UK

Engel Lefaucheux
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany
Université de Lorraine, Inria, LORIA, Nancy, France

Eike Neumann
Swansea University, Swansea, UK

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

James Worrell
Department of Computer Science, University of Oxford, UK

Abstract

We study the Escape Problem for discrete-time linear dynamical systems over compact semialgebraic sets. We establish a uniform upper bound on the number of iterations it takes for every orbit of a rational matrix to escape a compact semialgebraic set defined over rational data. Our bound is doubly exponential in the ambient dimension, singly exponential in the degrees of the polynomials used to define the semialgebraic set, and singly exponential in the bitsize of the coefficients of these polynomials and the bitsize of the matrix entries. We show that our bound is tight by providing a matching lower bound.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Discrete linear dynamical systems, Program termination, Compact semialgebraic sets, Uniform termination bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.39

Related Version *Full Version*: `https://arxiv.org/abs/2207.01550`

Funding *Julian D'Costa*: emmy.network foundation under the aegis of the Fondation de Luxembourg.
Joël Ouaknine: Also affiliated with Keble College, Oxford as `emmy.network` Fellow, and supported by DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).

1 Introduction

An *invariant set* of a dynamical system is a set K such that every trajectory that starts in K remains in K. Dually, an *escape set* K is one such that every trajectory that starts in K eventually leaves K (either temporarily or permanently). While it is usually straightforward to establish that a given set K is invariant, it can be challenging to decide whether it is an escape set. Indeed, while the former problem amounts to showing that K is closed under the transition function, the latter potentially involves considering entire orbits. In particular, even in case K has a finite escape time (the maximum number of steps for an orbit to escape the set), it can be highly non-trivial to establish an explicit upper bound on the escape time.

In this paper we focus on escape sets for (discrete-time) linear dynamical systems. Given a rational matrix $A \in \mathbb{Q}^{n\times n}$ we say that $K \subseteq \mathbb{R}^n$ is an escape set for A if for all points $x \in K$, there exists $t \in \mathbb{N}$ such that $A^t x \notin K$. The *compact escape problem (CEP)* asks to decide whether a given compact semialgebraic set K is an escape set for a given matrix A.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 39; pp. 39:1–39:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Decidability of CEP was shown in [18] and its computational complexity was characterised in [9] as being interreducible with the decision problem for a certain fragment of the theory of real closed fields.

The present paper focusses exclusively on positive instances (A, K) of CEP, that is, we assume that we are given a compact semialgebraic escape set for a linear dynamical system. In this situation it turns out, due to compactness of K, that there exists a finite time T such that for all $x \in K$ there exists $t \leq T$ with $A^t x \notin K$. The least such T is called the *escape time* of (A, K). Our main result (Theorem 1, shown below) gives an explicit upper bound on the escape time of (A, K) as a function of the length of the description of the matrix A and semialgebraic set K. In general, it is recognised that bounded liveness is a more useful property than mere liveness. Theorem 1 can be used to establish bounded liveness of several kinds of systems. For example, the result gives an upper bound on the termination time of a single-path linear loop with compact guard (cf. [22, 5]); it also gives a bound on the number of steps to remain in a particular control location of a hybrid system before a given (compact) state invariant becomes false, forcing a transition.

We next introduce some terminology to formalise our main contribution. We say that a semialgebraic set S has complexity at most (n, d, τ) if it can be expressed by a boolean combination of polynomial equations and inequalities $P(x_1, \ldots, x_n) \bowtie 0$ with $\bowtie \in \{\leq, =\}$, involving polynomials $P \in \mathbb{Z}[x_1, \ldots, x_n]$ in at most n variables of total degree at most d with integer coefficients bounded in bitsize by τ. Our main result is as follows:

► **Theorem 1.** *There exists an integer function* $\mathrm{CompactEscape}(n, d, \tau) \in 2^{(d\tau)^{n^{O(1)}}}$ *with the following property. If $K \subseteq \mathbb{R}^n$ is a compact semialgebraic set of complexity at most (n, d, τ) that is an escape set for a matrix $A \in \mathbb{Q}^{n \times n}$ with entries of bitsize at most τ, then the escape time of K is bounded by* $\mathrm{CompactEscape}(n, d, \tau)$.

As explained in the proof sketch below, Theorem 1 relies on the availability of certain quantitative bounds within semialgebraic geometry and number theory, particularly concerning quantifier elimination and Diophantine approximation. The latter results are crucial to handling the case in which the matrix A has complex eigenvalues of absolute value one.

Note that the upper bound on the escape time in Theorem 1 is singly exponential in the degrees and the bitsize of the coefficients of the polynomials used to define K and the bitsize of the coefficients of A. It is doubly exponential in the dimension. In Section 8 we provide two examples, one where A is an isometry and another in which all eigenvalues of A have absolute value strictly greater than one, that yield a corresponding lower bound of this form. It is moreover straightforward to give examples of non-compact escape sets for which the escape time is infinite.

Proof Overview. Let us now give a high-level overview of the proof of Theorem 1. As in the statement of the theorem, let $K \subseteq \mathbb{R}^n$ be a compact semialgebraic set of complexity at most (n, d, τ) and let $A \in \mathbb{Q}^{n \times n}$ be a matrix with entries of bitsize bounded by τ, and such that for all $x \in K$ there exists $t \in \mathbb{N}$ such that $A^t x \notin K$.

To facilitate the analysis of the dynamical behaviour of A we first transform our system into real Jordan normal form. A theorem of Cai [6] ensures that this step does not significantly increase the complexity of the system.

The dynamics of A naturally decomposes into a rotational part, corresponding to eigenvalues of modulus one, and an expansive or contractive part, corresponding to eigenvalues of absolute value different from 1 and to generalised eigenvalues of arbitrary moduli. Accordingly, the ambient space $\mathbb{R}^n$ decomposes into two subspaces V_{rec} and $V_{\text{non-rec}}$, such that A

exhibits rotational behaviour on V_{rec} and expansive or contractive behaviour on $V_{\text{non-rec}}$. We start by considering the special cases where either $V_{\text{rec}} = 0$ or $V_{\text{non-rec}} = 0$, so that only one of the two types of behaviours occurs.

First, assume that A has no complex eigenvalues of modulus 1. Since every trajectory under A escapes K we have in particular that $0 \notin K$. A theorem due to Jeronimo, Perrucci and Tsigaridas [15] shows that K is bounded away from zero by a function of the form $2^{-(d\tau)^{n^{O(1)}}}$ and a theorem due to Vorobjov [23] establishes an upper bound on the absolute value of every coordinate of every point in K of the form $2^{(d\tau)^{n^{O(1)}}}$. Furthermore, thanks to a result of Mignotte [17], we can bound the eigenvalues of A away from 1 by a function of the form $2^{\tau^{n^{O(1)}}}$. This yields a doubly exponential bound on how long it takes for A to leave the set K (either by converging to 0 or by converging to infinity in some eigenspace).

Now assume that all eigenvalues of A have modulus 1. This case is handled through a combination of two bounds. For the first bound we start by noting that for every $x \in K$ the closure of the orbit $\overline{\mathcal{O}_A(x)}$ is a compact semialgebraic set that is not entirely contained within K. In fact we show that for all $x \in K$ there exists a point $y \in \overline{\mathcal{O}_A(x)}$ whose distance to K is at least $2^{-(d\tau)^{n^{O(1)}}}$. This bound is achieved by applying [15, Theorem 1] to a suitable polynomial on an auxiliary semialgebraic set, which is constructed using quantifier elimination. The singly exponential bounds obtained in [14, 20] are crucial for this step to work. The second step of the argument combines Baker's theorem on linear forms in logarithms with a quantitative version of Kronecker's theorem on simultaneous Diophantine approximation to obtain a bound of the form $N_P \in 2^{(\tau P)^{n^{O(1)}}}$ such that for all positive integers P every point $z \in \overline{\mathcal{O}_A(x)}$ is within 2^{-P} of a point of the form $A^t x$ with $0 \leq t \leq N_P$. Combining the two bounds described above, we obtain a doubly exponential bound on the escape time.

In the presence of both types of behaviour, the analysis of each case becomes more involved. We select a parameter $\varepsilon > 0$ and partition K into three sets: $K_{\text{rec}} = K \cap V_{\text{rec}}$, $K_{\geq\varepsilon}$, and $K_{<\varepsilon}$. The matrix A exhibits purely rotational behaviour on K_{rec}. Intuitively, on $K_{\geq\varepsilon}$ the expansive or contractive behaviour of A dominates the overall dynamics, while on $K_{<\varepsilon}$ the rotational behaviour dominates the overall dynamics. We establish in Lemma 14 a bound N_{rec} such that for each initial point $x \in V_{\text{rec}}$, one of its first N_{rec} iterates is bounded away from K. In Lemma 15 we establish a bound $N_{\geq\varepsilon}$ such that every $x \in K_{\geq\varepsilon}$ either escapes or enters $K_{<\varepsilon} \cup K_{\text{rec}}$ within at most $N_{\geq\varepsilon}$ iterations. Finally, in Section 7, we establish a bound on how often the system can switch from a state where rotational behaviour dominates to one where expansive or non-expansive behaviour does and vice versa. We use this to combine the two bounds to an overall bound on the escape time, proving Theorem 1.

Main Contributions. While decidability of CEP was already established in [18], the proof given there was non-effective, combining two unbounded searches. To obtain a uniform quantitative bound on the escape time, the argument given in [18] needs to be refined and extended in two significant ways:

Firstly, one needs to establish non-trivial quantitative refinements of the techniques used in the decidability proof: to bound the escape time for purely expanding or retracting systems, we need to combine the sharp effective bounds on compact semialgebraic sets from real algebraic geometry established in [23, 15] with Mignotte's root separation bound [17]. The case of purely rotational systems requires an original combination of a quantitative version of Kronecker's theorem on simultaneous Diophantine approximation [12] and a quantitative version of Baker's theorem on linear forms in logarithms [1]. All of these techniques were completely absent from the decidability proof.

Secondly, to establish mere decidability of the problem, it was possible to study the possible behaviours of the system – rotating, expanding, or retracting – in isolation. For example, if the set K contains a point which has a non-zero component in an eigenspace of A for an eigenvalue whose modulus is strictly greater than one, then the system must eventually escape. However, no uniform bound on the escape time may be derived in this situation, for the component is allowed to be arbitrarily close to zero. Therefore, as outlined above, it is necessary in our proof to subdivide K into pieces where rotational, retractive, and expansive behaviour can be present simultaneously. The interaction of the three behaviours significantly increases the difficulty of the analysis and requires completely new ideas.

2 Mathematical Tools

We use the following singly exponential quantifier elimination result given in [2]. For a historical overview on this type of result see [2, Chapter 14, Bibliographical Notes].

▶ **Theorem 2** ([2, Theorem 14.16]). *Let $S \subseteq \mathbb{R}^{k+n_1+\cdots+n_\ell}$ be a semialgebraic set of complexity at most $(k + n_1 + n_2 + \cdots + n_\ell, d, \tau)$. Let $Q_1, \ldots, Q_\ell \in \{\exists, \forall\}$ be a sequence of alternating quantifiers. Consider the set $S' \subseteq \mathbb{R}^k$ of all $(x_1, \ldots, x_k) \in \mathbb{R}^k$ satisfying the first-order formula*

$$(Q_1(x_{1,1}, \ldots, x_{1,n_1})) \ldots (Q_\ell(x_{\ell,1}, \ldots, x_{\ell,n_\ell})) . \\ ((x_1, \ldots, x_k, x_{1,1}, \ldots, x_{1,n_1}, \ldots, x_{\ell,1}, \ldots, x_{\ell,n_\ell}) \in S)$$

Then S' is a semialgebraic set of complexity at most $(k, d^{O(n_1\cdot\dots\cdot n_\ell)}, \tau d^{O(n_1\cdot\dots\cdot n_\ell\cdot k)})$.

The next theorem is due to Vorobjov [23]. See also [13, Lemma 9] and [3, Theorem 4].

▶ **Theorem 3.** *There exists an integer function* $\operatorname{Bound}(n, d, \tau) \in 2^{\tau d^{O(n)}}$ *with the following property:*

Let K be a compact semialgebraic set of complexity at most (n, d, τ). Then K is contained in a ball centred at the origin of radius at most $\operatorname{Bound}(n, d, \tau)$.

A closely related result, due to [15], yields a lower bound on the minimum of a polynomial over a compact semialgebraic set, provided the minimum is non-zero. The result in [15] mentions explicit constants, which is more than we need.

▶ **Theorem 4** ([15, Theorem 1]). *There exists an integer function* $\operatorname{LowerBound}(n, d, \tau) \in 2^{(\tau d)^{n^{O(1)}}}$ *such that the following holds true:*

Let $P \in \mathbb{Q}[x_1, \ldots, x_n]$ be a polynomial of degree at most d, whose coefficients have bitsize at most τ. Let K be a compact semialgebraic set of complexity at most (n, d, τ). If $\min_{x\in K} P(x) > 0$ *then* $\min_{x\in K} P(x) > 1/\operatorname{LowerBound}$.

With the help of Theorem 2, Theorem 4 can be generalised to yield a lower bound on the distance of two disjoint compact semialgebraic sets. A very similar result is proved in [21] under more general assumptions. Unfortunately, the complexity bound stated there is not sufficiently fine-grained for our purpose, since the author do not distinguish the dimension of a set from the other complexity parameters.

▶ **Lemma 5.** *There exists an integer function* $\operatorname{Sep}(n, d, \tau) \in 2^{(\tau d)^{n^{O(1)}}}$ *with the following property:*

Let K and L be compact semialgebraic sets of complexity at most (n, d, τ). Assume that every $x \in K$ has positive euclidean distance to L. Then $\inf_{x\in K} d(x, L) > 1/\operatorname{Sep}(n, d, \tau)$.

Proof. See [10, Appendix E]. ◀

We require a version of Kronecker's theorem on simultaneous Diophantine approximation. See [19, Corollary 3.1] for a proof.

▶ **Theorem 6.** *Let $(\lambda_1, \ldots, \lambda_m)$ be complex algebraic numbers of modulus 1. Consider the free Abelian group*

$$L = \{(n_1, \ldots, n_m) \in \mathbb{Z}^m \mid \lambda_1^{n_1} \cdot \cdots \cdot \lambda_m^{n_m} = 1\}.$$

Let $(\beta_1, \ldots, \beta_s)$ be a basis of L. Let $\mathbb{T}^m = \{(z_1, \ldots, z_m) \in \mathbb{C}^m \mid |z_j| = 1\}$ denote the complex unit m-torus. Then the closure of the set $\left\{(\lambda_1^k, \ldots, \lambda_m^k) \in \mathbb{T}^m \mid k \in \mathbb{N}\right\}$ is the set $S = \left\{(z_1, \ldots, z_m) \in \mathbb{T}^m \mid \forall j \leq s.(z_1, \ldots, z_m)^{\beta_j} = 1\right\}$.

Moreover, for all $\varepsilon > 0$ and all $(z_1, \ldots, z_m) \in S$ there exist infinitely many indexes k such that $|\lambda_j^k - z_j| < \varepsilon$ for $j = 1, \ldots, m$.

Moreover, the integer multiplicative relations between given complex algebraic numbers in the unit circle can be elicited in polynomial space. For a proof see [7, 16]. We assume the standard encoding of algebraic numbers, see [8] for details.

▶ **Theorem 7.** *Let $(\lambda_1, \ldots, \lambda_m)$ be complex algebraic numbers of modulus 1. Consider the free Abelian group*

$$L = \{(n_1, \ldots, n_m) \in \mathbb{Z}^m \mid \lambda_1^{n_1} \cdot \cdots \cdot \lambda_m^{n_m}\}.$$

Then one can compute in polynomial space a basis $(\beta_1, \ldots, \beta_s) \in (\mathbb{Z}^m)^s$ for L. Moreover, the integer entries of the basis elements β_j are bounded polynomially in the size of the encodings of $\lambda_1, \ldots, \lambda_m$ and singly exponentially in m.

We need to be able to bound away the modulus of eigenvalues that fall outside the unit circle away from 1. This is achieved by combining a classic result due to Mignotte [17] on the separation of algebraic numbers with a bound on the height of the resultant of two polynomials, proved in [4, Theorem 10].

▶ **Lemma 8.** *Let λ be a complex algebraic number whose minimal polynomial has degree at most d and coefficients bounded in bitsize by τ. Assume that $|\lambda| \neq 1$. Then we have $||\lambda| - 1| > 2^{-(\tau d)^{O(1)}}$.*

Proof. See [10, Appendix C]. ◀

3 Preliminaries

3.1 Converting the matrix to real Jordan normal form

To obtain a bound on the escape time it will be important to work with instances of the Escape Problem in real Jordan normal form. In the following, let $\mathbb{A}$ denote the field of algebraic numbers. We establish the following reduction to this case:

▶ **Lemma 9.** *Let (K, A) be an instance of the Compact Escape Problem. Assume that K is given by a formula involving s polynomial equations and equalities $P \bowtie 0$ where $P \in \mathbb{Z}[x_1, \ldots, x_n]$ is a polynomial in n variables of degree at most d whose coefficients are bounded in bitsize by τ.*

Let $\gamma_1, \ldots, \gamma_m \in \mathbb{R}$ denote the real and imaginary parts of the eigenvalues of A. Let δ be a bound on the degrees of $\gamma_1, \ldots, \gamma_m$.

Then there exists an equivalent instance (J, K') of the Compact Escape Problem where $J \in \mathbb{A}^{(n+m)\times(n+m)}$ is in real Jordan normal form and K' is given by a formula involving at most $s + 3m$ polynomial equations and equalities $P \bowtie 0$ where $P \in \mathbb{Z}[x_1, \dots, x_{n+m}]$ is a polynomial in $n + m$ variables of degree at most $\delta \cdot d$ whose coefficients are bounded in bitsize by $\tau + d(\log(2n) + \log(\delta + 1) + \sigma)$, where σ depends polynomially on n and the bitsize of the entries of A.

Proof. See [10, Appendix B]. ◀

3.2 Decomposing K

Let $K \subseteq \mathbb{R}^n$ be a compact semialgebraic set. Let $A \in \mathbb{R}^{n\times n}$ be a matrix in real Jordan normal form,

$$A = \begin{pmatrix} J_1 & & \\ & \ddots & \\ & & J_m \end{pmatrix}.$$

Here, each J_i is a real Jordan block of the form

$$J_i = \begin{pmatrix} \Lambda_i & I_i & & \\ & \ddots & \ddots & \\ & & \Lambda_i & I_i \\ & & & \Lambda_i \end{pmatrix},$$

where $\Lambda_{i,1}$ is either a real number or a 2×2 real matrix of the form $\begin{pmatrix} a_i & -b_i \\ b_i & a_i \end{pmatrix}$ and, accordingly, I_i is either the real number 1 or the 2×2 identity matrix. The elements Λ_i correspond to real or complex eigenvalues $\lambda_i \in \mathbb{C}$ of A. By slight abuse of language we call $|\lambda_i|$ the modulus of Λ_i. By further slight abuse of language we define the "eigenspace" of Λ_i as the one- or two-dimensional space spanned by the vectors that correspond to the first entry of the Jordan block J_i. The "generalised eigenspaces" for Λ_i are defined analogously.

Write $\mathbb{R}^n$ as the direct sum of two spaces $\mathbb{R}^n = V_{\text{rec}} \oplus V_{\text{non-rec}}$ where V_{rec} is the direct sum of the eigenspaces for eigenvalues of modulus 1, and $V_{\text{non-rec}}$ is the direct sum of the eigenspaces and generalised eigenspaces for eigenvalues of modulus $\neq 1$ and the generalised eigenspaces for eigenvalues of modulus 1. By convention, if A has no eigenvalues of modulus 1 we let $V_{\text{rec}} = 0$. Similarly, if A has only eigenvalues of modulus 1 and no generalised eigenvalues we let $V_{\text{non-rec}} = 0$. Thus, we decompose the state space $\mathbb{R}^n$ into a part V_{rec} on which A exhibits purely rotational behaviour, and a part $V_{\text{non-rec}}$ where A is additionally expansive or contractive.

We will work with several different norms throughout this paper. In addition to the familiar ℓ^2 and ℓ^∞ norms we introduce a third norm, depending on the matrix A, that combines features of the two. It facilitates block-wise arguments while ensuring that the restriction of A to V_{rec} is an isometry.

Write $\mathbb{R}^n$ as a direct sum $\mathbb{R}^n = V_1 \oplus \cdots \oplus V_s \oplus W_1 \oplus \cdots \oplus W_t$, where $V_1, \dots, V_s$ correspond to the Jordan blocks of A associated with real eigenvalues and $W_1, \dots, W_t$ correspond to the Jordan blocks of A associated with non-real eigenvalues. Let $\pi_{W_j} : \mathbb{R}^n \to W_j$ and $\pi_{V_j} : \mathbb{R}^n \to V_j$ denote the orthogonal projections onto W_j and V_j respectively.

For a vector $x \in V_i$, let $\|x\|_J^{V_i} = \|x\|_\infty$. For a vector $x = (x_1, y_1, \dots, x_k, y_k) \in W_i$, let

$$\|x\|_J^{W_i} = \max_{j=1,\dots,k} \left(\sqrt{x_j^2 + y_j^2}\right).$$

For a vector $x \in \mathbb{R}^n$, let

$$\|x\|_J = \max\left\{ \max_{j=1,\dots,s} \left\|\pi_{V_j}(x)\right\|_J^{V_j}, \max_{j=1,\dots,t} \left\|\pi_{W_j}(x)\right\|_J^{W_j} \right\}.$$

Call $\|x\|_J$ the Jordan norm of x. Observe that $\|x\|_J$ depends on the choice of the V_i's and W_i's. The Jordan norm compares to the ℓ^2- and ℓ^∞- norms as follows:

$$n^{-1/2}\|x\|_J \leq n^{-1/2}\|x\|_2 \leq \|x\|_\infty \leq \|x\|_J \leq \|x\|_2 \leq n^{1/2}\|x\|_\infty \leq n^{1/2}\|x\|_J .$$

Let $\varepsilon > 0$. Consider the ball $B_J(0,\varepsilon) \subseteq \mathbb{R}^n$ about 0 with respect to the distance induced by the $\|\cdot\|_J$-norm. We partition K into three sets:

$$\begin{aligned} &K_{\text{rec}} = K \cap V_{\text{rec}} \\ &K_{<\varepsilon} = K \cap (V_{\text{rec}} \oplus ((V_{\text{non-rec}} \cap B_J(0,\varepsilon)) \setminus \{0\})) \\ &K_{\geq\varepsilon} = K \cap (V_{\text{rec}} \oplus (V_{\text{non-rec}} \setminus B_J(0,\varepsilon))) \end{aligned}$$

4 A quantitative version of Kronecker's theorem for complex algebraic numbers

Our central tool for bounding the escape time in the recurrent case is a quantitative version of Kronecker's theorem for complex algebraic numbers.

Let $(\lambda_1,\dots,\lambda_m)$ be complex algebraic numbers of modulus 1. Our goal is to find for all $\varepsilon > 0$ a bound N such that for all $(\alpha_1,\dots,\alpha_m) \in \mathbb{T}^m$ contained in the closure of the sequence $(\lambda_1^t,\dots,\lambda_m^t)_{t\in\mathbb{N}}$ there exists $t \leq N$ such that $|\lambda_j^t - \alpha_j| < \varepsilon$ for all $j = 1,\dots,m$.

We first consider the case where the λ_j's do not admit any integer multiplicative relations. In this case we can employ the following quantitative version of the continuous formulation of Kronecker's theorem, proved in [12]:

▶ **Theorem 10** ([12, Theorem 4.1]). *Let $\varphi_1,\dots,\varphi_N$ and $\zeta_1,\dots,\zeta_N$ be real numbers. Let $\varepsilon_1,\dots,\varepsilon_N$ be positive real numbers with $\varepsilon_j < 1/2$ for all j. Let $M_j = \left\lceil \frac{1}{\varepsilon_j}\log\frac{N}{\varepsilon_j}\right\rceil$. Let $\varphi = (\varphi_1,\dots,\varphi_N)$. Let $\delta = \min\left\{|\varphi\cdot m| \mid m \in \mathbb{Z}^N, |m_j| < M_j, m \neq 0\right\}$. Assume that $\delta > 0$. Then in any interval I of length $T \geq 4/\delta$ there is a real number t such that $\|\varphi_j t - \zeta_j\| < \varepsilon_j$, where $\|\cdot\|$ denotes distance to the nearest integer.*

Intuitively, the number δ in Theorem 10 is a quantitative measure of the linear independence of the φ_j's, as it bounds away from zero all integer linear combinations of the φ_j's with suitably bounded coefficients. In our case we consider the numbers $\varphi_j = \log\lambda_j$. For our purpose we need to obtain a bound on t, and thus a bound on δ, in terms of the algebraic complexity of the numbers $\lambda_1,\dots,\lambda_m$. This is achieved by invoking a quantitative version of Baker's theorem on linear forms in logarithms due to Baker and Wüstholz [1]. Recall that any algebraic number μ is the root of a unique irreducible polynomial p_μ with pairwise coprime integer coefficients. The *height* of an algebraic number μ is the maximum of the absolute values of the coefficients of p_μ. The *degree* of μ is the degree of p_μ. Recall that a field E is called an *extension* of a field F if E contains F as a subfield. The *degree* of a field extension $E \supseteq F$ is the dimension of E as an F-vector space.

▶ **Theorem 11.** *Let $\mu_1, \ldots, \mu_N$ be algebraic numbers, none of which is equal to 0 or 1. Let*

$$L(z_1, \ldots, z_N) = b_1 z_1 + \cdots + b_N z_N$$

be a linear form with rational integer coefficients $b_1, \ldots, b_N$. Let B be an upper bound on the absolute values of the b_j's. For $j = 1, \ldots, N$, let $A_j \geq \exp(1)$ be a bound on the height of μ_j. Let d be the degree of the field extension $\mathbb{Q}(\mu_1, \dot{;}\mu_N)$ generated by $\mu_1, \ldots, \mu_N$ over $\mathbb{Q}$. Fix a determination of the complex logarithm $\log$. Let $\Lambda = L(\log \mu_1, \ldots, \log \mu_N)$. If $\Lambda \neq 0$ then

$$\log |\Lambda| > -(16Nd)^{2(N+2)} \log A_1 \cdot \cdots \cdot \log A_N \log B.$$

Finally, in the case where the λ_j's admit integer multiplicative relations, we employ Theorem 7 to bound their complexity. We arrive at the following result:

▶ **Theorem 12.** *Let $(\lambda_1, \ldots, \lambda_m)$ be complex algebraic numbers of modulus 1. Assume that the numbers $2\pi i, \log \lambda_1, \ldots, \log \lambda_s$ are linearly independent over the rationals, where $0 \leq s \leq m$. Let d be the degree of the field extension $\mathbb{Q}(\lambda_1, \ldots, \lambda_s)$. Let $A_1, \ldots, A_s \geq \exp(1)$ be upper bounds on the heights of $\lambda_1, \ldots, \lambda_s$. Let $\ell \in \mathbb{N}$, and $\varepsilon_{s+1}, \ldots, \varepsilon_m \in \mathbb{Z}^s$ be such that*

$$\lambda_j^\ell = (\lambda_1, \ldots, \lambda_s)^{\varepsilon_j}$$

for all $j = s+1, \ldots, m$. By convention, if $s = 0$ the right-hand side of the above equation is to be taken equal to 1.

Let

$$L = \max \left\{ \ell, \sum_{k=1}^{s} |\varepsilon_{s+1,k}|, \ldots, \sum_{k=1}^{s} |\varepsilon_{m,k}| \right\}.$$

Let $\alpha_1, \ldots, \alpha_m \in \mathbb{T}^m$ be such that any rational linear relation between the numbers $2\pi i, \log \lambda_1, \ldots, \log \lambda_m$ is also satisfied by the numbers $2\pi i, \log \alpha_1, \ldots, \log \alpha_m$. Let $\varepsilon > 0$. Then there exists a positive integer

$$t \leq 8\pi\ell \left(\tfrac{2\pi L}{\varepsilon}\right)^s \left(2s \tfrac{2\pi L}{\varepsilon} \left\lceil \tfrac{4\pi L}{\varepsilon} \log \tfrac{4\pi s L}{\varepsilon} \right\rceil\right)^{(16(s+1)d)^{2(s+3)} \log A_1 \cdot \cdots \cdot \log A_s} + \ell$$

such that $\left|\lambda_j^t - \alpha_j\right| < \varepsilon$ for $j = 1, \ldots, m$.

Proof. An outline of the proof is sketched above. See [10, Appendix D] for a full proof. ◀

For the purpose of bounding the escape time, the following coarse bound suffices:

▶ **Corollary 13.** *There exists an integer function $\mathrm{Kron}(n, \tau, P) \in 2^{(\tau P)^{n^{O(1)}}}$, such that the following holds true:*

Let $\lambda_1, \ldots, \lambda_n$ be algebraic numbers of modulus 1. Assume that the degree of each λ_j is bounded by n. Let τ be a bound on the bitsize of the coefficients of the minimal polynomials of the λ_j's. Let P be a positive integer. Let $\alpha_1, \ldots, \alpha_n$ be complex numbers which are contained in the closure of the sequence $(\lambda_1^t, \ldots, \lambda_n^t)_{t \in \mathbb{N}}$. Then there exists a $t \leq \mathrm{Kron}(n, \tau, P)$ such that $|\alpha_j - \lambda_j^t| < 2^{-P}$ for all $j \in \{1, \ldots, n\}$.

Proof. By Kronecker's theorem, any integer multiplicative relation between the λ_j's is also satisfied by the α_j's. Theorem 12 hence yields a bound on t such that $|\alpha_j - \lambda_j^t| < 2^{-P}$ holds for all $j \in \{1, \ldots, n\}$.

This bound is given in terms of quantities s, d, ℓ, $\varepsilon_{s+1}, \ldots, \varepsilon_m \in \mathbb{Z}^s$, $A_1, \ldots, A_s$, and L. It remains to show that these quantities can be chosen to be suitably bounded in terms of n and τ.

Proposition 26 in [10, Appendix D], which is mainly based on Theorem 7, shows that numbers ℓ and $\varepsilon_1, \dots, \varepsilon_m$ can be computed in polynomial space. In particular, the absolute size of L and ℓ is of the form $2^{(n\tau)^{O(1)}}$. The numbers $\log A_i$ are bounded by τ by assumption. We have $s \leq m \leq n$ by definition. Finally, we have assumed that each λ_j has degree at most n. It follows that the degree d of the field extension $\mathbb{Q}(\lambda_1, \dots, \lambda_s)$ is bounded by n^n. The result follows from Theorem 12. ◀

5 The recurrent eigenspace

The next lemma establishes as a special case an escape bound for all initial values $x \in K_{\mathrm{rec}}$. In order to combine the recurrent and the non-recurrent case we need a stronger result, however. Thus, we establish not only a bound on the escape time for all initial values $x \in K_{\mathrm{rec}}$, but a bound N such that every $x \in V_{\mathrm{rec}}$ – not just in K_{rec} – has distance at least $1/N$ – not just positive distance – from K. Further, note that Lemma 14 is still applicable in the special cases where $K_{\mathrm{rec}} = \emptyset$ or $V_{\mathrm{rec}} = 0$.

▶ **Lemma 14.** *There exists an integer function* $\mathrm{Rec}(n, d, \tau) \in 2^{(\tau d)^{n^{O(1)}}}$ *with the following property:*

Let $A \in \mathbb{A}^{n \times n}$ *be a matrix in real Jordan normal form with algebraic entries. Assume that the minimal polynomial of* A *has rational coefficients whose bitsize is bounded by* τ*. Let* $K \subseteq \mathbb{R}^n$ *be a compact semialgebraic set of complexity at most* (n, d, τ)*. If every point* $x \in K_{\mathrm{rec}}$ *escapes* K *under iterations of* A *then for all* $x \in V_{\mathrm{rec}}$ *there exists* $t \leq \mathrm{Rec}(n, d, \tau)$ *such that*

$$\mathrm{dist}_{\ell^2}(A^t x, K) > \frac{\sqrt{n}}{\mathrm{Rec}(n, d, \tau)}.$$

Proof. The full proof is given in [10, Appendix F]. We only sketch an outline here.

We first prove the result for initial points $x \in K_{\mathrm{rec}}$. For these points, the closure of the orbit $\overline{\mathcal{O}_A(x)}$ of x under A is a compact semialgebraic set. We employ Corollary 13 to obtain for all $\varepsilon > 0$ a doubly exponential bound N such that for all $x \in K_{\mathrm{rec}}$ and all $y \in \overline{\mathcal{O}_A(x)}$ there exists $t \leq N$ such that $\|A^t x - y\|_2 < \varepsilon$. We then use Theorem 4 to obtain a uniform at most doubly exponentially small lower bound on the quantity

$$\inf_{x \in K_{\mathrm{rec}}} \sup_{y \in \overline{\mathcal{O}_A(x)}} \inf_{z \in K} \|y - z\|_2^2 .$$

In order to apply this theorem we construct an auxiliary semialgebraic set, whose complexity is controlled by Theorem 2. Combining these two steps, we obtain a function Rec_0 that satisfies the statement of the lemma for all initial points $x \in K_{\mathrm{rec}}$.

Finally, we extend the result to all initial points $x \in V_{\mathrm{rec}}$. The special case where $K_{\mathrm{rec}} = \emptyset$ is treated using Theorem 4.

In the case where K_{rec} is non-empty we obtain from Lemma 5 that every $x \in V_{\mathrm{rec}}$ which is doubly exponentially close to K with a sufficiently large constant in the third exponent is already doubly exponentially close to K_{rec}, with a slightly smaller constant in the third exponent. Now, any point that is sufficiently far away from K trivially satisfies the claim. By the preceding discussion, points $x \in V_{\mathrm{rec}}$ that are sufficiently close to K are already sufficiently close to K_{rec}, so that there exists an escaping orbit $\overline{\mathcal{O}_A(x')}$ with $x' \in K_{\mathrm{rec}}$ which is close to the orbit of x since A is an isometry on V_{rec}. This allows us to reduce the result to the already established result for initial values in K_{rec}. ◀

6 The non-recurrent eigenspace

The next lemma concerns the subset $K_{\geq\varepsilon}$ of K containing the points in K that are bounded away from V_{rec} by some $\varepsilon > 0$.

For any such point, there exist coordinates (or pairs of coordinates if the corresponding eigenvalues are not real) whose contribution to the Jordan norm is greater than ε. Moreover, the contribution to the Jordan norm of these coordinates does not stay constant under applications of A. If the contribution to the norm of at least one such coordinate is increasing under applications of A, the orbit will eventually leave K, since K is compact. Moreover, Theorem 3 yields an upper bound on the escape time.

Coordinates whose contribution to the norm is decreasing under applications of A will, after sufficiently many iterations, contribute less than ε. We establish a uniform upper bound on the number of iterations required to ensure this for all such coordinates. Combining this with the previous bound, we obtain a number N such that after at most N applications of A, every $x \in K_{\geq\varepsilon}$ has either escaped K, entered $K_{<\varepsilon} \cup K_{\text{rec}}$, or it remains in $K_{\geq\varepsilon}$ because it has a component whose contribution to the norm was initially smaller than ε, but grew beyond ε under iteration of A. In the last case, the point will grow in norm beyond the bound established in Theorem 3 and thus escape K after a further N applications of A. This yields a uniform bound on the number of iterations that are required for any point $x \in K_{\geq\varepsilon}$ to either leave K entirely or move into $K_{<\varepsilon} \cup K_{\text{rec}}$.

The overall structure of this proof closely follows the one given in [11], where the assumptions allow the authors to restrict the discussion to real eigenvalues.

▶ **Lemma 15.** *There exists an integer function* $\text{NonRec}(n, d, \tau, P) \in 2^{(d\tau P)^{n^{O(1)}}}$ *with the following property:*

Let K be a compact semialgebraic set of complexity at most (n, d, τ). Let $A \in \mathbb{A}^{n\times n}$ be a matrix in real Jordan normal form. Assume that the characteristic polynomial of A has rational coefficients whose bitsize is bounded by τ. Let P be a positive integer.

Then for all $x \in K_{\geq 2^{-P}}$ there exists $t \leq \text{NonRec}(n, d, \tau, P)$ such that $A^t x \notin K_{\geq 2^{-P}}$.

Proof. See [10, Appendix G] for details. ◀

7 Proof of Theorem 1

In the previous two sections, we successively showed how to establish a bound on the escape time for an instance (A, K) when the orbit remains in the recurrent eigenspace and how the orbit behaves when it starts away from the recurrent eigenspace. In this section, we show how to combine both results in order to establish an escape bound for any starting point in K. This will thus prove Theorem 1.

Let (A_0, K_0) be an instance of the compact escape problem, where $K_0 \subseteq \mathbb{R}^n$ is a compact semialgebraic set of complexity at most (n_0, d_0, τ_0) and $A_0 \in \mathbb{Q}^{n\times n}$ is a square matrix with rational entries whose bitsize is bounded by τ_0. Assume that every point $x \in K_0$ escapes K_0 under iterations of A_0.

Apply Lemma 9 to convert the instance (A_0, K_0) into an equivalent instance (A, K) such that $A \in \mathbb{A}^{n\times n}$ is in real Jordan normal form. Then the set K has complexity at most (n, d, τ), were $n = 2n_0$, $d = n_0 d_0$, and $\tau = (n_0\tau_0 d_0)^{C_\tau}$ for some absolute constant C_τ. By construction, the characteristic polynomial of A has rational coefficients of bitsize at most τ.

Let Rec be the function from Lemma 14. Let $\varepsilon = \frac{1}{\text{Rec}(n,d,\tau)}$ and $N_{\text{rec}} = \text{Rec}(n, d, \tau)$. Let $x \in K$. If $x \in K_{\text{rec}}$ then x escapes within N_{rec} steps. Suppose that $x \in K_{<\varepsilon}$.

Then there are two possibilities:

1. We have $A^t x \notin K_{\geq\varepsilon}$ for all $t \leq N_{\text{rec}}$.
2. We have $A^t x \in K_{\geq\varepsilon}$ for at least one $t \leq N_{\text{rec}}$.

In the first case, the orbit of x remains close to V_{rec} for long enough that we can rely on Lemma 14. Indeed, let x_0 denote the orthogonal projection of x onto V_{rec}. Let $t \leq N_{\text{rec}}$ be such that $\text{dist}_{\ell^2}(A^t x_0, K) > \sqrt{n}\varepsilon$. Since $A^t x \notin K_{\geq\varepsilon}$, we have $\|A^t x - A^t x_0\|_J < \varepsilon$, so that $\|A^t x - A^t x_0\|_2 < \sqrt{n}\varepsilon$. Let $y \in K$. Then

$$\left\|A^t x - y\right\|_2 \geq \left\|A^t x_0 - y\right\|_2 - \left\|A^t x - A^t x_0\right\|_2 > \sqrt{n}\varepsilon - \sqrt{n}\varepsilon = 0.$$

Thus, x escapes K under iterations of A.

In the second case, let t_1 be such that $A^{t_1} x \in K_{\geq\varepsilon}$. Let NonRec be the function from Lemma 15. Let $N_{\geq\varepsilon} = \text{NonRec}(n, d, \tau, \lceil\log(1/\varepsilon)\rceil)$. By Lemma 15 there exists $t_2 \leq N_{\geq\varepsilon}$ such that $A^{t_2} A^{t_1} x$ is contained either in $K_{<\varepsilon} \cup K_{\text{rec}}$ or in the complement of K. In the latter case we are done. In the former case we apply the initial case distinction: either for all $t \leq N_{\text{rec}}$ we have $A^t A^{t_2} A^{t_1} x \notin K_{\geq\varepsilon}$ or we have $A^{t_3} A^{t_2} A^{t_1} x \in K_{\geq\varepsilon}$ for at least one $t_3 \leq N_{\text{rec}}$. Once again, in the first case, the point has escaped. By repeating this reasoning, we construct a (finite or infinite) sequence $t_1, t_2, \dots$ such that $t_i \leq N_{\text{rec}}$ if i is odd and $t_i \leq N_{\geq\varepsilon}$ if i is even and

$$A^{t_s} \cdot \dots \cdot A^{t_1} x \in \begin{cases} K_{<\varepsilon} \cup K_{\text{rec}} & \text{if } s \text{ is even,} \\ K_{\geq\varepsilon} & \text{if } s \text{ is odd.} \end{cases}$$

We claim that the sequence $t_1, t_2, \dots$ is finite and contains at most n^3 elements.

Consider a real Jordan block of A of size $m \leq n$ associated to the eigenvalue Λ. Denote by x_J the orthogonal projection of x onto the dimensions associated with this block.

Assume first that Λ is a real eigenvalue (as opposed to a 2×2 block representing a complex eigenvalue). If $\Lambda = 0$, then clearly $\left\|J^k x_J\right\|_J$ is monotonically decreasing. Thus, assume in the sequel that $\Lambda \neq 0$.

Let $j \in \{1, \dots, m\}$. The $m - j + 1$'th component of the vector $J^k x_J$, viewed as a function of t, is an exponential polynomial $E_j(t) = \Lambda^t P(t)$, where $P \in \mathbb{R}[z]$ is a real polynomial of degree $j - 1$. Consider the real function

$$(E_j(\cdot))^2 \colon \mathbb{R} \to \mathbb{R},\ (E_j(t))^2 = |\Lambda|^{2t} |P(t)|^2.$$

This function is differentiable in t with derivative

$$\tfrac{d}{dt}(E_j(t))^2 = \Lambda^{2t} \left(\log(\Lambda^2)(P(t)^2) + 2P(t)P'(t)\right).$$

This derivative vanishes if and only if the factor $\left(\log(\Lambda^2)(P(t)^2) + 2P(t)P'(t)\right)$ vanishes. This factor is a polynomial of degree $2j - 2$, so that it has at most $2j - 2$ real zeroes. It follows that there exist numbers $t_{j,1}, \dots, t_{j,m_j}$ with $m_j \leq 2j - 2$ such that the function $(E_j(t))^2 - \varepsilon^2$ does not change its sign in any of the open intervals

$$(0, t_{j,1}), (t_{j,1}, t_{j,2}), \dots, (t_{j,m_j-1}, t_{j,m_j}), (t_{j,m_j}, +\infty).$$

Thus, the norm $\|J^t x_J\|_J$ changes from smaller than ε to bigger than ε at most

$$\sum_{j=1}^{m}(2j - 2) = 2\sum_{j=1}^{m} j - 2m = (m+1)m - 2m = m^2 - m$$

times.

The case where Λ represents a complex eigenvalue λ is similar. However, we now consider the evolution of the two coordinates corresponding to one Λ-block simultaneously.

For $j \in \{1, \dots, m\}$, write $E_j(t)$ for the $m-j+1$'th component of the vector $J^t x_J$, viewed as a function of t. We have for all $j \in \{1, \dots, m/2\}$ that the function

$$F_j(t) = (E_{2j}(t))^2 + (E_{2j-1}(t))^2$$

is an exponential polynomial $F_j(t) = |\lambda|^t P_j(t)$, where $P_j \in \mathbb{R}[z]$ is a real polynomial of degree $j-1$. Therefore, exactly as in case where Λ is a real eigenvalue, the derivative of F_j vanishes at most $2j-2$ times. From which we can deduce that the norm $\|J^t x_J\|_J$ crosses the ε-threshold at most $m^2 - m$ times.

Estimating generously, we have at most n Jordan blocks of size at most n, each of which crosses the ε-threshold at most $n^2 - n$ times. In total, we cross the threshold at most $n^3 - n^2$ times. The total escape bound is hence $n^3 \max\{N_{\mathrm{rec}}, N_{\geq\varepsilon}\}$. By the same argument, the same escape bound holds true when the initial point x lies in $K_{\geq\varepsilon}$.

Substituting the constants N_{rec}, $N_{\geq\varepsilon}$, n, d, and τ with their definitions, we obtain the upper bound

$$\begin{aligned}&\mathrm{CompactEscape}(n_0, d_0, \tau_0) = \\ &\quad (2n_0)^3 \max\Big\{ \mathrm{Rec}\left(2n_0, n_0 d_0, (n_0 d_0 \tau_0)^{C_\tau}\right), \\ &\qquad\qquad \mathrm{NonRec}\left(2n_0, n_0 d_0, (n_0 d_0 \tau_0)^{C_\tau}, \log\left\lceil \mathrm{Rec}\left(2n_0, n_0 d_0, (n_0 d_0 \tau_0)^{C_\tau}\right)\right\rceil\right) \Big\}.\end{aligned}$$

One easily verifies that $\mathrm{CompactEscape}(n, d, \tau) \in 2^{(d\tau)^{n^{O(1)}}}$ as claimed.

8 A matching lower bound on escape time

In Theorem 1 we established a uniform upper bound on the escape time for all positive instances of the Compact Escape Problem. Our bound is doubly exponential in the ambient dimension and singly exponential in the rest of the data. We will now show that this bound cannot be significantly improved by showing that a doubly exponential bound cannot be avoided even for purely rotational systems. A second example displaying a doubly exponential lower bound is presented in [10, Appendix H].

▶ **Example 16.** For $(n, d, \tau) \in \mathbb{N}^3$, let $K_{(n,d,\tau)} \subseteq \mathbb{R}^{n+2}$ be the set of all points $(x, y, u_1, \dots, u_n)$ satisfying the (in)equalities: $x^2 + y^2 = 1, u_1 = 2^{-\tau}$, $(x-1)^2 + y^2 \geq u_n$ and for $1 \leq i \leq n-1$, $u_{i+1} = (u_i)^d$.

Hence, $K_{(n,d,\tau)} = \left(S^1 \setminus B\left((1,0), 2^{-\tau d^{n-1}}\right)\right) \times \left\{\left(2^{-\tau}, 2^{-\tau d}, \dots, 2^{-\tau d^{n-1}}\right)\right\}$, where $S^1 \subseteq \mathbb{R}^2$ is the unit circle. Let $a = \frac{3}{5}$, $b = \frac{4}{5}$. Let

$$A_{(n,d,\tau)} = \begin{pmatrix} a & -b & 0 \\ b & a & 0 \\ 0 & 0 & I_n \end{pmatrix}$$

where I_n is the $n \times n$- identity matrix. It is easy to see that the complex number $\frac{3}{5} + i\frac{4}{5}$ has modulus 1 and is not a root of unity. It follows from Dirichlet's theorem on simultaneous Diophantine approximation that the orbit of A is equal to $S^1 \times \left\{\left(2^{-\tau}, 2^{-\tau d}, \dots, 2^{-\tau d^{n-1}}\right)\right\}$, so that every initial point escapes under A.

We claim that there exists a point $x \in K_{(n,d,\tau)}$ that requires $2^{\tau d^{n-1}}$ steps to escape. Indeed, let $x_0 \in K_{(n,d,\tau)}$ be an arbitrary initial point. Consider the orbit $x_t = A^t x_0$. Let $N < 2^{\tau d^{n-1}}$. By the pigeonhole principle, the finite set of points $x_0, \dots, x_N$ contains at least one consecutive pair of points x_i, x_j on the circle such that the points x_i and x_j are joined by an arc of the circle of length strictly greater than $2/N$. It follows that we can ensure that none of the points $x_1, \dots, x_N$ is outside of $K_{(n,d,\tau)}$ by applying a suitable planar rotation to all points. Since all planar rotations commute, there exists for each angle θ an initial point $x_\theta \in S^1 \times \left\{\left(2^{-\tau}, 2^{-\tau d}, \ldots, 2^{-\tau d^{n-1}}\right)\right\}$, such that the orbit of x_θ under A is equal to the orbit of x_0 under A rotated by θ. This proves the claim.

References

1 Alan Baker and Gisbert Wüstholz. Logarithmic forms and group varieties. *Journal für die reine und angewandte Mathematik*, 442:19–62, 1993.

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

3 Saugata Basu and Marie-Françoise Roy. Bounding the radii of balls meeting every connected component of semi-algebraic sets. *Journal of Symbolic Computation*, 45(12):1270–1279, 2010.

4 Yuval Bistritz and Alexander Lifshitz. Bounds for resultants of univariate and bivariate polynomials. *Linear Algebra and its Applications*, 432(8):1995–2005, 2010.

5 Mark Braverman. Termination of integer linear programs. In *CAV'06*, volume 4144 of *LNCS*. Springer, 2006.

6 Jin-Yi Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *International Journal of Foundations of Computer Science*, 5(3/4):293–302, 1994. doi:10.1142/S0129054194000165.

7 Jin-Yi Cai, Richard J. Lipton, and Yechezkel Zalcstein. The complexity of the A B C problem. *J. Computing*, 29(6), 2000.

8 Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.

9 Julian D'Costa, Engel Lefaucheux, Eike Neumann, Joël Ouaknine, and James Worrell. On the Complexity of the Escape Problem for Linear Dynamical Systems over Compact Semialgebraic Sets. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.33.

10 Julian D'Costa, Engel Lefaucheux, Eike Neumann, Joël Ouaknine, and James Worrell. Bounding the Escape Time of a Linear Dynamical System over a Compact Semialgebraic Set, 2022. arXiv:2207.01550.

11 Julian D'Costa, Engel Lefaucheux, Joël Ouaknine, and James Worrell. How fast can you escape a compact polytope? In *STACS'20*, volume 154 of *LIPIcs*, pages 49:1–49:11, 2020.

12 Steven M. Gonek and Hugh L. Montgomery. Kronecker's approximation theorem. *Indagationes Mathematicae*, 27(2):506–523, 2016.

13 Dima Grigoriev and Nicolai Vorobjov. Solving systems of polynomial inequalities in subexponential time. *J. Symbolic Computation*, 5:37–64, 1988.

14 Joos Heintz, Marie-Françoise Roy, and Pablo Solernó. Sur la complexité du princie de Tarski-Seidenberg. *Bull. Soc. Math. France*, 118(1):101–126, 1990.

15 Gabriella Jeronimo, Daniel Perrucci, and Elias Tsigaridas. On the minimum of a polynomial function on a basic closed semialgebraic set and applications. *SIAM Journal on Optimization*, 23(1):241–255, 2013.

16 David W. Masser. *Linear relations on algebraic groups*, pages 248–262. Cambridge University Press, 1988.

17 M. Mignotte. Some useful bounds. In *Computer Algebra*, 1982.

18 Eike Neumann, Joël Ouaknine, and James Worrell. On ranking function synthesis and termination for polynomial programs. In *CONCUR'20*, volume 171, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

19 Joël Ouaknine and James Worrell. *Positivity Problems for Low-Order Linear Recurrence Sequences*, pages 366–379. Society for Industrial and Applied Mathematics, USA, 2014.

20 James Renegar. On the computational complexity and geometry of the first-order theory of the reals. i-iii. *J. Symbolic Computation*, 13(3):255–352, 1992.

21 Marcus Schaefer and Daniel Štefankovič. Fixed Points, Nash Equilibria, and the Existential Theory of the Reals. *Theory Computing Systems*, 60(2):172–193, 2017.

22 Ashish Tiwari. Termination of linear programs. In *CAV'04*, volume 3114 of *LNCS*. Springer, 2004.

23 Nicolai Vorobjov. Bounds of real roots of a system of algebraic equations. *Zap. Nauchn. Sem. LOMI*, 137:7–19, 1984. (in Russian).

The Pseudo-Reachability Problem for Diagonalisable Linear Dynamical Systems

Julian D'Costa
Department of Computer Science, University of Oxford, UK

Toghrul Karimov
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

Rupak Majumdar
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

Mahmoud Salamati
Max Planck Institute for Software Systems, Kaiserslautern, Germany

James Worrell
Department of Computer Science, University of Oxford, UK

Abstract

We study fundamental reachability problems on pseudo-orbits of linear dynamical systems. Pseudo-orbits can be viewed as a model of computation with limited precision and pseudo-reachability can be thought of as a robust version of classical reachability. Using an approach based on *o*-minimality of $\mathbb{R}_{\exp}$ we prove decidability of the discrete-time pseudo-reachability problem with arbitrary semialgebraic targets for diagonalisable linear dynamical systems. We also show that our method can be used to reduce the continuous-time pseudo-reachability problem to the (classical) time-bounded reachability problem, which is known to be conditionally decidable.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases pseudo-orbits, Orbit problem, Skolem problem, linear dynamical systems, reachability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.40

Related Version *Full Version*: `https://arxiv.org/abs/2204.12253`

Funding *Julian D'Costa*: emmy.network foundation under the aegis of the Fondation de Luxembourg.
Rupak Majumdar: DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
Joël Ouaknine: DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`). Joël Ouaknine is also affiliated with Keble College, Oxford as `emmy.network` Fellow.

1 Introduction

A *discrete-time linear dynamical system* (LDS) is given by an update matrix $M \in \mathbb{Q}^{d\times d}$ and a starting point $s \in \mathbb{Q}^d$. An LDS describes a system whose state contains d rational numbers and evolves linearly. The *orbit* of such a system is the infinite sequence $\langle s, Ms, M^2s, \ldots\rangle$ of points in $\mathbb{Q}^d$. Orbits of LDS arise in many areas of computer science and mathematics, including verification of linear loops [10], automata theory [3], and the theory of linear recurrence sequences [17].

A fundamental problem about LDS is the question of deciding, given a system $\langle M, s\rangle$ and a semialgebraic target set $S \subseteq \mathbb{R}^d$, whether there exists $n \in \mathbb{N}$ such that $M^n s \in S$. This problem is known as the *reachability problem for LDS* and has been studied extensively over the last few decades. In their seminal work, Kannan and Lipton showed that the point-to-point reachability problem, i.e., the case in which S is a singleton, is decidable in polynomial time. At the same time they observed that the case in which S is a $(d-1)$-dimensional subspace of $\mathbb{R}^d$ (i.e. a hyperplane) is equivalent to the famous *Skolem problem* whose decidability remains open to this day. The Skolem problem asks, given a linear recurrence sequence defined by a recurrence relation $u_{n+d} = a_1 u_n + \ldots + a_d u_{n+d-1}$ and initial values $u_0, \ldots, u_{d-1}$, to decide whether there exists n such that $u_n = 0$. In addition to this Skolem-hardness, the difficulty of settling the reachability problem was further demonstrated by the results of [16], which show that solving the reachability problem with halfspace targets, known as the *positivity problem*, would entail major mathematical breakthroughs in the field of Diophantine approximation.

The reachability problem is defined with reference to the exact dynamics of an LDS. Since computational systems typically operate with finite precision, it is natural to consider an alternate notion of reachability involving so-called *pseudo-orbits.* The notion of pseudo-orbit is an important conceptual tool in dynamical systems that was studied by Anosov [2], Bowen [4], and Conley [6], and was used by the latter to prove what is sometimes called the fundamental theorem of dynamical systems. Given an LDS $\langle M, s\rangle$, a sequence $\langle x_n \mid n \in \mathbb{N}\rangle$ is an *ϵ-pseudo-orbit* of s under M if $x_0 = s$ and $\|Mx_n - x_{n+1}\| < \epsilon$ for all $n \in \mathbb{N}$. In other words, in a pseudo-orbit one considers an enlarged transition relation that is obtained by considering the dynamical system up to precision ϵ. Given $\epsilon > 0$, a set S is said to be *ϵ-pseudo-reachable* if there exists an ϵ-pseudo-orbit $\langle x_0 = s, x_1, x_2, \ldots\rangle$ of s under M that reaches S. We further say that S is *pseudo-reachable* if S is ϵ-pseudo-reachable for every $\epsilon > 0$. If a set S of error states is not pseudo-reachable then we can consider the system as being safe if implemented with sufficient precision, while if S is pseudo-reachable, it means that no finite amount of precision suffices to make the system reliably safe.

Recently, D'Costa et al. [7] considered the pseudo-reachability problem and, somewhat surprisingly, showed decidability in cases where S is a point (the pseudo-orbit problem), a hyperplane (the pseudo-Skolem problem) or a halfspace (the pseudo-positivity problem). Their proof of the first result relies on an exact characterisation of ϵ-pseudo-orbits. Their solution to the latter two problems, however, depends heavily on the fact that a hyperplane (a halfspace) can be defined using a single equality (inequality), an approach which unfortunately cannot be generalised to arbitrary semialgebraic targets. **In this work, we develop a novel logical approach to show the decidability of the pseudo-reachability problem for diagonalisable systems with arbitrary semialgebraic targets.**

1.1 High-level proof sketch of our approach

Our solution to the diagonalisable pseudo-reachability problem can be summarised as follows. Let $\widetilde{\mathcal{O}}_\epsilon(n)$ denote the set of all points that are reachable exactly at time n via an ϵ-pseudo-orbit. The pseudo-reachability problem then consists in checking whether the sentence $\Phi := \forall\epsilon.\, \exists n \in \mathbb{N} : \widetilde{\mathcal{O}}_\epsilon(n) \cap S \neq \emptyset$ is true. In this form, Φ is not amenable to application of logical methods as it involves both integer and real-valued variables, in addition to exponentiation with a complex base (coming from non-real eigenvalues of M). We therefore first move to the continuous domain and construct an abstraction $\mathcal{A}_\epsilon(t)$ for $t \in \mathbb{R}^{\geq 0}$ that is definable in $\mathbb{R}_{\exp}$ such that $\mathcal{A}_\epsilon(n) \supseteq \widetilde{\mathcal{O}}_\epsilon(n)$ for all $n \in \mathbb{N}$. We then investigate the values of ϵ and t that make $\Psi(\epsilon, t) := \mathcal{A}_\epsilon(t) \cap S \neq \emptyset$ true. We show that by the o-minimality of $\mathbb{R}_{\exp}$, either for every $\epsilon > 0$ there exists T such that for all $t > T$, $\Psi(\epsilon, t)$ holds, or the pseudo-reachability problem is equivalent to a finite-horizon reachability problem that is easily solvable. In the former case, it follows that for every $\epsilon > 0$, $\Psi(\epsilon, n)$ holds for all sufficiently large integer values n, thus establishing a bridge back to the discrete setting. We conclude by showing that in this case, S is pseudo-reachable. Intuitively, the idea is to use the universal quantification over ϵ to argue that if S can be reached using an $\epsilon/2$-abstraction at all but finitely many time steps, then it can be reached by an ϵ-pseudo-orbit, in fact at infinitely many possible time steps. The importance of the universal quantification is also illustrated by the following hardness result. For any fixed $\epsilon > 0$, it is decidable whether $\exists n \in \mathbb{N} : \mathcal{A}_\epsilon(n) \cap S \neq \emptyset$ holds, whereas the ϵ-pseudo-reachability problem of determining whether $\exists n \in \mathbb{N} : \widetilde{\mathcal{O}}_\epsilon(n) \cap S \neq \emptyset$ holds is hard with respect to (a hard subclass of) the Skolem problem, as shown in Section 4.

The approach outlined above can be adapted to solve a few other related problems about linear dynamical systems. An example would be the *robust reachability problem* recently considered by Akshay et al. in [1]: given an LDS $\langle M, s\rangle$ and a semialgebraic target S, decide whether for all $\epsilon > 0$ there exists a point s' in the ϵ-neighbourhood of s whose orbit reaches S. This problem can be thought of as a modification of the pseudo-reachability problem where only one perturbation is allowed at the very beginning. Due to this simplification, we are able to show, in the full version of this paper, full decidability (that is, without the restriction to diagonalisable systems) of the robust reachability problem. Finally, as the first step of our solution is to translate the problem into the continuous domain, the continuous versions of both the pseudo-reachability problem (Section 5) and the robust reachability problem (discussed in the full version) can be handled using the same approach, arguably more naturally. For the former, because we proceed by reducing the pseudo-reachability problem to bounded-time reachability problem, the decidability result assumes Schanuel's conjecture.

2 Mathematical tools

We write $B(c, r)$ for the closed ℓ_2-ball of radius r centred around $c \in \mathbb{R}^d$ and $\mathbf{0} \in \mathbb{R}^d$ for the d-dimensional zero vector. We denote by $\mathbb{T} \subseteq \mathbb{C}$ the unit circle in the complex plane and by $||x||$ the ℓ_2-norm of a vector $x \in \mathbb{R}^d$.

2.1 First-order logic

We denote by $\mathbb{R}_0$ the (structure of) real numbers with addition and multiplication, by $\mathbb{R}_{\exp}$ the real numbers with addition, multiplication and (unbounded) exponentiation and by $\mathbb{R}_{\exp,\cos\restriction[0,T]}$ the real numbers with exponentiation and bounded (in input, by some $T > 0$) trigonometric functions. By the Tarski-Seidenberg theorem, the theory of $\mathbb{R}_0$ admits effective quantifier elimination and is therefore decidable. The theories of $\mathbb{R}_{\exp}$ and $\mathbb{R}_{\exp,\cos\restriction[0,T]}$ are known to be decidable subject to Schanuel's conjecture (see, e.g., [11]) in transcendental number theory [13, 19]. However $\mathbb{R}_{\exp,\cos\restriction[0,T]}$ (and hence $\mathbb{R}_{\exp}$ and $\mathbb{R}_0$) are unconditionally known to be o-minimal [18]. That is, any subset of $\mathbb{R}$ definable using arithmetic operations, real exponentiation and bounded trigonometric functions is a finite union of intervals. In particular, any subset of $\mathbb{R}^{\geq 0}$ definable in this way is either bounded or contains all sufficiently large real numbers. For the discrete-time problems considered in this paper we will only need to work with $\mathbb{R}_{\exp}$. We will need $\mathbb{R}_{\exp,\cos\restriction[0,T]}$ only when considering the classical bounded-time reachability problem for continuous-time linear dynamical systems.

A *semialgebraic* set is a subset of $\mathbb{R}^d$ definable (without parameters) in $\mathbb{R}_0$. We say that a function $\varphi : \mathbb{R}^l \to \mathbb{R}^m$ is semialgebraic if its graph is a semialgebraic subset of $\mathbb{R}^{l+m}$. Intuitively, semialgebraic functions are exactly the functions that can be specified using arithmetic and logical operations over the real numbers.

2.2 Kronecker's theorem and its applications

The analysis of problems about linear dynamical systems often reduces to that of the orbit $\langle \Gamma^n \mid n \in \mathbb{N} \rangle$ where $\Gamma^n = (\gamma_1^n, \ldots, \gamma_k^n)$ for $\gamma_1, \ldots, \gamma_k \in \mathbb{T}$. Let $\mathcal{T} = \mathrm{cl}(\{\Gamma^n : n \in \mathbb{N}\})$ be the topological closure of this discrete orbit. The set $\mathcal{T}$ is semialgebraic and well-understood with the help of Kronecker's theorem in simultaneous Diophantine approximation [9].

▶ **Theorem 1** (Kronecker). *Let $\theta_1, \ldots, \theta_k, \varphi_1, \ldots, \varphi_k \in \mathbb{R}$ be such that for any $a_1, \ldots, a_k \in \mathbb{Z}$,*

$$\sum_{i=1}^{k} a_i \theta_i \in \mathbb{Z} \Rightarrow \sum_{i=1}^{k} a_i \varphi_i \in \mathbb{Z}.$$

For any $\epsilon > 0$ there exist infinitely many $n \in \mathbb{N}$ such that $\{n\theta_i - \varphi_i\} < \epsilon$ for all $1 \leq i \leq k$, where $\{x\}$ denotes the distance from $x \in \mathbb{R}$ to the nearest integer.

To apply this theorem to our situation, let

$$\mathcal{T} = \{(z_1, \ldots, z_k) : \forall a_1, \ldots, a_k \in \mathbb{Z} : \gamma_1^{a_1} \cdots \gamma_k^{a_k} = 1 \Rightarrow z_1^{a_1} \cdots z_k^{a_k} = 1\}.$$

For $z = (z_1, \ldots, z_k) \in \mathcal{T}$, by considering $\theta_i = \frac{\arg(\gamma_i)}{2\pi}$ and $\varphi_i = \frac{\arg(z_i)}{2\pi}$ for $1 \leq i \leq k$ we can deduce that for each $\epsilon > 0$ there exists n such that $||z - \Gamma^n|| < \epsilon$ and hence the orbit $\langle \Gamma^n \mid n \in \mathbb{N} \rangle$ is dense in $\mathcal{T}$. On the other hand, using Masser's deep results [14] about multiplicative relations between algebraic numbers one can compute, in polynomial time, a finite basis for $\{(a_1, \ldots, a_k) \in \mathbb{Z}^k : \gamma_1^{a_1} \cdots \gamma_k^{a_k} = 1\}$. Hence $\mathcal{T}$ is closed, semialgebraic and effectively computable. It then follows that $\mathcal{T} = \mathrm{cl}(\{\Gamma^n : n \in \mathbb{N}\})$.

We will also need the following lemma which is a consequence of the effective computability of $\mathcal{T} = \mathrm{cl}(\{\Gamma^n : n \in \mathbb{N}\})$ as a semialgebraic set.

▶ **Lemma 2.** *Let $R = \mathrm{diag}(\Lambda_1, \ldots, \Lambda_k) \in \mathbb{R}^{2k \times 2k}$ be a block diagonal matrix where Λ_i is an algebraic rotation matrix for $1 \leq i \leq k$. The closure of the set $\{R^n x : n \in \mathbb{N}\}$, for x with algebraic entries, is semialgebraic and effectively computable.*

The proof follows immediately from diagonalising R^n and observing that all eigenvalues of R are algebraic numbers in $\mathbb{T}$.

3 Decidability for discrete-time diagonalisable systems

In this section we prove our main result: the decidability of the pseudo-reachability problem for discrete-time diagonalisable *affine dynamical systems*, which are a generalisation of LDS. The reason we consider affine systems is that the well-known homogenisation trick (increasing the dimension by one and adding a coordinate that is always equal to 1) used for reducing the classical reachability problem for affine systems to the reachability problem for LDS doesn't work for the pseudo-reachability problem: when perturbations are allowed, one cannot force a coordinate to remain constant. Hence affine systems require separate treatment.

▶ **Problem 3** (pseudo-reachability). *Let $M \in \mathbb{Q}^{L \times L}$ be an update matrix, $s \in \mathbb{Q}^L$ be a starting point, $b \in \mathbb{Q}^L$ be an affine term and $S \subseteq \mathbb{R}^L$ be a semialgebraic target set. A sequence $\langle x_0 = s, x_1, x_2 \ldots \rangle$ is an ϵ-pseudo-orbit of s if $||Mx_n + b - x_{n+1}|| \leq \epsilon$ for all n. The pseudo-reachability problem asks: given M, b, s and S, decide whether for each $\epsilon > 0$ there exists an ϵ-pseudo-orbit of s that reaches the set S.*

Let $\widetilde{\mathcal{O}}_\epsilon(n)$ denote the set of all points that are reachable via an ϵ-pseudo-orbit of s under the map $x \mapsto Mx + b$ at time n. Since $\widetilde{\mathcal{O}}_\epsilon(0) = \{s\}$ and $\widetilde{\mathcal{O}}_\epsilon(n+1) = M\widetilde{\mathcal{O}}_\epsilon(n) + b + \epsilon B(\mathbf{0}, 1)$, by induction we can show that $\widetilde{\mathcal{O}}_\epsilon(n) = M^n s + \sum_{i=0}^{n-1} M^i b + \epsilon \sum_{i=0}^{n-1} M^i B(\mathbf{0}, 1)$. The pseudo-reachability problem is then equivalent to determining the truth of $\forall\epsilon.\, \exists n : \widetilde{\mathcal{O}}_\epsilon(n) \cap S \neq \emptyset$. Here $B(\mathbf{0}, 1)$ can be viewed as a set of "control inputs", and the pseudo-reachability problem can be viewed as the problem of determining whether S can be reached using arbitrarily small control inputs. The next lemma shows that we can in fact, choose any reasonable control set.

▶ **Lemma 4** (Invariance under change of the control set). *Let $\mathcal{B} \subseteq \mathbb{R}^L$ be a bounded set containing an open ball around the origin.*

1. *The pseudo-reachability problem as defined above is equivalent to the problem of determining whether*

$$\forall\epsilon.\, \exists n : (M^n s + \sum_{i=0}^{n-1} M^i b + \epsilon \sum_{i=0}^{n-1} M^i \mathcal{B}) \cap S \neq \emptyset.$$

2. *We may assume the matrix M is in real Jordan form.*

Proof. Since $\mathcal{B}$ is assumed to be bounded and to contain an open neighbourhood around the origin, there must exist constants C_1, C_2 such that $C_1 B(\mathbf{0}, 1) \subseteq \mathcal{B} \subseteq C_2 B(\mathbf{0}, 1)$. Hence

$$C_1 \epsilon \sum_{i=0}^{n-1} M^i B(\mathbf{0}, 1) \subseteq \epsilon \sum_{i=0}^{n-1} M^i \mathcal{B} \subseteq C_2 \epsilon \sum_{i=0}^{n-1} M^i B(\mathbf{0}, 1).$$

The proof of (1) then follows from the fact that ϵ is universally quantified: one can simulate (i) an ϵ-pseudo-orbit with control set $\mathcal{B}$ using a $C_2\epsilon$-pseudo-orbit with control set $B(\mathbf{0}, 1)$ and (ii) an ϵ-pseudo-orbit with control set $B(\mathbf{0}, 1)$ using a ϵ/C_1-pseudo-orbit with control set $\mathcal{B}$. Proof of (2) follows from observing that multiplying $B(\mathbf{0}, 1)$ by an invertible change of basis matrix results in a bounded control set containing a neighbourhood around $\mathbf{0}$. ◀

Observe that the change of the control set described above is not applicable when ϵ is fixed, as in the ϵ-pseudo-reachability problem discussed in Section 4.

3.1 A closed form for $\widetilde{\mathcal{O}}_\epsilon(n)$

We now use Lemma 4 to choose a control set that results in $\widetilde{\mathcal{O}}_\epsilon(n)$ with a convenient first-order closed form: observe that the naïve formulation above involves the term $\sum_{i=0}^{n-1} M^i B(\mathbf{0}, 1)$ which is not "first-order".

Assume M is diagonalisable and in real Jordan form: $M = \operatorname{diag}(\Lambda_1, \ldots, \Lambda_k, \rho_{k+1}, \ldots, \rho_d)$. That is, M consists of d block, the first k of which have dimension 2×2 and a pair of non-real conjugate eigenvalues, whereas the remaining blocks are 1×1 and real. Write ρ_j for the spectral radius of the jth block. We can factor M into a "scaling" and a "rotation" as $M = DR$ where $D = \operatorname{diag}(\rho_1, \rho_1, \ldots, \rho_k, \rho_k, \rho_{k+1}, \rho_{k+2}, \ldots, \rho_d)$ is diagonal and R is a block-diagonal matrix that consists of blocks that are either 2×2 rotation matrices or 1×1 and equal to $[\pm 1]$. Hereafter we will be using the convenient "rotation-invariant" control set

$$\mathcal{B} = \prod_{j=1}^{k} B((0,0), 1) \times \prod_{j=k+1}^{d} [-1, 1] = \prod_{j=1}^{d} B(\mathbf{0}, 1)$$

where $B((0,0),1)$ is the unit disc. Observe that $\mathcal{B}$ is a product of ℓ_2-balls that matches the block structure of M. It follows that $R\mathcal{B} = \mathcal{B}$ and hence

$$\widetilde{\mathcal{O}}_\epsilon(n) = D^n R^n s + \sum_{i=0}^{n-1} M^i b + \epsilon \sum_{i=0}^{n-1} D^i R^i \mathcal{B} = D^n R^n s + \sum_{i=0}^{n-1} M^i b + \epsilon\mathcal{B}(n)$$

where $\mathcal{B}(n) = \sum_{i=0}^{n-1} D^i \mathcal{B}$. We then have

$$\mathcal{B}(n) = \sum_{i=0}^{n-1} D^i \prod_{j=1}^{d} B(\mathbf{0},1) = \sum_{i=0}^{n-1}\prod_{j=1}^{d} B(\mathbf{0},\rho_j^i) = \prod_{j=0}^{d} B(\mathbf{0},\sum_{i=0}^{n-1}\rho_j^i).$$

Geometrically, the idea is that a 2×2 or a 1×1 block of D maps an origin-centred disc (which corresponds to a symmetric interval in 1D) to an origin-centred disc, and a set-sum of such discs is again an origin-centred disc. Note that our ability to reason in this way crucially depends on the fact that M is diagonalisable. Finally, since $\sum_{i=0}^{n-1} \rho_j^i$ is either $\frac{\rho_j^n - 1}{\rho_j - 1}$ or $n\rho_j$, we can write $\mathcal{B}(n) = \{z : \varphi(z, n, \rho_1^n, \ldots, \rho_d^n)\}$, where φ is a semialgebraic predicate.

We can apply the blockwise summation technique, distinguishing between the cases where the spectral radius of the block is 1 or different from 1, to the term $\sum_{i=0}^{n-1} M^i b$ to obtain the closed form $\sum_{i=0}^{n-1} M^i b = D^n R^n x' + cn + d$, where x', c and d only depend on M and b. We then fold s and x' into a new, fictive starting point x to obtain the final closed form

$$\widetilde{\mathcal{O}}_\epsilon(n) = D^n R^n x + cn + d + \epsilon\mathcal{B}(n).$$

In order to solve the pseudo-reachability problem, we henceforth consider the problem of determining the truth of the sentence $\forall\epsilon > 0.\, \exists n : (D^n R^n x + cn + d + \epsilon\mathcal{B}(n)) \cap S \neq 0$, where all the input vectors and matrices have real algebraic entries.

3.2 Passing to the abstraction

The expression for $\widetilde{\mathcal{O}}_\epsilon(n)$ contains the term $D^n R^n x$, which is the last obstacle to obtaining an expression which we can attack using known results about theories of real numbers. To address this issue we resort to abstracting $\widetilde{\mathcal{O}}_\epsilon(n)$. Let

$$\mathcal{T} := \mathrm{cl}\left(\{R^n x : n \in \mathbb{N}\}\right) \text{ and } \mathcal{A}_\epsilon(n) := D^n\mathcal{T} + cn + d + \epsilon\mathcal{B}(n)$$

where $\mathcal{T}$ is the closure of the orbit of x under R, and is semialgebraic and effectively computable by the discussion in Subsection 2.2. Moreover, recall that by Kronecker's theorem for every $z \in \mathcal{T}$ and $\epsilon > 0$ there exist infinitely many integers $0 < n_1 < n_2 < \ldots$ such that $||R^{n_i}x - z|| < \epsilon$ for all i.

Here $\mathcal{A}_\epsilon(n)$ acts as an abstraction of $\widetilde{\mathcal{O}}_\epsilon(n)$. In particular, for all $\epsilon > 0$ and $n \in \mathbb{N}$ we have $\mathcal{A}_\epsilon(n) \supseteq \widetilde{\mathcal{O}}_\epsilon(n)$. Observe that $\mathcal{A}_\epsilon(n) = \{z : \varphi(z, \epsilon, n, \rho_1^n, \ldots, \rho_d^n)\}$ for a semialgebraic predicate φ. Viewing $\mathcal{A}_\epsilon(n)$ as a proxy for $\widetilde{\mathcal{O}}_\epsilon(n)$, we arrive at the following dichotomy.

▶ **Lemma 5.** *Either*

1. *for every $\epsilon > 0$ there exists N_ϵ such that for all $n > N_\epsilon$, $\mathcal{A}_\epsilon(n)$ intersects S, or*
2. *there exist N and $\epsilon > 0$, both effectively computable, such that $\mathcal{A}_\epsilon(n)$ does not intersect S for all $n > N$.*

Moreover, it can be effectively determined which case holds.

Proof. First we show that the dichotomy holds, putting the issues of effectiveness aside. Let

$$\Phi(\epsilon, n) = \bigvee_{\alpha \in A} \bigwedge_{\beta \in B} p_{\alpha,\beta}(\epsilon, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$$

be a quantifier-free formula equivalent to $\mathcal{A}_\epsilon(n) \cap S \neq \emptyset$. Such $\Phi(\epsilon, n)$ must exist because $\mathcal{A}_\epsilon(n)$ is semialgebraic with parameters from $\{\epsilon, n, \rho_1^n, \ldots, \rho_d^n\}$ and by the Tarski-Seidenberg theorem, each such set can be described using a quantifier-free formula of the form given above. Suppose Case 1 does not hold. Then there exists a particular $\epsilon > 0$ such that $\Phi(\epsilon, n)$ does not hold for arbitrarily large n. Treating n as a continuous parameter, consider the set $\{n \in \mathbb{R}_{\geq 0} : \Phi(\epsilon, n) \text{ does not hold}\}$. By o-minimality of $\mathbb{R}_{\text{exp}}$ this set is a finite union of intervals and and by the assumption that Case 1 does not hold, it contains arbitrarily large integers. Hence it must contain all integers in (N, ∞) for some $N \in \mathbb{N}$. That is, for all $n > N$ the formula $\Phi(\epsilon, n)$ does not hold.

Effectiveness. We now address the issues of effectiveness. Consider the formula

$$\Psi(\epsilon) = \exists N_\epsilon . \forall n > N_\epsilon : \Phi(\epsilon, n).$$

We show that $\Psi(\epsilon)$ is equivalent to a formula $\psi(\epsilon)$ in the language of $\mathbb{R}_0$. To determine which case holds it then remains to determine the truth value of the sentence $\forall \epsilon : \psi(\epsilon)$.

By the o-minimality argument above, given $\epsilon > 0$, each $p_{\alpha,\beta}(\epsilon, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$ either holds for finitely many integer values of n or holds for all sufficiently large integer values n. By elementary considerations it follows that $\Psi(\epsilon)$ is equivalent to

$$\bigvee_{\alpha \in A} \bigwedge_{\beta \in B} \exists N_\epsilon . \forall n > N_\epsilon : p_{\alpha,\beta}(\epsilon, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0.$$

Hence it suffices to show how to construct a formula $\psi(\epsilon)$ in the language of $\mathbb{R}_0$ that is equivalent to $\exists N_\epsilon . \forall n > N_\epsilon : p_{\alpha,\beta}(\epsilon, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$. For each $\epsilon > 0$, the formula first tests if $p_{\alpha,\beta}(\epsilon)$ (as a polynomial in $d+1$ remaining variables) is identically zero. If yes, then $\varphi(\epsilon)$ is true or false depending only on $\bowtie_{\alpha,\beta}$. Otherwise, write $p_{\alpha,\beta}(\epsilon, n, \rho_1^n, \ldots, \rho_d^n) = \sum_{i=1}^k q_i(\epsilon, n) R_i^n$ where $q_i(\epsilon)$ is not identically zero for all i and $R_1 > \cdots > R_k > 0$ are real algebraic numbers of the form $\rho_1^{p_1} \cdots \rho_d^{p_d}$ for $p_1, \ldots, p_d \in \mathbb{N}$. Since $|q_1(\epsilon, n) R_1^n| > \left|\sum_{i=2}^k q_i(\epsilon, n) R_i^n\right|$ for sufficiently large n, whether $p_{\alpha,\beta}(\epsilon, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$ holds for sufficiently large n depends only on $q_1(\epsilon, n)$. Hence we can choose $\psi(\epsilon)$ to be $\lim_{n \to \infty} q_1(\epsilon) \bowtie_{\alpha,\beta} 0$, which amounts to a sign condition on the coefficients of $q_1(\epsilon, n)$.

Computing N. Finally, we show that in Case 2, the value N can be effectively computed. To this end, by repeatedly trying smaller and smaller values of ϵ first compute a rational $e > 0$ such that $\Psi(e)$ (equivalently, $\psi(e)$) does not hold. To be able to compute N it then suffices to compute, for a particular (α, β), a value $N_{\alpha,\beta}$ such that $p_{\alpha,\beta}(e, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$ does not hold for all $n > N_{\alpha,\beta}$, assuming that it does not hold for sufficiently large n. We can then take N to be the maximum of $N_{\alpha,\beta}$ over $(\alpha, \beta) \in A \times B$.

To compute $N_{\alpha,\beta}$, consider $p := p_{\alpha,\beta}(e)$. Assuming it is not identically zero (otherwise we can choose $N_{\alpha,\beta}$ to be any positive integer), write $p(n, \rho_1^n, \ldots, \rho_d^n) = \sum_{i=1}^k q_i(n) R_i^n$ where q_i is not identically zero for all i and $R_1 > \cdots > R_k > 0$ are real algebraic. Since $p_{\alpha,\beta}(e, n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$ and hence $p(n, \rho_1^n, \ldots, \rho_d^n) \bowtie_{\alpha,\beta} 0$ do not hold for sufficiently large n, it must be the case that $q_1(n) \bowtie_{\alpha,\beta} 0$ does not hold for sufficiently large n. Hence it remains to choose $N_{\alpha,\beta}$ large enough so that for all $n > N_{\alpha,\beta}$, $|q_1(n) R_1^n|$ dominates $\left|\sum_{i=2}^k q_i(n) R_i^n\right|$. ◀

3.3 From the abstraction back to ϵ-pseudo-orbits

In this section we consider the relationship between the two cases of Lemma 5 and our original pseudo-reachability problem. We start with Case 2. Observe that $\mathcal{O}_\epsilon(n) \subseteq \mathcal{A}_\epsilon(n)$ for every $n \in \mathbb{N}$ and $\epsilon > 0$. Therefore, when Case 2 holds, for any $n > N$ and $\epsilon > 0$ the target set cannot be reached by $\mathcal{O}_\epsilon$. It remains to check pseudo-reachability at time steps $n \leq N$. We claim that S is pseudo-reachable if and only

$$\forall \epsilon. \exists n \leq N : (M^n x + cn + d + \epsilon \mathcal{B}(n)) \cap S \neq \emptyset.$$

Let $\overline{S}$ denote the topological closure of S. We show that the statement above is equivalent to $\exists n \leq N : M^n x + cn + d \in \overline{S}$. Observe that if for all $n \leq N$ the point $M^n x + cn + d$ is not in $\overline{S}$, then by compactness the smallest distance from $\{M^n x + cn + d \mid n \leq N\}$ to $\overline{S}$ is positive and hence for sufficiently small ϵ the target S cannot be ϵ-pseudo-reached within the first N steps. Conversely, if $M^n x + cn + d \in \overline{S}$ for some $n \leq N$, then because $\mathcal{B}(n)$ is full dimensional and contains $\mathbf{0}$ in its interior, it follows that $(M^n x + cn + d + \epsilon \mathcal{B}(n)) \cap S \neq \emptyset$ for all $\epsilon > 0$. Therefore, in Case 2 pseudo-reachability can be decided by simply checking if $\{M^n x + cn + d \mid n \leq N\}$ reaches $\overline{S}$.

Next we will show that S is pseudo-reachable if Case 1 holds. Given $z \in \mathcal{T}$, we define a "localisation" of the abstraction at the point z as $\mathcal{A}_\epsilon(n)(z) := D^n z + cn + d + \epsilon \mathcal{B}(n)$. Observe that $\mathcal{A}_\epsilon(n) = \{\mathcal{A}_\epsilon(n)(z) : z \in \mathcal{T}\}$. This definition of a localisation will allow us to select a "concrete trajectory" from the set of all possible (abstract) trajectories.

Fix $\epsilon > 0$ and let $T_n := \{z \in \mathcal{T} : \mathcal{A}_\epsilon(n)(z)$ intersects $S\}$. The next lemma implies that the sequence T_n must tend towards a limiting shape; i.e. it cannot "jump around" forever.

▶ **Lemma 6.** *Let $T_n = \{z : \varphi(z, n, \rho_1^n, \dots, \rho_d^n)\}$, where φ is a semialgebraic predicate and $\rho_1, \dots, \rho_d$ are real algebraic, be a family of non-empty sets contained in a compact set $\mathcal{T}$. There exists a non-empty limiting set $L \subseteq \mathcal{T}$ to which the sequence T_n converges as $n \to \infty$, in the following sense.*

a *For every $\epsilon > 0$, there exists N such that for all $n > N$, $T_n \subseteq L + B(\mathbf{0}, \epsilon)$.*

b *For all $z \in L$ and $\epsilon > 0$ there exists N such that for all $n > N$, $z + B(\mathbf{0}, \epsilon)$ intersects T_n.*

Proof. Write $\varphi(z, n, \rho_1^n, \dots, \rho_d^n) = \bigvee_{\alpha \in A} \bigwedge_{\beta \in B} p_{\alpha,\beta}(z, n, \rho_1^n, \dots, \rho_d^n) \bowtie_{\alpha,\beta} 0$. We can define the sequence $\langle T_t \mid t \in \mathbb{R} \rangle$ as $T_t = \{z : \varphi(z, t, \rho_1^t, \dots, \rho_d^t)\}$. Let $L = \{x : \liminf d(x, T_t) = 0\}$ where $d(x, T_t)$ denotes the shortest Euclidean distance from x to a point in T_t.

We prove the first claim by contradiction. Suppose there exists $\epsilon > 0$ such that at infinitely many unbounded time steps $t_1 < t_2 < \dots$ there are points $z_1, z_2, \dots$ such that $z_i \in T_i$ but $z_i \notin L + B(\mathbf{0}, \epsilon)$. Then the sequence z_i must have an accumulation point z in $\mathcal{T} \setminus L$. But z will also satisfy $\liminf d(z, T_t) = 0$ and hence $z \in L$, a contradiction.

We prove the second claim using o-minimality of $\mathbb{R}_{\exp}$. Fix $z \in L$ and $\epsilon > 0$ and consider the set $Z = \{t \in \mathbb{R} : z + B(\mathbf{0}, \epsilon)$ intersects $T_t\}$. The set Z is o-minimal, and since $z \in L$, it is unbounded from above. Hence it must contain an interval of the form (N, ∞), which implies the desired result. ◀

One can also show that the set L described above is in fact semialgebraic, but this is not necessary for our arguments. We are now ready to show that S is pseudo-reachable if Case 1 of Lemma 5 holds.

▶ **Lemma 7.** *If for every $\epsilon > 0$ there exists N_ϵ such that for all $n > N_\epsilon$, $\mathcal{A}_\epsilon(n)$ intersects S then S is pseudo-reachable.*

The main idea of the proof is to use the assumption that $\mathcal{A}_{\epsilon/2}(n)$ intersects S for sufficiently large n to construct an ϵ-pseudo-orbit that hits S. Intuitively, in order to simulate $\mathcal{A}_{\epsilon/2}(n)$ using an ϵ-pseudo-orbit, $\epsilon/2$ of the total control allowance is used to replicate the effect of the control inputs (of size at most $\epsilon/2$, corresponding to the $\frac{\epsilon}{2}\mathcal{B}(n)$ term in the definition of $\mathcal{A}_{\epsilon/2}(n)$) and the remaining $\epsilon/2$ is used to compensate for the abstraction from the starting point a to the set $\mathcal{T}$. In fact, we do not know if one can deduce that S is ϵ-pseudo-reachable from knowing that $\mathcal{A}_{\epsilon}(n)$ intersects S for sufficiently large n. This illustrates the reason why the pseudo-reachability problem is easier than the ϵ-pseudo-reachability problem; see Section 4 for a more concrete argument.

Proof. Fix $\epsilon > 0$. We show how to construct an ϵ-pseudo-orbit that hits S. Consider $\mathcal{A}_{\epsilon/2}(n)$. By assumption, there exists N_1 such that for all $n > N_1$, $\mathcal{A}_{\epsilon/2}(n)$ intersects S. We now investigate which localisations of the abstraction are responsible for intersecting S. Apply Lemma 6 to the sequence of sets $T_n = \{z \in \mathcal{T} : \mathcal{A}_{\epsilon/2}(n)(z) \text{ intersects } S\}$ to obtain their "limit" L. Fix any $p \in L$.

Let ϵ' be small enough so that $\epsilon' D^n \mathcal{B} \subseteq \frac{\epsilon}{2}\mathcal{B}(n)$ for all $n > 0$. Intuitively, such ϵ' must exist because $D^n\mathcal{B}$ and $D^{n-1}\mathcal{B}$ only differ by at most a constant factor that only depends on the magnitudes $\rho_1, \dots, \rho_d$ of eigenvalues of M, and we have that $D^{n-1}\mathcal{B} \subseteq \sum_{i=0}^{n-1} D^i\mathcal{B} = \mathcal{B}(n)$. By Lemma 6 (b), there exists $N > N_1$ such that for all $n > N$, $p + B(\mathbf{0}, \epsilon'/2)$ intersects T_n. That is, for all $n > N$ there exists $p_n \in \mathcal{T}$ such that $||p - p_n|| < \epsilon'/2$ and $p_n \in T_n$. Equivalently,

$$||p - p_n|| < \epsilon'/2 \text{ and } \mathcal{A}_{\epsilon/2}(n)(p_n) \text{ intersects } S.$$

By Kronecker's theorem there must exist $m > N$ such that $||R^m x - p|| < \epsilon'/2$. Hence we have $||R^m x - p_m|| < \epsilon'$ which implies $R^m x - p_m \in \epsilon'\mathcal{B}$ and hence $D^m(R^m x - p_m) \in \epsilon' D^m\mathcal{B}$. Since by construction of ϵ' we have $\epsilon' D^m \mathcal{B} \subseteq \frac{\epsilon}{2}\mathcal{B}(m)$, it follows that $D^m(R^m x - p_m) \in \frac{\epsilon}{2}\mathcal{B}(m)$ and hence $D^m p_m \in D^m R^m x + \frac{\epsilon}{2}\mathcal{B}(m)$. Therefore,

$$\widetilde{\mathcal{O}}_{\epsilon}(m) = (D^m R^m x + \frac{\epsilon}{2}\mathcal{B}(m)) + cm + d + \frac{\epsilon}{2}\mathcal{B}(m) \supseteq D^m p_m + cm + d + \frac{\epsilon}{2}\mathcal{B}(m) = \mathcal{A}_{\epsilon/2}(m)(p_m).$$

Since $\mathcal{A}_{\epsilon/2}(m)(p_m)$ intersects S, it then follows that $\widetilde{\mathcal{O}}_{\epsilon}(m)$ too must intersect S. ◀

3.4 The algorithm

To summarise, the analysis above gives us the following algorithm for determining if S is pseudo-reachable, i.e. if $\forall \epsilon > 0 . \exists n : \widetilde{\mathcal{O}}_{\epsilon}(n) \cap S \neq \emptyset$. Let $\varphi(n, \epsilon)$ be a quantifier-free formula in $\mathbb{R}_{\exp}$ defining the abstraction $\mathcal{A}_{\epsilon}(n)$. First determine, using the algorithm described in the proof of Lemma 5, whether Case 1 or Case 2 holds. If the former holds, then conclude that S is pseudo-reachable. If Case 2 holds, then compute the value of N effectively and check if there exists $n < N$ such that $(M^n x + cn + d) \cap \overline{S} \neq \emptyset$.

4 Skolem-hardness of the ϵ-pseudo-reachability problem

In this section we consider the ϵ-pseudo-reachability problem for discrete diagonalisable systems: given diagonalisable M, starting point s, a target set S and $\epsilon > 0$, decide whether there exists n such that $M^n s + \sum_{k=0}^{n-1} M^k B(\mathbf{0}, \epsilon) \cap S \neq 0$. This problem is also known as the reachability problem for linear time-invariant systems [8] with the control set $B(\mathbf{0}, \epsilon)$. We will reduce a hard subclass of the Skolem problem to our ϵ-pseudo-reachability problem.

The Skolem problem is not known to be decidable for orders $d \geq 5$, even for diagonalisable recurrences. The largest class of sequences for which decidability is known is the MSTV (Mignotte-Shorey-Tijdeman-Vereschagin) class, which consists of all linear recurrence sequences over integers that (i) have at most three dominant roots with respect to the usual (Archimedean) absolute or (ii) have at most two dominant roots with respect to a p-adic absolute value [12]. We consider the Skolem problem for integer sequences whose roots $\rho, \lambda_1, \ldots, \lambda_d$ satisfy $\rho = |\lambda_1| = \cdots = |\lambda_d|$. This class of sequences contains infinitely many instances that are not in the MSTV class and hence is a hard subclass of the Skolem problem.

Recall that any linear recurrence sequence can be written as $u_n = c^\top M^n s$ where M is the companion matrix of u_n whose eigenvalues are the roots of u_n. Let u_n be a diagonalisable sequence that belongs to the hard subclass described above, i.e. $u_n = c^\top M^n s$ where $M = \operatorname{diag}(\Lambda_1, \ldots, \Lambda_d, \rho)$ and Λ_i is a 2×2 real Jordan block with $\rho(\Lambda_i) = \rho$ for $1 \leq i \leq d$. We reduce the problem "does u_n have a zero?" to an ϵ-pseudo-reachability problem.

Consider the sequence $v_n = u_n^2$. Observe that $v_n = \sum_{i=1}^{L} c_i \Gamma_i^n s_i + C r^n$ where

- $r = \rho^2$,
- Γ_i is a 2×2 real Jordan block with $\rho(\Gamma_i) = r$ for $1 \leq i \leq L$,
- $c_i, s_i \in \mathbb{R}^2$ for $1 \leq i \leq L$, and
- $C > 0$.

The first two statements follow from the fact that the eigenvalues of v_n are products of eigenvalues of u_n. That $C > 0$ can be deduced as follows. Consider $w_n = \sum_{i=1}^{L} c_i \Gamma_i^n s_i$. It only has non-real roots and hence by [15] is infinitely often positive and negative. Hence if C is not positive, then $v_n < 0$ for infinitely many n, which contradicts the fact that $v_n \geq 0$.

Next observe that u_n has a zero iff there exists n such that $v_n \leq 0$. Since we are interested only in the sign of v_n, by scaling v_n by $C(2r)^n$ if necessary we assume that $r \in (0, 1)$ and $C = 1$. We will construct an instance of the ϵ-pseudo-reachability problem that is positive if and only if there exists n such that $v_n \leq 0$.

Define

- $A = \operatorname{diag}(\Gamma_1, \ldots, \Gamma_L)$,
- $s = (s_1, \ldots, s_L)$ and $c = (c_1, \ldots, c_L)$,
- $\epsilon = \frac{1-r}{||c||}$, and
- $H = \{z : c^\top \cdot z + 1 \leq 0\}$.

Observe that H is ϵ-pseudo-reachable if and only if $A^n s + \sum_{i=0}^{n-1} A^i B(\mathbf{0}, \epsilon) \cap H \neq 0$ for some n. Since $A^i B(\mathbf{0}, \epsilon) = B(\mathbf{0}, r^i \epsilon)$, we have $\sum_{i=0}^{n-1} A^i B(\mathbf{0}, \epsilon) = B(\mathbf{0}, \frac{1-r^n}{1-r}\epsilon) \coloneqq \mathcal{B}(n)$ and

$$H \text{ is } \epsilon\text{-pseudo-reachable} \iff \min_{z \in \mathcal{B}(n)} c \cdot (A^n s + z) + 1 \text{ is} \leq 0 \text{ for some } n.$$

We will show that in fact $\min_{z \in \mathcal{B}(n)} c \cdot (A^n s + z) + 1 = v_n$, which will conclude the proof.

$$\begin{aligned}
\min_{z \in \mathcal{B}(n)} c \cdot (A^n s + z) + 1 &= \sum_{i=1}^{L} c_i \Gamma_i^n s_i + 1 + \min_{z \in \mathcal{B}(n)} c \cdot z \\
&= \sum_{i=1}^{L} c_i \Gamma_i^n s_i + 1 - ||c|| \frac{1 - r^n}{1 - r} \epsilon \\
&= \sum_{i=1}^{L} c_i \Gamma_i^n s_i + r^n \\
&= v_n.
\end{aligned}$$

5 The continuous-time pseudo-reachability problem

In this section we show that the approach we described in Section 3 for deciding the discrete-time pseudo-reachability problem for diagonalisable systems also works in the continuous setting with one important difference: to handle Case 2 of the dichotomy lemma (exactly the same as Lemma 5) we need to solve the *bounded-time reachability problem for continuous-time affine dynamical systems*, which is only known to be decidable assuming Schanuel's conjecture [5]. For detailed proofs see the full version of the paper.

Let $M = \operatorname{diag}(\Lambda_1, \dots, \Lambda_k, \rho_{k+1}, \dots, \rho_d) \in (\mathbb{R} \cap \overline{\mathbb{Q}})^{L \times L}$ be a diagonalisable matrix in real Jordan form, $s \in \mathbb{Q}^L$ be a starting point, $b \in \mathbb{Q}^L$ be an affine term and $S \subseteq \mathbb{R}^L$ be a semialgebraic target set. The trajectory of the system (in the absence of additional control inputs) is given by

$$x(t) = e^{Mt}s + \int_0^t e^{Mh} b \, dh.$$

Intuitively, while in the discrete setting control inputs are applied after each unit of time and thus are represented by a sequence $\langle d_n \mid n \in \mathbb{N} \rangle$, in the continuous setting they are represented by a continuous function $\Delta : \mathbb{R}_{\geq 0} \to \mathbb{R}^L$. Hence an ϵ-pseudo-orbit is defined as a trajectory

$$x(t) = e^{Mt}s + \int_0^t e^{Mh} b \; dh + \int_0^t e^{Mh} \Delta(t-h) \, dh.$$

for some control signal $\Delta : \mathbb{R}_{\geq 0} \to \mathbb{R}^L$ satisfying $||\Delta_\epsilon(t)|| \leq \epsilon$ for all $t \geq 0$. The pseudo-reachability problem is then defined in the same way as before: decide whether for every $\epsilon > 0$ there exists an ϵ-pseudo-orbit that reaches S.

Let $\mathcal{B}$ be the same control set as defined in Subsection 3.1. For $1 \leq j \leq k$, let $r_j = \operatorname{Re}(\lambda_j)$ and $\omega_j = \operatorname{Im}(\lambda_j)$ where λ_j is a non-real eigenvalue of the block Λ_j. For $k < j \leq d$ let $r_j = \rho_j$ and $\omega_j = 0$. By using essentially the same arguments as in Subsection 3.1, we can show that the pseudo-reachability problem is equivalent to determining the truth of

$$\forall \epsilon > 0. \, \exists t : (e^{Mt}x + ct + d + \epsilon\mathcal{B}(t)) \cap S \neq \emptyset$$

where x, c, d are L-dimensional vectors and $\mathcal{B}(t) = \{z : \varphi(z, t, e^{r_1 t}, \dots, e^{r_d t})\}$ for semialgebraic predicate φ. We denote the term $e^{Mt}x + ct + d + \epsilon\mathcal{B}(t)$ by $\widetilde{\mathcal{O}}_\epsilon(n)$.

To define a convenient abstraction, we again write $e^{Mt} = D(t)R(t)$ where $D(t) := \operatorname{diag}(e^{r_1 t}, e^{r_1 t}, \dots, e^{r_k t}, e^{r_k t}, e^{r_{k+1} t}, e^{r_{k+2} t}, \dots, e^{r_{k+d} t})$ is diagonal and $R(t)$ is a block diagonal matrix whose blocks are rotation matrices of the form $\begin{bmatrix} \cos(\omega_j t) & -\sin(\omega_i t) \\ \sin(\omega_j t) & \cos(\omega_j t) \end{bmatrix}$ for $1 \leq j \leq k$ and are of the form $\Omega_i = \begin{bmatrix} 1 \end{bmatrix}$ for $k + 1 \leq j \leq d$. Just as in the discrete case, we next define

$$\mathcal{T} := \operatorname{cl}(\{R(t)x : t \in \mathbb{R}_{\geq 0}\}) \text{ and } \mathcal{A}_\epsilon(t) := D(t)\mathcal{T} + ct + d + \epsilon\mathcal{B}(t),$$

where $\mathcal{T}$ is again semialgebraic and effectively computable [5] and $\mathcal{A}_\epsilon$ acts as an abstraction of $\widetilde{\mathcal{O}}_\epsilon(t)$. In particular, for all $\epsilon > 0$ and $t \in \mathbb{R}_{\geq 0}$, we have $\widetilde{\mathcal{O}}_\epsilon(t) \subseteq \mathcal{A}_\epsilon(t)$. Moreover, observe that $\mathcal{A}_\epsilon(t) = \varphi(t, e^{r_1 t}, \dots, e^{r_d t})$ for a semialgebraic function φ, which is the most important property we need. In the full verision, using the same approach based on o-minimality we show the following.

▶ **Theorem 8.** *The continuous-time pseudo-reachability problem reduces to the bounded-time reachability problem for continuous-time affine dynamical systems.*

Intuitively, the dichotomy lemma (Lemma 5) holds verbatim for the continuous systems, and in Case 1 again S is always pseudo-reachable. It then remains to handle Case 2. Since bounded-time reachability problem for continuous-time affine dynamical systems can be encoded in $\mathbb{R}_{\exp,\cos\restriction[0,T]}$, we have the following (conditional) decidability result.

▶ **Corollary 9.** *Continuous-time pseudo-reachability problem for diagonalisable affine dynamical systems is decidable subject to Schanuel's conjecture.*

6 Discussion

The main technical result of our paper is that it is decidable whether

$$\forall \epsilon > 0. \exists n : (M^n x + f(n) + \epsilon\mathcal{B}(n)) \cap S \neq 0$$

where M is a diagonalisable matrix with algebraic entries, x is an algebraic starting point, f is a semialgebraic function, S is a semialgebraic target and $\mathcal{B}(n) = \varphi(n, \rho_1^n, \dots, \rho_d^n)$ for $\rho_1, \dots, \rho_d \in \mathbb{R} \cap \overline{\mathbb{Q}}$ and a semialgebraic function φ. We used this result to show decidability of the discrete-time pseudo-reachability problem for diagonalisable systems in the following way. We first observed that the pseudo-reachability problem can be cast as the problem of determining whether $\forall \epsilon > 0. \exists n : \widetilde{\mathcal{O}}_\epsilon(n) \cap S \neq \emptyset$, where $\widetilde{\mathcal{O}}_\epsilon(n)$ is the set of all points that are reachable exactly at the time n via an ϵ-pseudo-orbit. After choosing $\mathcal{B}$ as the most convenient control set (see Lemma 4 and Subsection 3.1), we then showed that $\widetilde{\mathcal{O}}_\epsilon(n)$ can be written as $M^n x + f(n) + \epsilon\mathcal{B}(n)$.

The reason we are unable to show decidability for non-diagonalisable systems in this fashion is that we are unable to write $\widetilde{\mathcal{O}}_\epsilon(n)$ as $M^n x + f(n) + \epsilon\mathcal{B}(n)$. For example, already for the Jordan block $M = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, and in general already for blocks with a single repeated real eigenvalue, we do not know whether it is even possible to eliminate the summation $\sum_{i=0}^{n-1} M^i B$ and express $\widetilde{\mathcal{O}}_\epsilon(n) = M^n x + \epsilon \sum_{i=0}^{n-1} M^i B$, where B is any full dimensional shape containing $\mathbf{0}$ in its interior, in the required fashion.

Our approach, however, can be used to solve, in full generality, the robust reachability problem of [1]: given M, x and S, decide whether $\forall \epsilon > 0 : \exists n : (M^n x + \epsilon M^n B(\mathbf{0}, 1)) \cap S \neq \emptyset$. Intuitively, the reason is that in this version there is no summation of the form $\sum_{i=0}^{n-1} M^i B$. Detailed proofs (for both the discrete-time and the continuous-time versions) can be found in the full version. For diagonalisable systems in particular, decidability of the robust reachability problem is almost immediate. First, one can again show that the problem is equivalent to determining whether $\forall \epsilon > 0 : \exists n : (M^n x + \epsilon M^n \mathcal{B}) \cap S \neq \emptyset$. It then remains to observe that $M^n \mathcal{B} = \varphi(n, \rho_1^n, \dots, \rho_d^n)$ for a semialgebraic predicate φ and apply the technical result described above.

References

1 S. Akshay, Hugo Bazille, Blaise Genest, and Mihir Vahanwala. On Robustness for the Skolem and Positivity Problems. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.5.

2 Dmitri V. Anosov. Geodesic flows on closed Riemannian manifolds of negative curvature. *Proc. Steklov Inst. Math.*, 90, 1967.

3 Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki. A Robust Class of Linear Recurrence Sequences. In *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2020.9.

4 Rufus Bowen. *Equilibrium States and the Ergodic Theory of Anosov Diffeomorphisms*, volume 470 of *Lecture Notes in Mathematics*. Springer-Verlag, 1975.

5 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem Problem for Continuous Linear Dynamical Systems. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2016.100.

6 Charles C. Conley. *Isolated invariant sets and the Morse index*, volume 25 of *CBMS Regional Conference Series in Mathematics*. American Mathematical Society, 1978.

7 Julian D'Costa, Toghrul Karimov, Rupak Majumdar, Joël Ouaknine, Mahmoud Salamati, Sadegh Soudjani, and James Worrell. The Pseudo-Skolem Problem is Decidable. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.34.

8 Nathanaël Fijalkow, Joël Ouaknine, Amaury Pouly, João Sousa-Pinto, and James Worrell. On the decidability of reachability in linear time-invariant systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 77–86, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3302504.3311796.

9 Godfrey H. Hardy and Edward M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 1999.

10 Toghrul Karimov, Engel Lefaucheux, Joël Ouaknine, David Purser, Anton Varonka, Markus A. Whiteland, and James Worrell. What's decidable about linear loops? *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi:10.1145/3498727.

11 Serge Lang. *Introduction to transcendental numbers*. Addison-Wesley series in mathematics. Addison-Wesley Pub. Co., 1966.

12 Richard Lipton, Florian Luca, Joris Nieuwveld, Joël Ouaknine, David Purser, and James Worrell. On the skolem problem and the skolem conjecture. *To appear in LICS 2022*, 2022. URL: https://people.mpi-sws.org/~joel/publications/skolem_five22.pdf.

13 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

14 David W. Masser. Linear relations on algebraic groups. In *New Advances in Transcendence Theory*. Camb. Univ. Press, 1988.

15 Kenji Nagasaka and Jau-Shyong Shiue. Asymptotic positiveness of linear recurrence sequences. *Fibonacci Quart*, 28(4):340–346, 1990.

16 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 366–379, 2014.

17 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, April 2015. doi:10.1145/2766189.2766191.

18 Laurentius Petrus Dignus "Lou" van den Dries. *Tame Topology and O-minimal Structures*. Cambridge University Press, 1998.

19 Alex J. Wilkie. Schanuel's conjecture and the decidability of the real exponential field. In *Algebraic Model Theory*, pages 223–230. Springer Netherlands, Dordrecht, 1997.

New Lower Bounds and Upper Bounds for Listing Avoidable Vertices

Mingyang Deng
Massachusetts Institute of Technology, Cambridge, MA, USA

Virginia Vassilevska Williams
Massachusetts Institute of Technology, Cambridge, MA, USA

Ziqian Zhong
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We consider the problem of listing all avoidable vertices in a given n vertex graph. A vertex is avoidable if every pair of its neighbors is connected by a path whose internal vertices are not neighbors of the vertex or the vertex itself. Recently, Papadopolous and Zisis showed that one can list all avoidable vertices in $O(n^{\omega+1})$ time, where $\omega < 2.373$ is the square matrix multiplication exponent, and conjectured that a faster algorithm is not possible.

In this paper we show that under the 3-OV Hypothesis, and thus the Strong Exponential Time Hypothesis, $n^{3-o(1)}$ time is needed to list all avoidable vertices, and thus the current best algorithm is conditionally optimal if $\omega = 2$. We then show that if $\omega > 2$, one can obtain an improved algorithm that for the current value of ω runs in $O(n^{3.32})$ time. We also show that our conditional lower bound is actually higher and supercubic, under a natural High Dimensional 3-OV hypothesis, implying that for our current knowledge of rectangular matrix multiplication, the avoidable vertex listing problem likely requires $\Omega(n^{3.25})$ time. We obtain further algorithmic improvements for sparse graphs and bounded degree graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Avoidable Vertex, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.41

1 Introduction

The notion of an "avoidable" vertex in a graph has been studied since the 1970s in the context of minimal elimination orderings for Gaussian elimination [24, 26], though the name "avoidable" was first used by Beisegel et al. [9]. A vertex is avoidable if and only if a minimal elimination ordering can start from it.

In Gaussian elimination of sparse matrices, one iteratively selects a pivot, and then eliminates its row from the rest of the matrix, potentially creating new nonzero entries: i.e. fill-in. One seeks to find a good elimination ordering to minimize the fill-in, as the fill-in determines both the running time and the necessary storage. An iterative approach from the 1970s by [24, 26] was to pick an avoidable vertex (in a graph corresponding to the current sparse matrix), use it as a pivot, and repeat.

A single avoidable vertex always exists and can be found in linear time in the size of the matrix as shown by Beisegel, Chudnovsky, Gurvich, Milanic and Servatius [9]. By repeating n times (for an $n \times n$ matrix), one can complete the Gaussian elimination in $O(n^3)$ time; cubic time can be necessary, as the fill-in can make an originally sparse matrix into a dense one very quickly (though for some matrices this approach can still work well).

A natural question is, can one still use the minimal elimination orderings approach from the 1970s to find a minimal elimination ordering and hopefully perform Gaussian elimination in truly subcubic time $O(n^{3-\varepsilon})$ for $\varepsilon > 0$, without using the heavy Strassen-like techniques that achieve the fastest matrix multiplication algorithms nowadays [28, 12, 6]?

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 41; pp. 41:1–41:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The hope here is that instead of iteratively finding a new pivot n times, we can compute several pivots in batch faster and then use them in the Gaussian elimination process. As a first step, one would ask whether one can find all the (at most n) possible avoidable vertices in an n-node graph in truly subcubic time? Finding n avoidable vertices as the matrix graph changes due to fill-in, can intuitively be more complicated.

This motivates studying the problem of listing all avoidable vertices in a graph. Beisegel et al. [9] present further motivations for studying avoidable vertices, including faster clique algorithms in certain classes of graphs. Beisegel et al. [9] showed that every graph on at least two vertices contains at least two avoidable vertices that are at distance the diameter of the graph, and that two avoidable vertices can be found in linear time. They provided extensions for avoidable edges, and also used their results for avoidable vertices to provide fast algorithms for maximum weight clique detection in special classes of graphs.

While the algorithm of [9] can find an avoidable vertex in linear time, the authors only mention a very slow algorithm for finding the set of all avoidable vertices of a graph. Recently, Papadopolous and Zisis [25] provide algorithms for computing all avoidable vertices of an n vertex, m edge graph, running in time $O(\min\{n^{\omega+1}, n^2+m^2\})$, where $\omega < 2.37286$ is the exponent of square matrix multiplication [6].

A simpler definition than the one based on minimal elimination orderings is that v is avoidable if every two neighbors of v, are connected by a path that has no internal vertices that are v or neighbors of v. As [9] put it, if vertices represent people, then any two neighbors of an avoidable person v can communicate a secret without any direct acquaintances of v learning the secret.

A special case of avoidable vertices are *simplicial vertices*. A vertex is simplicial if its neighborhood is a clique. Simplicial vertices are very well studied in graph theory. For example, in the 1960s Dirac [17] showed that every chordal graph contains a simplicial vertex.

Kloks, Kratsch and Müller [20] showed how to list all simplicial vertices in an n-node graph in $O(n^\omega)$ time. It has been open for over 20 years whether a faster algorithm exists. (Spinrad [27] explicitly states finding such an algorithm as an open problem.) In the preliminaries we give a simple proof that the $O(n^\omega)$ time algorithm of [20] is optimal for *listing* all simplicial vertices, unless triangles can be found faster. It still remains open whether finding a single simplicial vertex can be done faster.

Papadopoulos and Zisis [25] state that they do not believe that the running times of their algorithms for listing avoidable vertices can be improved without resolving the open problem for simplicial vertices. In this paper we show that it is actually *possible* to improve the avoidable vertex listing running time slightly, and further provide fine-grained hardness reductions that point to concrete hurdles that need to be overcome to further improve the running time.

1.1 Our results

We provide new algorithms and fine-grained lower bounds for listing all avoidable vertices.

1.1.1 Fine-grained lower bounds

In Section 3, we provide a tight connection between Avoidable Vertex Listing and one of the central problems in fine-grained complexity, 3-Orthogonal Vectors.

In the k-Orthogonal Vectors problem (k-OV), one is given k sets of n Boolean Vectors each $S_1, \ldots, S_k \subseteq \{0,1\}^d$ in d dimensions, and one wants to determine whether there exist $a_1 \in S_1, \ldots, a_k \in S_k$ so that $\sum_{c=1}^{d} \prod_{i=1}^{k} a_i[c] = 0$, i.e. that $a_1, \ldots, a_k$ are orthogonal.

The k-OV Hypothesis of Fine-Grained Complexity (see e.g. [31]) states that there is no $O(n^{k-\varepsilon})$ time algorithm with $\varepsilon > 0$ that solves size n instances of k-OV when $d = \text{poly}\log(n)$, in the word-RAM model of computation with $O(\log n)$ bit words.

For any integer $k \geq 2$, the k-OV Hypothesis follows from the popular Strong Exponential Time Hypothesis (SETH) as shown by Williams [33]. The k-OV Hypothesis is even more believable than SETH, as even if SETH fails, the k-OV Hypothesis might still be true. SETH and the k-OV Hypothesis have been shown to imply a large variety of tight conditional lower bounds (see the survey [31] for some results). 3-OV in particular implies such bounds for graph diameter approximation [8, 23, 16, 14], various dynamic problems [1] and more.

Our first theorem is:

▶ **Theorem 1.** *Under the 3-OV Hypothesis, listing all avoidable vertices in an n vertex graph requires $n^{3-o(1)}$ time in the word-RAM model of computation with $O(\log n)$ bit words.*

One can think of this as a negative result. It gives a concrete limitation to using minimal elimination orderings to obtain a general truly subcubic time algorithm for Gaussian elimination.

The $O(n^{\omega+1})$ time algorithm of [25] would run in $O(n^3)$ time if $\omega = 2$ (as some believe). Thus, if $\omega = 2$, their algorithm would be optimal, under SETH and the 3-OV Hypothesis.

Nevertheless, ω might not be 2. Indeed, there has been quite a bit of work (e.g. [7, 10, 5, 4]) that shows that the known techniques for matrix multiplication cannot achieve $\omega = 2$. Can we have a conditional lower bound in terms of ω?

Here, we notice that the reduction used to prove Theorem 1 can also be used to tightly reduce to Avoidable Vertex Listing the 3-OV problem for vectors of dimension n, rather than polylogarithmic in n, which the standard 3-OV Hypothesis is about.

While 3-OV in polylogarithmic dimensions has a simple $n^{3+o(1)}$ time algorithm (compute the inner product of every triple of vectors), the fastest known algorithm for 3-OV in n dimensions is essentially the same as that for 4-clique [18]. It runs in $O(n^{\omega(1,2,1)})$ time where $\omega(1, 2, 1)$ is the exponent of multiplying an n by n^2 matrix by an n^2 by n matrix (see the preliminaries). The current best bound on this exponent is $\omega(1, 2, 1) < 3.251640$ [22].

We formulate the High Dimensional 3-OV Hypothesis that states that 3-OV for n vectors in n dimensions requires $n^{\omega(1,2,1)-o(1)}$ time on the word RAM with $O(\log n)$ bit words. This Hypothesis is the natural extension of the High Dimensional 2-OV Hypothesis used in prior work (e.g. [13]).

We then extend Theorem 1:

▶ **Corollary 2.** *Under the High Dimensional 3-OV Hypothesis, listing all avoidable vertices in an n vertex graph requires $n^{\omega(1,2,1)-o(1)}$ time on the word RAM with $O(\log n)$ bit words.*

Thus, with the current bounds on ω and $\omega(1, 2, 1)$, the algorithm of [25] runs in $O(n^{3.373})$, while our conditional lower bound is $n^{3.251-o(1)}$. This gap motivates the question:

In the case when $\omega > 2$, is there an algorithm for listing all avoidable vertices that runs faster than $O(n^{\omega+1})$? Can one achieve a running time closer to $O(n^{\omega(1,2,1)})$?

1.1.2 Faster Algorithms

Utilizing rectangular matrix multiplication techniques, in Section 4-7 we develop a method for listing all avoidable vertices with a running time strictly between $O(n^{\omega+1})$ and $O(n^{\omega(1,2,1)})$.

▶ **Theorem 3.** *All avoidable vertices in an m-edge, n-vertex graph can be listed in time*

$$O\left(\min\{m^{1.7}n^{0.2} + mn^{1+o(1)}, m^{0.977}n^{1.4+o(1)}, n^{3.32}\}\right).$$

In particular, in dense graphs the above running time is $O(n^{3.32})$ which is faster than $O(n^{\omega+1})$ for the current value of $\omega < 2.373$. In fact, for any value of ω greater than 2, our algorithm would run faster than in $O(n^{\omega+1})$ time, as rectangular matrix multiplication can always give us some savings. Our algorithm runs even faster in sparse graphs. Finally, we also address the case of bounded degree graphs in Appendix B and obtain improved running times for that case as well.

2 Preliminaries

Let $G = (V_G, E_G)$ be an undirected, unweighted graph. For a vertex $v \in V$, we use $N_G(v)$ to denote the neighborhood of v. For $X \subset V_G$, we define $G(X)$ to be the induced subgraph of X, namely the graph with vertex set X and edge set $\{uv|u, v \in X\} \cap E_G$, and define $G\backslash X$ to be $G(V_G\backslash X)$. We call u, v connected in a graph if there is a path connecting them. The connected components of a graph V are the maximal subsets of V_G such that any two vertices in the subset are connected. When G is clear from the context, we omit G as a subscript.

Avoidable Vertex Listing
Input: $G = (V_G, E_G)$
Task: Call a vertex $v \in V_G$ avoidable if for any $a \neq b \in N_G(v)$, there exists a path from a to b not containing v and any other neighbor of v ("avoiding" v and neighbors). Find all avoidable vertices.

Throughout the paper, unless otherwise noted, we denote $n = |V_G|$ and $m = |E_G|$, where G is the current graph.

There is an alternative definition which defines v as avoidable if and only if for any $x \neq y \in N(v)$, there exists $T \subseteq V_G$ where $\{v, x, y\} \subseteq T$ and $G(T)$ is a single simple cycle (e.g. [9]). We can see both definitions are equivalent by taking T as the shortest valid path from a to b in the first definition.

A vertex v in G is *simplicial* if for every pair of vertices $x, y \in N(v)$, (x, y) is an edge.

▶ **Proposition 4** ([20]). All simplicial vertices of an n node graph can be listed in $O(n^\omega)$ time.

Proof. The following simple $O(n^\omega)$ time algorithm lists all simplicial vertices of G: Define A as the $n \times n$ adjacency matrix of G, where $A[i,j] = 1$ if (i,j) is an edge of G, and $A[i,j] = 0$ otherwise. Let B be the $n \times n$ matrix with $B[i,j] = 1$ if $i \neq j$ and (i,j) is not an edge of G; let $B[i,j] = 0$ otherwise. Compute the matrix product $C = ABA$ in $O(n^\omega)$ time. For a vertex i, $C[i,i]$ is nonzero iff there exist distinct neighbors j, k of i for which (j,k) is not an edge. Thus $C[i,i] = 0$ if and only if i is a simplicial vertex. Thus by going through the diagonal of C, we can in additional $O(n)$ time list all simplicial vertices of G. ◀

It is not hard to show that listing all simplicial vertices is at least as hard as Triangle detection. This means that the problem probably needs $n^{\omega-o(1)}$ time, unless one can detect triangles faster. Moreover via [32], we get that there is a truly subcubic time combinatorial fine-grained reduction from Boolean Matrix Multiplication to listing all simplicial vertices. Thus any subcubic algorithm for the problem likely requires matrix multiplication.

▶ **Proposition 5.** If one can list all simplicial vertices of a graph in $T(n)$ time, then one can detect a triangle in an n node graph in $O(T(n))$ time.

Proof. Let $G = (V, E)$ be a graph in which we want to find a triangle. Create a new graph G' as follows. Every $v \in V$ has two copies v_1 and v_2 in G'. Let V_1 and V_2 be the sets of vertices with the corresponding subscripts. For every edge (u, v) of G, add edges (u_1, v_2) and (u_2, v_1) in G'. Finally, for every non-edge (x, y) of G where $x \neq y$, add an edge (x_2, y_2) to G'.

Suppose v_1 is a simplicial vertex of G'. All neighbors of v_1 are in V_2, and the only way for v_1 to be simplicial is if in G no pair of its neighbors has an edge between them, i.e. v_1 does not appear in a triangle. On the other hand, any vertex that is not simplicial must have a pair of neighbors in G that are connected by an edge. Thus, if we can list all simplicial vertices, we can determine if a vertex of V_1 is not in the list, and that will tell us whether G has a triangle. ◀

The 3-OV problem (see e.g. [31]) is defined as follows[1].

3-OV
Input: Three sets of Boolean vectors $A, B, C \subseteq \{0,1\}^d$ where $|A| = |B| = |C| = n$
Task: decide whether there are $a \in A, b \in B, c \in C$ so that their generalized inner product is 0, i.e. $\sum_{i=1}^{d} a_i b_i c_i = 0$.

The 3-OV Hypothesis (see e.g. [31]), states that in the word-RAM model with $O(\log n)$ bit words, 3-OV requires $n^{3-o(1)}$ even if d is polylogarithmic.

A now well-known result that follows from the seminal work of Williams [33] is that the 3-OV Hypothesis follows from the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, Zane and Calabro [11, 19].

In the high dimensional version of 3-OV, when the dimension d can be large, the fastest algorithm for the problem is no longer cubic. In particular, when $d = O(n)$, the fastest known algorithm runs in $O(n^{\omega(1,2,1)}) \leq O(n^{3.252})$ time. The running time $O(n^{\omega(1,2,1)})$ is a bit faster than $O(n^{\omega+1})$ as long as $\omega > 2$, and is $O(n^3)$ if $\omega = 2$.

We give the folklore algorithm here for completeness.

▶ **Proposition 6** (Folklore). 3-OV on vectors of dimension $d = O(n)$ can be solved in $O(n^{\omega(1,2,1)})$ time.

Proof. Let A and B be the given sets of d-dimensional vectors. Without loss of generality, assume that $d = n$. Create an $n^2 \times n$ matrix Boolean D such that for $a \in A, b \in B$, $i \in [n]$, we have $D[(a,b), i] = 1$ if and only if $a_i = b_i = 1$. Let C' be the $n \times n$ matrix which for $i \in [n], c \in C$, $C'[i, c] = 1$ if and only if $c_i = 1$. Then $DC'[(a,b), c]$ contains a 0 if and only if the generalized inner product of a, b, c is 0. ◀

Prior work (see e.g. [13, 3]) conjectured that the high dimensional variant (when $d = O(n)$) of 2-OV requires $n^{\omega-o(1)}$ time. A natural extension of this assumption is that the matrix multiplication based running time for 3-OV is best possible.

▶ **Definition 7.** *The High Dimensional 3-OV Hypothesis states that in the word-RAM model with $O(\log n)$ bit words, 3-OV in n dimensions requires $n^{\omega(1,2,1)-o(1)}$ time.*

Our hypothesis above is also related to a popular hypothesis (e.g. [15, 31]) that the best known running time for 4-Clique is best possible. The best known algorithm for 4-Clique by [20] is analogous to the best known algorithm for high-dimensional 3-OV above, and it is natural to conjecture that if this algorithm is best for 4-Clique, it could also be best for 3-OV.

[1] An equivalent definition has as an input a single set of vectors and we want to find a triple of orthogonal vectors in the set.

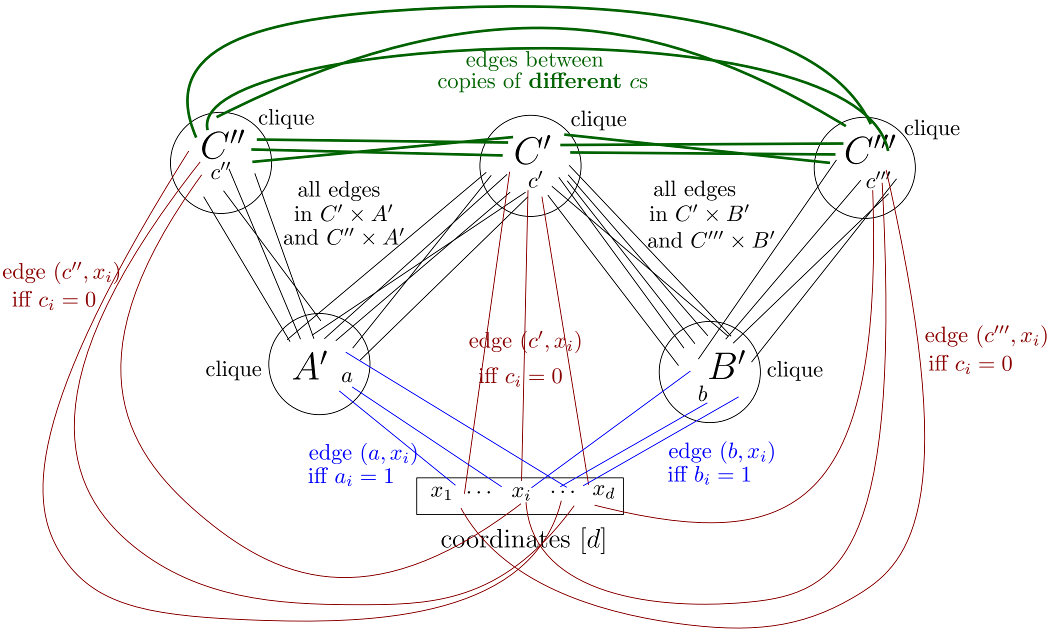


Figure 1 A depiction of the construction from 3-OV to Avoidable Vertex Listing.

3 Conditional Lower Bound

In this section, we derive a conditional lower bound for the problem based on 3-OV-hardness. We show that an instance of 3-OV of size n can be reduced to the Avoidable Vertex Listing problem in a graph of $O(n)$ vertices. Therefore Avoidable Vertex Listing requires $\Omega(n^{3-o(1)})$ time, under the 3-OV Hypothesis and the Strong Exponential Time Hypothesis.

Construction

Let an instance $A, B, C \subseteq \{0,1\}^d$ of 3-OV be given. We create a graph G from it as follows. The construction is depicted in figure 1.

For every vector $a \in A$, we create a node a' in G. Denote by A' the set of nodes formed by all such a's. Similarly, for every vector $b \in B$, we create a node b' in G, forming a set of nodes B', and for every vector $c \in C$, we create a node c', forming a set of nodes C'.

We make each of A', B', C' into a clique by adding an edge between every pair of nodes in it. Also, we add edges between each node of C' and every node in $A' \cup B'$.

We add d "coordinate nodes" $x_1, \ldots, x_d$, corresponding to each coordinate. For each $a \in A$ and $i \in [d]$, add an edge (a', x_i) if $a_i = 1$. Similarly, for each $b \in B$ and $i \in [d]$, add an edge (b', x_i) if $b_i = 1$. For each $c \in C$ and $i \in [d]$, we add an edge (c', x_i) if $c_i = 0$. Notice we treat C differently from A and B: we add edges for C if the corresponding bit is not set but we add edges for A, B if the bit is set.

For each $c \in C$, we add two more new nodes c'' and c''', forming sets C'' and C''' respectively. We add edges from each c'' to all of A', and edges from each c''' to all of B'. We add edges from both c'' and c''' to all x_i for which $c_i = 0$. For every $p \neq q \in C$, we add edges $(p', q'), (p', q''), (p', q'''), (p'', q''), (p'', q'''), (p''', q''')$. In other words, we connect all pairs of nodes in $C' \cup C'' \cup C'''$ corresponding to different vectors in C.

Proof of Correctness

We now prove the correctness of our construction. Specifically, for every $c' \in C'$ corresponding to vector $c \in C$, we will show that c' is not avoidable in G if and only if there exists $a \in A, b \in B$ such that $a \cdot b \cdot c = 0$.

Take some $c' \in C'$. Let X_c be the set of coordinate nodes x_i for which $c_i = 0$.

First, consider a pair of the neighbors of c' that is not in $A' \times B'$. We'll show such a pair is connected without visiting c' and other neighbors of c'. Since a neighbor of c' can either fall in A', B', X_c or $C' \cup C'' \cup C'''$, there are several cases:

- If the pair contains a vertex $a' \in A'$, let it be (a', t). When $t \in A' \cup C' \cup C''$, there is a direct edge between t and a'. When $t \in C'''$, we must have $t = e'''$ with $e \neq c$. Thus there is a path from a' to c'' (which is not a neighbor of c') to t. When $t \in X_c$, there is also a path from a' to c'' (which is not a neighbor of c') to t.
- Similarly, if the pair of neighbors contains a vertex $b' \in B'$, the pair is also connected.
- If the pair of neighbors (e, f) are in $X_c \cup C' \cup C'' \cup C'''$, then these nodes are not copies of c', so they have a path through $c'' \in C''$.

Thus every pair of neighbors of c' except for those in $A' \times B'$ has either an edge between them or a path through non-neighbors of c'.

Consider now some $a' \in A', b' \in B'$. If there is a path from a' to b' through non-neighbors of c', then this path can only go through x_i such that $c_i = 1$. The path cannot go through c'' or c''' as those nodes only have neighbors that are also neighbors of c'. Hence the only possible paths from a' to b' through non-neighbors of c' are of the form a' to x_i to b' where $a_i = b_i = 1 = c_i$. Thus, there is a neighbor pair of c' that is not connected when c' and its neighbors are removed if and only if there are $a \in A, b \in B$ so that $a \cdot b \cdot c = 0$.

Proof of Theorem 1, Corollary 2. Any algorithm for Avoidable Vertex Listing can solve High Dimensional 3-OV in roughly the same time: we convert the given instance to the aforementioned graph (which takes $O(n^2 + nd)$ time), run Avoidable Vertex Listing on the graph, and check if there is any non-avoidable c'. The converted graph has $O(n + d)$ vertices. Thus, if avoidable vertices for n-node graphs could be listed in $O(n^\alpha)$ time, so does n dimensional 3-OV. ◀

4 Basic Algorithm

In this section, we present a basic algorithm for Avoidable Vertex Listing, which performs a series of Boolean matrix multiplications. The approach is similar to the method of Papadopolous and Zisis [25].

Let $G = (V, E)$ be a given graph, an instance of Avoidable Vertex Listing.

For each $v \in G$, we check if v is avoidable in two steps. We first find all the connected components $C(v)$ in $G \backslash \{v\} \backslash N_G(v)$. Then for every $x \neq y \in N(v)$, we check if there exists $a \in N(x), b \in N(y)$ such that a, b are in the same connected component in $G \backslash \{v\} \backslash N_G(v)$.

The first step takes $O(nm)$ time via e.g., breadth-first search. For the second step, we can set a matrix A with entries labeled $N(v) \times C(v)$, where $A[i, j] = 1$ iff $N(i) \cap j \neq \emptyset$. Then it suffices to check if $A \times A^T$ (where $\times$ corresponds to Boolean matrix multiplication) has a zero outside of its diagonal. Calculating matrix multiplication directly takes $O(n^\omega)$ time per vertex, thus we can solve the problem in $O(n^{\omega+1})$ time.

With a more careful analysis, we can show the above algorithm is in fact of $O(mn^{\omega-1})$ time. For each vertex v, we need to do a matrix multiplication of size $(\deg v, n) \times (n, \deg v)$, which can be done in $O((\deg v)^\omega (n/\deg v)) = O(n(\deg v)^{\omega-1})$ time. Notice that $\sum_{v \in V} \deg v = 2m$, by convexity of function $x^{\omega-1}$, the total time complexity $O(\sum_{v \in V} n(\deg v)^{\omega-1}) \leq O((m/n)n^\omega) = O(mn^{\omega-1})$ - minimized when $2m/n$ vertices each have degree n.

For sparse graphs, we can do even better. Notice that every entry in A corresponds to an edge in G, so there are only $O(m)$ non-zero entries in A. We can use sparse matrix multiplication [34] to perform the multiplication, taking $\widetilde{O}((\deg v)^{1.2}m^{0.7} + (n \deg v)^{1+o(1)})$ time[2]. By convexity, the function is maximized when $\deg v \in \{0, n\}$, so $\widetilde{O}(\sum_v((\deg v)^{1.2}m^{0.7} + (n \deg v)^{1+o(1)})) = \widetilde{O}((m/n)(n^{1.2}m^{0.7} + (n^2)^{1+o(1)})) = \widetilde{O}(m^{1.7}n^{0.2} + mn^{1+o(1)})$, which works better when m is small.

5 Algorithm for Dense Graphs and High Degree Vertices

We first show how to handle dense graphs and high degree vertices. We achieve this by dividing paths into short and long paths, and utilizing rectangular matrix multiplication.

Let the set of vertices we consider be $V_h \subseteq V$. For every $v \in V_h$ and $p, q \in N(v)$, we call a path from p to q passing no other neighbours of v *short* if its length $\leq \ell$ where ℓ is a parameter to be set, or *long* if its length $> \ell$. We treat short paths and long paths separately.

5.1 Short paths

On short paths, for every $l \leq \ell$, we want to compute for every $v \in V_h$, $p, q \in V$, if there is a path from p to q in G that has at most l edges and does not use any internal nodes that are neighbors of v or v itself. Let $X_v^l(p, q) = 1$ if so and $X_v^l(p, q) = 0$ otherwise. Notice here we do not require $p, q \in N(v)$.

We compute this matrix incrementally. Let A be the adjacency matrix of G and suppose we have calculated X_v^{l-1}. For some $X_v^l(p, q) = 1$, consider the last node r in the corresponding path from p to q, we must have $X_v^{l-1}(p, r) = 1$, $A(r, q) = 1, r \notin N(v) \cup \{v\}$, and vice versa.

Therefore, we can construct a $(n \cdot |V_h|) \times n$ matrix C where

$$C[(v, p), r] = 1 \text{ if } X_v^{l-1}(p, r) = 1 \text{ and } r \notin N(v) \cup \{v\}, \text{ and } C[(v, p), r] = 0 \text{ otherwise,}$$

then $X_v^l(p, q) = 1$ if and only if $(C \times A)[(v, p), q] \neq 0$.

Thus X_v^l can be computed from X_v^{l-1} in $O(M(n|V_h|, n, n))$[3] time with rectangular matrix multiplication (e.g. [22]). The computation of X_v^ℓ would thus take overall time $O(\ell M(n|V_h|, n, n))$.

5.2 Long paths

For $p, q \in V$ and $v \in V_h$, suppose that there is a path from p to q with at least ℓ edges that does not use internal vertices that are v or neighbors of v. Let $P_v(p, q)$ be such a path.

We randomly sample a set of vertices $S \subseteq V$ of size $10n/\ell \log n$. We claim that S "splits" every $P_v(p, q)$ into pieces of length $\leq \ell$ with high probability. (This lemma is well-known and widely used. We include it for completeness.)

▶ **Lemma 8.** *Suppose the path is of nodes $t_0, t_1, \cdots, t_m$ where $t_0 = p$, $t_m = q$, $m \geq \ell$. For every i, call $t_i, t_{i+1}, \cdots, t_{i+\ell-1}$ an ℓ-length segment. For a randomly sampled $S \subseteq V$ where $|S| = \lceil 10n/\ell \log n \rceil$, every ℓ-length segment of the path contains an element from S with probability $\geq 1 - n^{-9}$.*

[2] $\widetilde{O}$ hides logarithmic factors. Our matrix multiplication is rectangular with size $(\deg v, n, \deg v)$, but the original proof also holds for this case.

[3] $M(a, b, c)$ is the time complexity of multiplying (possibly rectangular) matrices of size $a \times b$ and $b \times c$.

Proof. For any ℓ elements, the probability that none of them lies in S is $\binom{n-\ell}{|S|}/\binom{n}{|S|} = \prod_{i=0}^{|S|-1} \frac{n-\ell-i}{n-i} \leq (1-\frac{\ell}{n})^{|S|} \leq (1-\frac{\ell}{n})^{n/\ell \cdot 10 \log n} \leq n^{-10}$. The result then follows from a union bound. ◀

By the lemma, with high probability S splits each $P_v(p,q)$ into consecutive pieces of length at most ℓ: the first is from p to some $s_1 \in S$, then a piece from s_1 to some $s_2 \in S$, …, a piece from some $s_{t-1} \in S$ to $s_t \in S$, and finally a piece from s_t to q. Since we have computed for every pair of vertices whether there is a path of length at most ℓ between them not using neighbors of v, we can use this information to compute the full paths as follows.

For every $v \in V_h$, we build a graph G_v whose vertices are the nodes S' of S that are not v or neighbors of v, and there is an edge in G_v between a and b if $X_v^\ell(a,b) = 1$ (i.e. there is a path of length at most ℓ avoiding the neighbors of v and v). We compute the transitive closure of G_v. Since G_v is undirected, this can be done in time $\widetilde{O}(n^2/\ell^2)$ for each v.

Let T_v be the transitive closure matrix, compute $Y_v = X_v^\ell[\cdot, S'] \times T_v \times X_v^\ell[S', \cdot]$ which accounts for the first and the final piece in the path. By the above argument, this will tell us for every pair of nodes if there is a path between them avoiding the neighbors of v with high probability.

The running time of this over all v is $\widetilde{O}(|V_h| \cdot M(n, n/\ell, n))$, where $M(r,s,t)$ is the time to multiply an $r \times s$ by an $s \times t$ matrix.

5.3 Overall runtime on dense graphs

Combining the two algorithms, the final running time would be

$$\widetilde{O}(\ell M(n|V_h|, n, n) + |V_h| \cdot M(n, n/\ell, n)).$$

Take $V_h = V$, we immediately get an algorithm for dense graphs. Let $\ell = n^a$,

$$\widetilde{O}(\ell M(n|V_h|, n, n) + |V_h| \cdot M(n, n/\ell, n)) = \widetilde{O}(n^{\max(\omega(1,2,1)+a, 1+\omega(1,1,1-a))})$$

Using bounds from [22], let $a = 0.0682157$, we have $\omega(1,1,1-a) < 2.319856$, $\max(\omega(1,2,1)+a, 1+\omega(1,1,1-a)) < 3.319856$, so the running time exponent of our algorithm is < 3.320, slightly better than $\omega + 1 \approx 3.373$. As long as $\omega > 2$, this algorithm will benefit from rectangular matrix multiplication and have a complexity of $O(n^{\omega-\varepsilon})$ for some $\varepsilon > 0$.

6 Algorithm for Low Degree Vertices

The result in the previous section works well for dense graphs or high degree vertices. One simple improvement is to combine the previous algorithm with sparse matrix multiplication as in Section 4. In the following, we show we can actually do a bit better by a carefully designed counting method.

We call a vertex "low-degree" if it has degree $\leq d$. Let the set of low-degree vertices be $V_l \subseteq V$. Again for every $v \in V_l$ and $p, q \in N(v)$, we call a path from p to q passing no other neighbours of v or v short or long depending on whether its length is $\leq L$.

For long paths, we calculate connected components of $G_v = G \backslash \{v\} \backslash N_G(v)$ for each $v \in V_l$ and perform the matrix multiplication as in Section 4, but here we can keep only connected components with size $\geq L$, since otherwise the shortest path will have length $\leq L$ and a short path could instead be considered. As there are at most n/L such components, we only need to perform matrix multiplications of size $(\deg(v), n/L, \deg(v))$, in time $M(\deg(v), n/L, \deg(v))$.

Now we show how to handle the short paths. We count the number of short avoiding paths (not necessarily simple) modulo some random prime p and check if it equals to zero. Since the number of paths of a given length $< n$ is $\leq n^n$, it has no more than n different prime factors $\geq n$. We randomly sample p as a prime in (n^5, n^6), by prime number theorem there are $\Omega(n^{5.99})$ primes within the range, so with probability $\geq 1 - O(n^{-4.99})$, if the number of paths is non-zero, it would remain non-zero modulo p. Since we need to check whether the number of paths is zero at most n^3 times, by union bound, all the checks would be correct with high probability.

For each $i \in [1, L]$ and each $u, v \in G$, we compute the number of paths (not necessarily simple) from u to v using exactly i steps, modulo p. This could be done by matrix multiplication in time $\widetilde{O}(Ln^\omega)$.

For a vertex v with neighbors $v_1, v_2, \cdots, v_s$, let $v_0 = v$. Construct the following matrices F_i where

$$F_i[x, y] = \text{number of paths from } v_x \text{ to } v_y \text{ with length } i \text{ without visiting other } v\text{'s} \pmod p$$

To compute F, let G_i be the matrix where

$$G_i[x, y] = \text{the number of paths from } v_x \text{ to } v_y \text{ with length } i \pmod p$$

G's can be directly retrieved from the computed matrices. For an invalid path that visits neighbors of v or v midways (which are the paths counted in G but not in F), we consider the number of steps j took before the first such visit, and the contribution will be $F_j G_{i-j}$. Therefore we have $F_i \equiv G_i - \sum_{1 \leq j < i} F_j G_{i-j} \pmod p$.

To compute $F_1, F_2, \cdots, F_L$ from $G_1, G_2, \cdots, G_L$, we can use the standard divide-and-conquer technique (see e.g. [30]). To compute $F_l, F_{l+1}, \cdots, F_r$, let $m = \lfloor (l + r)/2 \rfloor$, we first recursively compute $F_l, F_{l+1}, \cdots, F_m$, then calculate the contribution of $F_l, F_{l+1}, \cdots, F_m$ to $F_{m+1}, F_{m+2}, \cdots, F_r$, then recursively compute $F_{m+1}, F_{m+2}, \cdots, F_r$.

To calculate the contribution, for every $i \in [m + 1, r]$, we need to calculate $\sum_{j=l}^{m} F_j G_{i-j}$. By introducing variable x, the problem can be formulated as a polynomial multiplication:

$$\begin{aligned}\sum_{j=l}^{m} F_j G_{i-j} &= \sum_{j=0}^{m-l} F_{j+l} G_{i-l-j} \\ &= [x^{i-l}] \left(\sum_{j=0}^{m-l} F_{j+l} x^j \right) \left(\sum_{j=0}^{r-l} G_j x^j \right)^{4}\end{aligned}$$

Therefore we can let $F' = \sum_{j=0}^{m-l} F_{j+l} x^j$, $G' = \sum_{j=0}^{r-l} G_j x^j$, compute their matrix product modulo p, extract coefficients of x to get the contributions. Entries in these matrices are polynomials in $F_p[x]$ with degree $O(r - l)$, and operations on such polynomials can be done in $\widetilde{O}(r - l)$ time via Fast Fourier Transform (e.g. [2]). Each divide-and-conquer process takes $\widetilde{O}((r - l)s^\omega)$ time and computing $F_1, F_2, \cdots, F_L$ takes $\widetilde{O}(Ls^\omega)$ time in total.

Combining the algorithm for short and long paths, we now have an algorithm that takes $\widetilde{O}(Ln^\omega)$ time for pre-computation and spends $\widetilde{O}(M(\deg(v), n/L, \deg(v)) + L \deg(v)^\omega)$ time for each vertex $v \in V_L$.

[3] $[x^c]$ extracts coefficient of x^c in a polynomial $p(x)$.

7 Final Algorithm

Finally, we combine the ideas in Section 5 and Section 6. We set a parameter d, use the algorithm in Section 5 to treat vertices with degree $> d$ and use the algorithm in Section 6 to treat vertices with degree $\le d$.

The number of vertices with degree $> d$ is $O(m/d)$, so the combined complexity would be

$$\widetilde{O}(\ell M(nm/d,n,n)+(m/d)\cdot M(n,n/\ell,n)+Ln^{\omega}+\sum_{\deg(v)\le d}(M(\deg(v),n/L,\deg(v))+L\deg(v)^{\omega})).$$

By convexity, it would be maximized when $\deg(v)\in\{0,d\}$ for low-degree vertices. The final complexity would then be

$$\widetilde{O}(\ell M(nm/d,n,n)+Ln^{\omega}+(m/d)(M(n,n/\ell,n)+M(d,n/L,d)+Ld^{\omega})).$$

Optimizing with respect to d,ℓ,L, we find the complexity to be $O(m^{0.977}n^{1.4+o(1)})$ (Appendix A).

Combining the algorithms discussed in Section 4, Section 5, Section 7, we arrive at the following theorem.

▶ **Theorem 9.** *Avoidable Vertex Listing can be solved in time*

$$O(\min\{m^{1.7}n^{0.2}+mn^{1+o(1)},m^{0.977}n^{1.4+o(1)},n^{3.32}\}).$$

8 Conclusion

In this paper, we present a new fine-grained reduction from 3OV to Avoidable Vertex Listing, providing essentially cubic and even supercubic (if $\omega > 2$) conditional hardness for the problem. We also present new improved algorithms for the problem, combining techniques such as rectangular matrix multiplication, sparse matrix multiplication, the principle of inclusion-exclusion and counting.

As analyzed in Section 4, the algorithm by Papadopoulos and Zisis [25] solves Avoidable Vertex Listing in $O(mn^{\omega-1})$ time, and our algorithm improves upon this bound when $\omega > 2$. When $\omega = 2$, that algorithm runs in $O(mn)$ time, and by our 3OV lower bound, this running time is conditionally optimal when $m = \Omega(n^2)$. An interesting problem would be, when $m = o(n^2)$ and $\omega = 2$, can we do better than $\Omega(nm)$? While we can break this bound in regular or random graphs[4] (Appendix B), it remains open for more general graphs.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014.

2 Ramesh C. Agarwal and Charles S. Burrus. Fast convolution using Fermat number transforms with applications to digital filtering. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-(2):87–97, 1974.

[4] Random graphs have degrees of nodes bounded by $\widetilde{O}(m/n)$ with high probability.

3 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

4 Josh Alman. Limits on the universal method for matrix multiplication. *Theory Comput.*, 17:1–30, 2021.

5 Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 580–591. IEEE Computer Society, 2018.

6 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, pages 522–539. SIAM, 2021.

7 Andris Ambainis, Yuval Filmus, and François Le Gall. Fast matrix multiplication: Limitations of the coppersmith-winograd method. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 585–593. ACM, 2015.

8 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Toward tight approximation bounds for graph diameter and eccentricities. *SIAM J. Comput.*, 50(4):1155–1199, 2021.

9 Jesse Beisegel, Maria Chudnovsky, Vladimir Gurvich, Martin Milanic, and Mary Servatius. Avoidable vertices and edges in graphs. In *Algorithms and Data Structures – 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 2019.

10 Jonah Blasiak, Thomas Church, Henry Cohn, Joshua A. Grochow, and Chris Umans. On cap sets and the group-theoretic approach to matrix multiplication. *Discrete Analysis*, 3:27pp., 2017.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified k-cnf. *Algorithmica*, 65(4):817–827, 2013.

12 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

13 Mina Dalirrooyfard and Jenny Kaufmann. Approximation algorithms for min-distance problems in dags. 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 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

14 Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. In *Proceedings of FOCS 2021*, page to appear, 2021.

15 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles. *SIAM J. Comput.*, 50(5):1627–1662, 2021.

16 Mina Dalirrooyfard and Nicole Wein. Tight conditional lower bounds for approximating diameter in directed graphs. 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 1697–1710. ACM, 2021.

17 G.A. Dirac. On rigid circuit graphs. *Abh.Math.Semin.Univ.Hambg.*, 25:71–76, 1961.

18 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

20 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000.

21 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC 2014 – Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, New York, 2014.

22 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. 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 1029–1046. SIAM, 2018.

23 Ray Li. Settling SETH vs. approximate sparse directed unweighted diameter (up to (NU)NSETH). 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 1684–1696. ACM, 2021.

24 Tatsuo Ohtsuki, Lap Kit Cheung, and Toshio Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *Journal of Mathematical Analysis and Applications*, 54(3):622–633, 1976.

25 Charis Papadopoulos and Athanasios Zisis. Computing and listing avoidable vertices and paths, 2021. arXiv:2108.07160.

26 Donald J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

27 Jeremy Spinrad. Some open problems. URL: http://dts-web1.it.vanderbilt.edu/~spinrajp//open.html.

28 Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, 1969.

29 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 343–350. ACM, New York, 2000.

30 Joris van der Hoeven. Lazy multiplication of formal power series. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, pages 17–20, New York, NY, USA, 1997. Association for Computing Machinery.

31 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018.

32 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

33 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

34 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005.

A Missing Calculation in Section 7

Suppose $m = n^\alpha$ $(\alpha \in [1,2])$, let

$$\begin{cases} d = n^{0.015122\alpha^2 - 0.07237\alpha + 1.056742} \\ \ell = n^{-0.0364\alpha^2 + 0.17612\alpha - 0.138437} \\ L = \ell^{0.180156}. \end{cases}$$

When $t \in [1,2]$, $\omega(1,1,t) < 0.0625t^2 + 0.697t + 1.614$ (this can be verified by bounds in [22], [21] and convexity). Let $f(t) = 0.0625t^2 + 0.697t + 1.614$.

$$\begin{aligned}\ell M(nm/d,n,n) &= n^{-0.0364\alpha^2+0.17612\alpha-0.138437} M(n^{-0.015122\alpha^2+1.07237\alpha-0.056742},n,n)\\ &= O(n^{-0.0364\alpha^2+0.17612\alpha-0.138437+\omega(1,1,-0.015122\alpha^2+1.07237\alpha-0.056742)})\\ &= O(n^{-0.0364\alpha^2+0.17612\alpha-0.138437+f(-0.015122\alpha^2+1.07237\alpha-0.056742)})\\ &= O(n^{0.977\alpha+1.4})\end{aligned}$$

The last step can be proved by calculating the extrema of the function $-0.0364\alpha^2+0.17612\alpha-0.138437+f(-0.015122\alpha^2+1.07237\alpha-0.056742)-(0.977\alpha+1.4)$, which stays negative in $[1,2]$.

Similarly we can verify the following by computing the extrema of functions:

$$Ln^\omega = n^{0.180156(-0.0364\alpha^2+0.17612\alpha-0.138437)+\omega} = O(n^{0.977\alpha+1.4})$$

$$\begin{aligned}(m/d)M(n,n/\ell,n) =&O(n^{\alpha-\log_n d+\omega(1,1,\log_n(n/\ell))})\\ =&O(n^{\alpha-\log_n d+f(1-\log_n \ell)})\\ =&O(n^{0.977\alpha+1.4})\end{aligned}$$

$$\begin{aligned}(m/d)M(d,n/L,d) =&O(n^{\alpha-\log_n d+\log_n d\cdot\omega(1,1,\log_n(n/L)/\log_n d)})\\ =&O(n^{\alpha-\log_n d+\log_n d\cdot f((1-\log_n L)/\log_n d)})\\ =&O(n^{0.977\alpha+1.4})\end{aligned}$$

$$(m/d)Ld^\omega = O(n^{\alpha-\log_n d+\log_n L+\omega\log_n d}) = O(n^{0.977\alpha+1.4})$$

Thus the whole complexity is bounded by $O(n^{0.977\alpha+1.4}) = O(m^{0.977}n^{1.4+o(1)})$.

B An $\widetilde{O}(nd^3)$ Time Algorithm for Bounded Degree Graphs

Suppose every vertex in the graph G has degree not exceeding d, we provide a simple algorithm listing all avoidable vertices of G in $\widetilde{O}(nd^3)$ time.

We maintain a fully-dynamic graph connectivity oracle X supporting addition and removal of edges [29]. In the beginning, we add all edges in G to X.

For each vertex v, to judge if v is avoidable, we remove all edges adjacent to v or neighbors of v in X. Then we enumerate every two neighbors of v: p and q. Assume p and q is not directly connected, we add all edges adjacent to p and q, but not to v or neighbors of v, to X, and in X check if p and q is now connected. After checking we remove these edges and consider the next pair. After considering each vertex v we add back initially removed edges.

In this way, for each vertex $O(d^3)$ edge modifications (enumerating neighboring pairs and their adjacent edges) to X will be performed, thus taking $\widetilde{O}(nd^3)$ time in total.

Constant-Factor Approximation Algorithm for Binary Search in Trees with Monotonic Query Times

Dariusz Dereniowski
Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Poland

Izajasz Wrosz
Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Poland
Intel, Gdańsk, Poland

Abstract

We consider a generalization of binary search in linear orders to the domain of weighted trees. The goal is to design an adaptive search strategy whose aim is to locate an unknown target vertex of a given tree. Each query to a vertex v incurs a non-negative cost $\omega(v)$ (that can be interpreted as the duration of the query) and returns a feedback that either v is the target or the edge incident to v is given that is on the path towards the target. The goal of the algorithm is to find a strategy that minimizes the worst-case total cost. We propose a constant-factor approximation algorithm for trees with a monotonic cost function. Such function is defined as follows: there exists a vertex r such that for any two vertices u, v on any path connecting r with a leaf it holds that if u is closer to r than v, then $\omega(u) \geq \omega(v)$. The best known approximation algorithm for general weight functions has the ratio of $\mathcal{O}(\sqrt{\log n})$ [Dereniowski et al. ICALP 2017] and it remains as a challenging open question whether constant-factor approximation is achievable in such case. This gives our first motivation towards considering monotonic cost functions and the second one lies in the potential applications.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases binary search, graph search, approximation algorithm, query complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.42

Funding Work partially supported by National Science Centre (Poland) grant number 2018/31/B/ST6/00820.

1 Introduction

We study a natural generalization of the classical binary search problem, where the algorithm is asked to locate a specific element in a sorted array of keys with as few comparisons as possible [20]. We generalize the binary search in two dimensions. Instead of arrays, we consider trees, with the keys represented as vertices of the tree. Additionally, we allow the cost of comparisons to vary across the vertices of the tree. As a result, instead of minimizing the worst-case number of queries, our objective is to minimize the worst-case total cost of the search. This problem has been introduced for trees in [26] and for general graphs in [15]. There are two characteristics of such search algorithms: the computational complexity of calculating a search strategy and the worst-case cost called the *query complexity*.

As a fundamental problem in computer science, binary search in trees and graphs has numerous practical applications in data management systems or scheduling of parallel processes (through graph coloring), machine learning, and other fields. We discuss the relevant practical problems later on in more detail.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 42; pp. 42:1–42:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

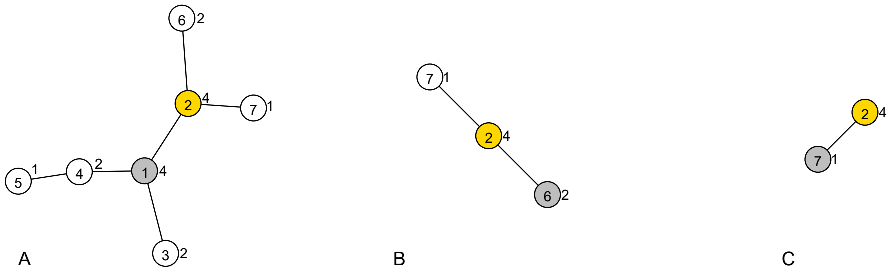


Figure 1 Binary search on weighted trees. The input tree A contains a target vertex (2), whose position is unknown to the algorithm. A series of queries is performed to the vertices 1, 6, and 7, which incurs a cost equal to 4, 2 and 1, for each query respectively. As a result of the subsequent queries, subtrees B and C are generated that consist of all vertices that may still contain the target, given the information revealed by previous queries.

1.1 Problem statement

An a priori unknown target vertex t in a known input tree $T = (V, E, \omega)$, with a query cost function $\omega\colon V(T) \to \mathbb{R}_+$, should be located by an algorithm via a series of queries. Each query selects a vertex v and as an answer receives information that either v is the target (which completes the search), or otherwise it is given an edge $\{v, u\}$ such that u lies on a path between v and t. (Note that in the case of a tree the path is unique but for general graphs, any of the shortest paths is provided, see [15].) The answer is generated at the cost of $\omega(v)$ and we want to design an algorithm that finds the target with the minimum cost in the worst case. More precisely, once the target has been returned by the algorithm, the cost of the search equals the sum of the costs of all queries that have been asked, and the performance of the algorithm is the maximum search cost taken over all vertices as potential targets. One might think about this search process that with each query the search is narrowed to a subtree of T that can still contain the target. The process is *adaptive* in the sense that the choice of the subsequent elements to be queried depends on the locations and answers of the previous queries. Also, we consider only *deterministic* algorithms, where given a fixed sequence of queries performed so far (and the information obtained), the algorithm selects always the same element as the next query for a given tree. E.g., the optimal algorithm for the classic binary search problem on paths that always queries the median element is adaptive and deterministic. It can be equivalently stated that the algorithm calculates a *strategy* for the given input tree, which encodes the queries to be performed for each potential target.

1.2 Related work and earlier techniques

While in this work we analyze queries that ask questions about vertices of a tree (vertex query model), an edge query model has been also studied. In the latter, the answer to each queried edge $e \in E(G)$ returns one of the two subgraphs in $G \setminus \{e\}$ which contains the target. The two search models are basically equivalent for paths, but in general, a query to a vertex with a high degree reveals more information than a query to an incident edge. For example, to locate the target in a star, one query is sufficient in the vertex model, while in the edge model all edges need to be queried in the worst case. The binary search can be further generalized to general graphs, where the answer to a vertex (or edge) query returns directional information with respect to the *shortest* path to the target [15].

It should be noted that other optimization criteria can be considered (e.g., the average search cost), as well as noisy search models where the response to a query can be incorrect with a certain probability.

For paths, a natural dynamic programming approach obtains an optimal algorithm for the weighted version of the problem, which runs in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space [19]. The cubic time can be improved by exploiting a monotonic structure of search costs of the sub-intervals, which yields an $\mathcal{O}(n^2)$ time solution [5], for any assignment of the query costs. A linear time $(2 + \epsilon + o(1))$-approximation algorithm has been presented in [21], where the approach is based on a trade-off between querying the vertices with small costs and maintaining a balanced decision tree.

For unweighted trees, optimal search strategies can be computed in linear time by calculating vertex rankings of the input tree [23, 26], which is a type of graph coloring, where on a path between any two vertices with the same color there must be a vertex with a color of a higher value [17, 18]. Once the ranking is calculated, the search strategy is obtained by always querying the vertex with the highest color, considering the set of vertices that can still contain the target. Tree rankings have been studied as an independent line of research leading to the discovery of several algorithms, including linear time algorithms [22, 28].

For weighted trees, the binary search problem becomes NP-complete [12]. A quasi polynomial-time approximation scheme was obtained through a dynamic programming approach [8]. The QPTAS is then recursively applied to carefully selected subtrees of the input tree, which results in a polynomial-time algorithm with an approximation ratio of $\mathcal{O}(\sqrt{\log n})$.

The binary search problem has been extended to general graphs in [13, 15], where a function Φ is defined over the vertex set of the graph that can still contain the search target, given the responses to the queries performed so far. The Φ is defined in such a way that for each considered vertex it encodes the sum of distances to all other considered vertices. In each step, the vertex that minimizes Φ is selected as the next query, and the search space is reduced to the vertices consistent with the query response, i.e., the vertices with a shortest path from the queried vertex containing the edge returned by the query. The algorithm uses at most $\log_2 n$ queries to find arbitrary vertex in a graph with a uniform cost of queries. A further study of using graph median for queries can be found in [11].

In the context of practical applications, a related search model has been used to characterize the confidentiality of searching information in databases of genetics information [16].

1.3 Our contribution

An open question has been posed several times about whether a constant-factor approximation algorithm exists for arbitrary weight functions, see e.g. [2, 4, 5]. The main motivation of this work is to address this question by finding a natural input instance for which a constant-factor approximation is achievable. We define this instance as follows: we allow an input tree to have arbitrary structure but the weight function is assumed to be monotonic:

▶ **Definition 1.** *Given a tree $T = (V, E, \omega)$, we say that a cost function ω is* monotonic *if there exists a vertex $r \in V$ such that for any $u, v \in V$, if v lies on the path between r and u in T, then $\omega(u) \leq \omega(v)$.*

Our primary contribution is in providing a constant-factor approximation algorithm for the binary search problem in this subclass of trees with non-uniform weights. Our solution shows how a method developed for the problem with uniform costs [26] can be leveraged for the monotonic case. Our main result is the following.

▶ **Theorem 2.** *There exists an 8-approximate adaptive search algorithm for the search problem in weighted trees with monotonic cost functions. The algorithm runs in linear time.*

The organization of the remaining parts of the paper is as follows. Section 1.4 describes the motivation for our work, based on practical applications of the studied search model and several use cases of the monotonic structure of the cost function. Section 2 introduces our notation and essential concepts commonly used in the related body of research. Section 3 presents the class of decision trees (structured decision trees) that we restrict ourselves to when building the solution for the search problem. Section 4 introduces further concepts, which are needed to formulate our algorithm in the subsequent Section 5 and to prove the approximation factor in Section 6. Section 7 concludes the paper, where we suggest potential directions for future research.

1.4 Motivation and applications

Graphs form a natural abstraction for processes in many domains. Large graphs are becoming common in data management and processing systems [27]. Compared to data structures without order over its elements, it is known that maintaining at least a partial order improves the performance of fundamental operations like search, update or insert in terms of the number of comparisons needed [20]. At the same time, creating optimal strategies for searching in structures with a partial order, can be inherently hard [12]. We conclude that efficient approximation algorithms for searching in large graphs are important for the advancement of systems focused on large graph analysis.

The classic binary search and its generalizations is a basic problem in computer science that occurs in numerous practical problems. The binary search problem posed as a graph ranking problem can be used to model parallel Cholesky factorization of matrices [9], scheduling of parallel database queries [10], and VLSI layouts [29]. The binary search model for graphs has been used to build a general framework for interactive learning of classifiers, rankings, or clusterings [14].

A monotonic structure of the comparisons cost occurs naturally when considering data access times in computer systems, e.g., due to memory hierarchies of modern processors, characteristics of the storage devices, or distributed nature of data management systems. In the literature on the binary search problem, a Hierarchical Memory Model of computation has been studied [1], in which the memory access time is monotonic with respect to the location of a data element in the array. Another work was devoted to modeling the problem of text retrieval from magnetic or optical disks, where a cost model was such that the cost of a query was monotonic from the location of the previous query performed in the search process [24]. We also mention an application of tree domains in automated bug search in computer code [3]. In such a case naturally occurs a possibility that the tree to be searched has the monotonicity property. In particular, each query represents performing an automatic test that determines whether the part of the code that corresponds to the subtree under the tested vertex has an error. Thus, the vertices that are closer to the root represent larger parts of the code and thus may require more or longer lasting tests.

2 Preliminaries

The set of vertices of a tree T is denoted by $V(T)$. Given a tree $T=(V,E,w)$ and a connected subset of vertices $V' \subseteq V(T)$, we denote by $T[V']$ the induced subtree of T consisting of all vertices from V'. For a rooted T, we denote the root of T by $r(T)$, while for any $v \in V(T)$, T_v denotes the subtree of T rooted at v and consisting of v and all its descendants. For any two intervals $I, I' \in \{[a,b) \colon 0 \leq a < b\}$ we write $I > I'$ when $i > i'$ for each $i \in I$ and $i' \in I'$.

The *sequence of queries* is defined as a sequence of vertices queried by a search strategy $\mathcal{A}$ for a target t and is denoted by $\mathcal{Q}_{\mathcal{A}}(T,t)$. The vertex queried by $\mathcal{A}$ in the i-th step is denoted by $\mathcal{Q}_{\mathcal{A},i}(T,t)$. Given a search strategy for T, one can easily generate a search strategy for any subtree T' by simply discarding the queries to the vertices outside of T'. A useful way of encoding search strategies is by using decision trees.

▶ **Definition 3.** *We define a* decision tree *for $T=(V,E,w)$ generated by a search strategy $\mathcal{A}$ as a rooted tree $\mathcal{T}=(\mathcal{V},\mathcal{E})$, where $\mathcal{V}=V(T)$. The root of $\mathcal{T}$ is the first vertex queried by $\mathcal{A}$. For any $v\in\mathcal{V}$, v has a child v' if and only if v' is queried by $\mathcal{A}$ right after v for some choice of the target. Each $v\in\mathcal{V}$ corresponds to a subset $\mathcal{T}(v)\subset V(T)$ which contains all vertices that can still contain the target when $\mathcal{A}$ queries v.*

It can be seen that the root r of any decision tree corresponds to $\mathcal{T}(r)=V(T)$ and the leaves correspond to single vertices in $V(T)$.

Lemma 4 shows a property of the decision trees with respect to the positions of vertices that belong to some connected component of the input tree.

▶ **Lemma 4.** *Let $\mathcal{T}$ be a decision tree for T. If T' is a subtree of T, then there exists a vertex $u\in V(T')$, such that for every $v\in V(T')$ it holds $v\in V(\mathcal{T}_u)$.*

Proof. We traverse the decision tree $\mathcal{T}$ starting at the root of $\mathcal{T}$ and following a path to some leaf. If $r(\mathcal{T})\in V(T')$, then the lemma follows. Otherwise, consider children v_i of $r(\mathcal{T})$. Because in T there is only one edge incident to $r(\mathcal{T})$ that lies on a path to all vertices of T', all vertices of T' belong to exactly one subtree $\mathcal{T}_{v_i}$. We continue the traversal in $\mathcal{T}_{v_i}$ and set u as the first visited vertex in this process that belongs to T'. ◀

We now formally define the cost of a search process, which is the property the algorithm is trying to minimize. In terms of the search strategy $\mathcal{A}$ for an input tree $T=(V,E,\omega)$, the sum of costs of all queries performed when finding a target $t\in V(T)$ is denoted by $COST(\mathcal{A},t)$, while the *cost* of $\mathcal{A}$ is defined as $COST(\mathcal{A})=\max_{t\in V} COST(\mathcal{A},t)$.

Sometimes in our analysis, we will take one decision tree and measure its cost with different query times, i.e., for different weights. This approach will allow us to artificially increase the cost of some queries in the decision tree, which we formalize as follows. Let $\mathcal{T}$ be a decision tree for $T=(V,E,\omega)$ and ω' an arbitrary cost function defined on $V(T)$, potentially a different one than ω. We denote by $COST(\mathcal{T},\omega')$ the *cost of the decision tree according to the cost function* ω':

$$COST(\mathcal{T},\omega')=\max\{\sum_{v\in P}\omega'(v)\mid P \text{ is a path from the root to a leaf in } \mathcal{T}\}.$$

The *cost* of $\mathcal{T}$ is then $COST(\mathcal{T},\omega)$ and we shorten it to $COST(\mathcal{T})$. (Note that one may equivalently define the cost of $\mathcal{T}$ by taking $COST(\mathcal{T})=COST(\mathcal{A})$ where $\mathcal{T}$ is generated by $\mathcal{A}$.) The minimum cost that is achievable for T is denoted by

$$OPT(T)=\min\{COST(\mathcal{T})\mid \mathcal{T} \text{ is a decision tree for } T\}.$$

We say that $\mathcal{T}$ *is optimal* if and only if $COST(\mathcal{T})=OPT(T)$.

We will use the following simple folklore lemma whose proof is given for completeness. Lemma 5 has been previously formulated in [5] in the context of a slightly different problem of edge search in weighted trees.

▶ **Lemma 5.** *If $\mathcal{T}$ is a decision tree for T, then for any subtree T' of T there exists a decision tree $\mathcal{T}'$, such that $COST(\mathcal{T}')\leq COST(\mathcal{T})$.*

Proof. We first apply Lemma 4 and reduce $\mathcal{T}$ to its subtree rooted at $u \in V(T')$. Then, we sequentially remove from $\mathcal{T}_u$ all nodes that do not belong to T'. For every removed node v, consider all subtrees rooted at the children of v in the current decision tree. When removing v we also remove all subtrees that do not contain any vertex from T' but we connect the roots of all other subtrees (note that there will be only one such subtree) directly under the parent of v. Hence, the reduced tree is still a decision tree. What is more, all paths from the root to a leaf in the obtained decision tree result from corresponding paths from $\mathcal{T}$ by removing zero or more nodes, which upperbounds the cost of $\mathcal{T}'$ by the cost of $\mathcal{T}$. ◀

We say that a cost function ω is *rounded* if the values taken by ω are powers of two (for any $v \in V(T)$, $\omega(v) = 2^k$ for some $k \in \mathbb{N}$). We also say that T' is the *rounding* of T if it is obtained from T by rounding the cost function to the closest, greater power of two. We obtain that $OPT(T') \leq 2\,OPT(T)$ because $\omega' \leq 2\omega$. On the other hand, Lemma 6 shows how large is the overhead from applying an optimal decision tree for the input with a rounded cost function to search through the input tree with the original weights.

▶ **Lemma 6.** *Let $T' = (V, E, \omega')$ be the rounding of $T = (V, E, \omega)$. Let $\mathcal{T}$ and $\mathcal{T}'$ be optimal decision trees respectively for T and T'. We have:*

$$COST(\mathcal{T}', \omega) \leq 2\,COST(\mathcal{T}) = 2OPT(T).$$

Proof. We have $COST(\mathcal{T}', \omega) \leq COST(\mathcal{T}', \omega') \leq COST(\mathcal{T}, \omega') \leq 2COST(\mathcal{T}, \omega)$. The first inequality holds because $\omega \leq \omega'$. The second inequality holds because $\mathcal{T}'$ is optimal for T'. The last inequality holds because $\omega' \leq 2\omega$. ◀

3 Structured decision trees

For a rooted tree $T = (V, E, \omega)$ with a monotonic ω, we assume that the root is selected as in Definition 1, that is informally speaking, while traversing any path from the root to a leaf, the weights of visited vertices are non-increasing. From now on we assume that the input tree is rooted. We define a *layer* of a tree T as a subgraph L of T such that all vertices in $V(L)$ have the same cost, $\omega(u) = \omega(v)$ for any $u, v \in V(L)$. A connected component of L is called a *layer component*. We also denote by $\omega(L)$ the cost of querying a vertex that belongs to L, i.e., $\omega(L) = \omega(v)$ for any $v \in V(L)$.

The *upper border* of layer L is the set of roots of its layer components. The subset of $V(L)$ consisting of vertices that have at least one child that belongs to a different layer is called the *lower border* of L. We will also say that a layer component L' is *directly below* a layer component L if and only if $r(L')$ has a parent in L. The *top* layer component is the one that contains the root of T and a *bottom* layer component is any L such that there is no component directly below L. We note that we will apply the above terms to a rounded ω.

▶ **Definition 7.** *Let T be a tree with a monotonic cost function and $\mathcal{T}$ a decision tree for T. Consider a layer component L of T. Let $v = r(L)$. We say that $\mathcal{T}$ is* structured *with respect to L if for every vertex $u \in V(T_v)$, it holds that $u \in V(\mathcal{T}_v)$. We say that a decision tree $\mathcal{T}$ for T is* structured *if and only if $\mathcal{T}$ is structured with respect to all layer components of T.*

Informally speaking, in a structured $\mathcal{T}$ we require that each vertex in the entire subtree T_v is below v in $\mathcal{T}$. This is one of the central ideas in our work for the following reason. While performing bottom-up processing of (an unweighted) T in [26] (and also other works e.g. [22]) each vertex v encodes which steps are already used for queries of the vertices from T_v. In our case, that is in a weighted T, we need to encode intervals and the technical problem with that is that they have different durations while moving upwards from one layer component to the next. Thus, some intervals that are free to perform queries while moving to a parent of v may be too short due to the weights in the next layer component. To deal with that, we make $v = r(L)$ also the root of $\mathcal{T}_v$. In this way only one interval, i.e., the one assigned to v needs to be taken into account while moving upwards.

Our method requires the decision tree to be structured only when processing the internal components of T. By not enforcing structuring with respect to the top layer component, a potential additive cost of a single query can be avoided. See the *Process Component* procedure (Line 9, Algorithm 2). However, this optimization does not change the worst-case cost of the strategy. In result, the decision tree generated by our algorithm is not structured in general, and although $r(\mathcal{T})$, the first vertex queried by the strategy, always belongs to the top layer component of T, we may have that $r(\mathcal{T}) \neq r(T)$, while in a structured decision tree we always have $r(\mathcal{T}) = r(T)$, because $\mathcal{T}$ is structured with respect to the top layer component of T.

It turns out in our analysis that considering only structured decision trees introduces an overall multiplicative cost of 2 to the performance of the algorithm:

▶ **Lemma 8.** *Let $T = (V, E, \omega)$ be a tree with a monotonic and rounded cost function. There exists a structured decision tree $\mathcal{T}'$ for T, such that $COST(\mathcal{T}') \leq 2\,OPT(T)$.*

Proof. Let $\mathcal{T}$ be any decision tree for T. Since $\mathcal{T}$ is selected arbitrarily, it is enough to prove that $COST(\mathcal{T}') \leq 2\,COST(\mathcal{T})$.

In order to construct the structured decision tree, we traverse $\mathcal{T}$ in a breadth-first fashion starting at the root of $\mathcal{T}$. The $\mathcal{T}'$ is constructed in the process.

By Lemma 4, for an arbitrary layer component L of T, there exists a vertex $u' \in V(L)$ such that all vertices from L belong to $\mathcal{T}_{u'}$. Among all vertices of L, u' is visited first during the traversal.

Consider an arbitrary node u being accessed during the traversal. Let L be the layer component such that $u \in L$. If $\mathcal{T}$ is structured with respect to L we continue with the next node. Otherwise (which for a specific L will happen only once, when $u = u'$, as argued in the last paragraph of the proof), we modify $\mathcal{T}$ by performing the following steps.

Let $v = r(L)$. We want to replace the subtree $\mathcal{T}_u$ with a subtree consisting of the same nodes but rooted at v. Consider a subtree T' of the input T, which can still contain the target when the search process is about to query u according to $\mathcal{T}$. E.g., $V(T') = V(\mathcal{T}_u)$. Let v has k neighbors in T'. We attach under v the decision trees created with Lemma 5 for the k components obtained by removing v from T'. Lemma 5 shows that the cost of any of the k decision trees is not greater than $COST(\mathcal{T}_u)$. Accounting for the cost of v, structuring L increases $COST(\mathcal{T})$ no more than $\omega(L)$. Consider an arbitrary path P from root to a leaf in $\mathcal{T}$. For every node $u \in P$ that triggers structuring (informally, u is u' for some L) an additional node from L is inserted in P. Thus, in the worst case, the weighted length of arbitrary P increases 2 times when transforming $\mathcal{T}$ into structured $\mathcal{T}'$.

By starting the breadth-first traversal at the root of $\mathcal{T}$, structuring with regards to one layer component does not make $\mathcal{T}$ not structured with regards to any of the previously structured layer components. ◀

By combining Lemmas 6 and 8 one obtains Corollary 9, which relates the cost of an optimal decision tree for an input T with the cost of a structured decision tree obtained through the analysis of the rounding of T.

▶ **Corollary 9.** *For any tree T with a monotonic cost function there exists a structured decision tree with rounded weights, whose cost is at most $4 \cdot OPT(T)$.*

4 Bottom-up tree processing

Our method extends the ranking-based method for searching in trees with uniform costs of queries [26, 28]. The state-of-the-art algorithm calculates a function $f\colon V(T) \to \mathbb{N}$ called a *strategy function* (also called in the literature a *ranking* [22, 28], or tree-depth [25]). The strategy function is such that for each pair of distinct vertices $v_1, v_2 \in V(T)$, if $f(v_1) = f(v_2)$, then each path connecting v_1 and v_2 has a vertex v_3 such that $f(v_3) > f(v_1)$. One property of the strategy function is that it encodes the (reversed) order of the queries. E.g., in any sequence of queries, a vertex with a higher value of f is always queried before those with lower values of the strategy function. Thus, the maximal value of f over the vertices of a tree is the worst-case number of queries necessary to search the tree. In order to express the variable costs of queries we extend the strategy function into an *extended strategy function* (see also [7]).

▶ **Definition 10.** *A function $f\colon V(T) \to \{[a,b)\colon 0 \leq a < b\}$, where $|[a,b)| \geq \omega(v)$ for each $v \in V(T)$, is an* extended strategy function *if for each pair of distinct vertices $v_1, v_2 \in V(T)$, if $f(v_1) \cap f(v_2) \neq \emptyset$, then the path connecting v_1 and v_2 has a v_3 such that $f(v_3) > f(v_1) \cup f(v_2)$. The length of an interval assigned to a vertex v encodes the cost corresponding to querying v.*

Keeping in mind that the cost function is the time needed to perform a query, the values of an extended strategy function indicate the time periods in which the respective vertices will be queried. Note that, e.g., due to the rounding, the time periods will be longer than the query durations and this is easily resolved either by introducing idle times or by following the actual query durations while performing a search based on a strategy produced by our algorithm.

We say that $u \in V(T)$ is *visible from* $v \in V(T)$ if on the path from v to u there is no vertex x such that $f(x) > f(u)$, where f is a strategy function defined on $V(T)$. If u is visible from v we also say that the value $f(u)$ is visible from v. The sequence of values visible from v in descending order is called a *visibility sequence* for v.

In our approach, informally speaking, we will partition the rounding of the input tree into several subtrees (the layer components) and for each of them, we will use the algorithm from [26]. To clearly describe the initialization required prior to triggering the leveraged subroutine, and also for completeness, we recall a *vertex extension operator* used in [26] for calculating a strategy function. Let f be a strategy function defined on all vertices below $v \in V(T)$. Let v_i, $1 \leq i \leq k$, be the children of v. The vertex extension operator attributes v with $f(v)$, based on the visibility sequences $S(v_i)$ of the children according to the following procedure. Consider a set M with values that belong to at least two distinct visibility sequences $S(v_i)$. Let m be the maximum value in M if $M \neq \emptyset$ or -1 otherwise. The $f(v)$ is set to the lowest integer greater than m that does not belong to any $S(v_i)$. The following lemma characterizes the vertex extension operator.

▶ **Lemma 11** ([26]). *Given the visibility sequences assigned to the children of $v \in V(T)$, the* vertex extension operator *assigns to v a visibility sequence that is lexicographically minimal (the vertex extension operator is minimizing). Moreover, (lexicographically) increasing the visibility sequence of a child of v does not decrease the visibility sequence calculated for v (the vertex extension operator is monotone).*

In our approach, we restrict ourselves to extended strategy functions which generate decision trees that are structured (see Definition 7).

▶ **Definition 12.** *We say that an extended strategy function f, defined on the vertices of a tree T with monotonic cost function, is* structured *if for any layer component L, $f(u) > f(v)$ for each $v \in V(T_u)$, where $u = r(L)$.*

It follows that a structured extended strategy function represents a structured decision tree.

As mentioned earlier, we will process each layer component L with the vertex extension operator but in order to do it correctly, we need to initialize appropriately the roots of the layer components below L. For that, we define the following operators. Let f be an extended strategy function defined on the vertices of T_v, where v is the root of a layer component L'. The *structuring operator* assigns to v the minimal interval so that f is a structured extended strategy function on T_v. (We note that in our algorithm v will have some interval assigned when the structuring operator is applied to v. Hence, the application will assign the new required interval.) The *cost scaling operator* aligns the interval attributed to v: if $f(v) = [a, b)$ and L is the layer component directly above L', the cost scaling operator assigns v a new interval $[\omega(L)\lceil\frac{b}{\omega(L)}\rceil - \omega(L), \omega(L)\lceil\frac{b}{\omega(L)}\rceil)$. We will say that the interval $f(v)$ is *aligned to* $\omega(L)$ after this modification. Informally speaking, this is done so that the children of the leaves in L have intervals whose endpoints are multiples of $\omega(L)$ so that they become "compatible" with the allowed placements for intervals of the vertices from L. Yet in other words, this allows us to translate the intervals of the children of the leaves in L into integers that can be treated during bottom-up processing of L in a uniform way.

▶ **Observation 13.** *Let a layer component L' be directly below a layer component L in $T = (V, E, \omega)$. If $r(L')$ is assigned an interval $[a, b)$, whose endpoints are consecutive multiples of $\omega(L')$ (i.e. $a = k\omega(L')$ and $b = (k+1)\omega(L')$ for some integer k), then the cost scaling operator assigns to $r(L')$ an interval $[a', b')$ such that $b' \leq b + \omega(L) - \omega(L')$.*

Proof. Consider a layer component L of $T = (V, E, \omega)$ and a structured extended strategy function f defined on the vertices below L. The cost scaling operator selects b' as the smallest multiple of $\omega(L)$ that is greater or equal than b. Since $a' = b' - \omega(L)$, we have that $a' < b$. On the other hand, because $b - a = \omega(L')$, a is the highest multiple of $\omega(L')$ that is less than b. Subsequently, because $w(L)$ is divisible by $w(L')$, we obtain that $a' \leq a$. What follows, $a' + \omega(L') \leq b$, and finally $b' \leq b + \omega(L) - \omega(L')$. ◀

5 Algorithm

We propose an algorithm *Structured Tree Search* (Algorithm 1). The algorithm starts with a pre-processing pass, which transforms the input tree to its rounding and selects a root arbitrarily from the set of vertices with the highest cost o query (cf. Definition 1). An extended strategy function f is then calculated during a bottom-up traversal over the layer components of the tree by applying the *Process Component* procedure (Algorithm 2). Hence, the extended strategy function for the vertices of a layer component L is calculated only after it is already defined for all vertices below the component. Moreover, since we use a structured f, only its values assigned to the roots of the layer components below L are important when calculating f for the vertices of L.

Given a layer component L, the procedure *Process Component* iterates over the vertices of L in a depth-first, postorder fashion. Suppose that a structured extended strategy function f is defined on all vertices below L and that f is aligned to $\omega(L)$ at all roots of layer components directly below L. To calculate f for the vertices of L, we first calculate the strategy function f' for each vertex of L and then obtain f from f' using the formula: $f(u) = [f'(u)\omega(L), (f'(u) + 1)\omega(L))$, where $u \in V(L)$. (Recall that this is a conversion

Algorithm 1 Structured Tree Search.

```
Let $T$ be an input tree with a monotonic cost function, for which the extended strategy
function $f$ is calculated.
$T \leftarrow$ the rounding of $T$
for each layer component $L$ in bottom-up fashion do
    ProcessComponent($L, f$)
end for
return $f$
```

that takes us from the integer-valued strategy function f' to the interval-valued extended strategy function.) To provide an applicable input for the vertex extension operator at the leaves of L, for each child $u \in V(L')$ of such leaf, we calculate the integer $f'(u) = \frac{b}{\omega(L')}$ that is derived from the extended strategy function at u, i.e. from $f(u) = [a, b)$. Note that since our decision trees are structured, the visibility sequence corresponding to each root of a layer component below L (which includes u) consists of only one value, the one that equals the strategy function at the root. What follows, deriving the value of f' only for the roots of the layer components directly below L is sufficient to create a valid input for the vertex extension operator. In other words, we make these preparations to use the operator and the corresponding method from [26]. Once each vertex $v \in V(L)$ obtains the corresponding extended strategy function $f(v)$, we apply for the root of L the structuring operator followed by the cost scaling operator. This will close the entire “cycle” of processing one layer component.

Algorithm 2 Calculates extended strategy function f for vertices of a layer component L.

```
procedure ProcessComponent($L$, $f$)
    for each root $v$ of a layer component directly below $L$ do
        $f'(v) \leftarrow \frac{b}{\omega(L)}$, where $b$ is the right endpoint of the interval $f(v)$
    end for
    for each $u \in V(L)$ in a postorder fashion do
        Obtain $f'(u)$ by applying the vertex extension operator to $v$
        $f(u) \leftarrow$ the interval derived from $f'(u)$, see Section 4
    end for
    if the root $v$ of $L$ is not the root of $T$ then
        Update $f(v)$ by applying the structuring operator to $v$
        Update $f(v)$ by applying the cost scaling operator to $v$
    end if
end procedure
```

6 Analysis

Lemma 14 characterizes the interval of the extended strategy function that the *Process Component* procedure assigns to the root of a layer component.

▶ **Lemma 14.** *Consider a layer component L of $T = (V, E, \omega)$ and a structured extended strategy function f defined on the vertices below L. Let the intervals of f assigned to the roots of the layer components directly below L be aligned to $\omega(L)$. Suppose that the values of f over $V(L)$ are obtained by a call to procedure* ProcessComponent(L,f). *We have that f is a structured extended strategy function on $V(T_{r(L)})$, and the lowest possible interval I, such that $I > f(u)$, for each u below $r(L)$, is assigned to $r(L)$.*

Proof. Due to the alignment of f at the roots of the components directly below L, the strategy function f' derived from f for these roots is consistent with the query costs in L. That is, the construction of f' done at the beginning of the procedure gives an integer-valued strategy function. Since f is structured, among all vertices below L, the extension operator correctly needs to consider only the values of f' at the roots of components directly below L.

By Lemma 11, the visibility sequence calculated for the root of L is (lexicographically) minimized among all possible valid assignments of f'. Hence in particular, the value of $\max\{f'(v) \mid v \in V(T_{r(L)})\}$ is minimized. It follows that if there exists such an optimal assignment of f' that is structured, then this f' is calculated by the procedure is structured and consequently the f obtained from f' is minimal and structured. If there exists no optimal assignments of f' over $V(L)$ that is structured, the structuring operator modifies f and assigns to $r(L)$ the lowest interval that is greater than $f(u)$ for any u below $r(L)$. ◀

What follows from Lemma 14 and Observation 13 is that given an extended strategy function f fixed below a layer component L, Algorithm 1 extends f to the vertices of L in such a way that the interval assigned to $r(L)$ is not higher than the optimal interval (assuming that f is fixed below L) incremented by an additive factor $\omega(L') - \omega(L)$, where L' is a layer component directly above L. If L is the top component, the structuring transform is not applied and the additional cost is not incurred.

We now formulate a technical Lemma 15 that helps analyze the strategy generated by Algorithm 1.

▶ **Lemma 15.** *Let $T = (V, E, \omega)$ be a tree with a monotonic and rounded cost function. We denote by f_{opt} an optimal structured extended strategy function on $V(T)$. Let L be a layer component of T with k layer components L_j, $1 \leq j \leq k$, directly below L. Let $v_j = r(L_j), 1 \leq j \leq k$. We define f' as a function defined on $V(T)$ such that for all vertices below any v_j, f' is equal to f_{opt}, while for the vertices v_j and above, f' is equal to $f_{opt} + \omega(L)$. It holds that f' is a structured extended strategy function on $V(T)$.*

Proof. We first show that f' is an extended strategy function. We need to show that for any pair $x, y \in V(T)$, $x \neq y$, if $f'(x) = f'(y)$, then there is a vertex $z \in V(T)$ separating x, y such that $f'(z) > f'(x)$. (See Definition 10.)

We analyze the following cases with respect to the locations of the vertices x and y. In all cases we assume that $x \neq y$, $f'(x) = f'(y)$, and z is the vertex separating x and y according to f'.

- $x \in V(L_i)$, $y \in V(L_j)$ and $i = j$.
  Let $v = r(L_i)$. Because f_{opt} is structured, f' is also structured within the subtree T_v, since it assigns v a higher interval than f_{opt} and f' is otherwise identical to f_{opt}. What follows, because f' is structured and we assumed $f'(x) = f'(y)$, we have that $x \neq v$ and $y \neq v$.
  Consider the f_{opt} as defined on all vertices of L_i. Because $x, y \in V(L_i)$, we have $z \in V(L_i)$. If $z \neq v$, then $f'(z) = f_{opt}(z) > f_{opt}(x) = f'(x)$, where the last equation holds because $x \neq v$. If $z = v$, we also have $f'(z) > f'(x)$, because f' is structured in T_v.
- $x \in V(L_i)$, $y \in V(L_j)$ and $i \neq j$.
  Because f' is structured in L_i, we have $z = r(L_i)$.
- Both x and y belong to $V(L)$.
  Because f_{opt} is an extended strategy function in L, there is a $z \in V(L)$ such that $f_{opt}(z) > f_{opt}(x)$. Then, by adding $\omega(L)$ to both sides of the inequality one obtains $f'(z) > f'(x)$.

- $x \in V(L_i)$ for some i and $y \in V(L)$.
  If $x \neq r(L_i)$ then $z = r(L_i)$, because f' is structured in L_i.
  Otherwise, both x and y are in the subtree $T[V(L) \cup \{x\}]$. Recall that in this subtree f' is derived from f_{opt} by adding a constant offset, which gives an extended strategy function.

To show that f' is structured in the remaining cases, we observe that for all vertices above L, f' is derived from the structured f_{opt} by shifting the assigned intervals by a positive offset, which does not change the structural property of the layer components' roots. ◀

▶ **Lemma 16.** *The cost of the decision tree generated by Algorithm 1 is at most 2 times greater than the cost of an optimal structured decision tree.*

Proof. The proof is by induction, bottom-up over the layer components of the input tree. Take a tree $T = (V, E, \omega)$ with a monotonic and rounded cost function. We denote by $c(i), 1 \leq i \leq l$, the query cost to a vertex in the $i - th$ layer, where the layers are ordered according to their cost of query and l is the number of layers in T. Let f_{opt} be an optimal (structured) extended strategy function and f_{alg} be the extended strategy function generated by Algorithm 1. We denote by $z(v)$ the cost of querying the parent of v or the cost of v itself if there is no vertex above v, i.e. when v is the root of T.

We want to prove the following induction claim. For any layer component L it holds $f_{alg}(v) \leq f_{opt}(v) + z(v)$, where $v = r(L)$.

The cost of the decision tree generated by f_{alg} is equal to the supremum of $f_{alg}(r(T))$. Since the cost of a single query to the top layer is not more than the cost of an optimal structured decision tree, the lemma follows from the induction claim applied to the top layer component.

Let L be a bottom layer component. Based on Lemma 14 we know that by applying the vertex extension and structuring operators, Algorithm 1 generates an optimal structured assignment of the extended strategy function to the vertices of L. In a case where L is also the top component (which is the trivial case of T being a single layer component) we have f_{alg} is optimal and the induction claim holds. Otherwise, the cost scaling operator extends the interval assigned to $r(L)$, increasing the cost of L by (at most) an additive factor of $c(2) - c(1)$ over the optimal cost. Since the added factor is less than $z(r(L))$, the induction claim holds for the bottom layer component L.

Let L be a layer component from the i-th layer, with k layer components L_j, $1 \leq j \leq k$, directly below L. We denote the roots of the layer components L_j by v_j. Consider a strategy function f' which is defined as in Lemma 15, that is for all vertices below any v_j, f' is equal to f_{opt}, while for vertices v_j and all vertices above, f' is equal to $f_{opt} + \omega(L)$. Lemma 15 implies that f' is a structured extended strategy function on $V(T)$.

From the induction hypothesis we have that for any v_j, $f_{alg}(v_j) \leq f_{opt}(v_j) + z(v_j) = f_{opt}(v_j) + \omega(L)$. According to Lemma 14 and Observation 13, Algorithm 1 assigns intervals to the vertices of L such that the interval assigned to the root of L is the lowest possible (when L is the top component), or the lowest possible interval incremented by a positive offset of $\omega(L') - \omega(L)$, where L' is the layer component directly above L. If we sum up the additive increments due to cost scaling in all layer components below L, we obtain $\sum_{i<l} c(i) < c(l) = z(v)$ (recall that $c(i)$'s are powers of 2). Thus, $f_{alg}(v) \leq f_{opt}(v) + z(v)$. ◀

By combining Lemmas 6, 8, and 16 we obtain that Algorithm 1 generates a solution for the binary search problem with a cost at most 8 times greater than that of an optimal solution. We note that Algorithm 1 maintains the main structure of the linear time algorithm from [26] (a single bottom-up pass over vertices of the tree.) Our method extends the state-of-the-art algorithm by adding a fixed number of $\mathcal{O}(1)$ steps, computed when visiting vertices during the bottom-up traversal. See the *Process Component* procedure, Algorithm 2. What follows, Algorithm 1 also runs in linear time. This proves Theorem 2.

7 Conclusions

We recall a related tree search problem, called *edge search*, in which one performs queries on edges: each reply provides information which endpoint of the queried edge is closer to the target [3]. One can similarly define a monotonic cost function $\omega\colon E(T) \to \mathbb{R}_+$ for the edge search by requiring that there exists a choice for the root r so that for any two edges $\{x, y\}$ and $\{y, z\}$, if x is closer to r than y, then $\omega(\{x, y\}) \geq \omega(\{y, z\})$.

Interestingly, the edge search problem is NP-complete for monotonic weight function which follows from [6], although the authors do not obtain this fact directly. More precisely, the reduction in [6] constructs a spider in which each leg has three edges with weights $2a, a$, and $\Theta(an)$ (listing them by following from the root to the leaf). Denote by e the latter edge incident to a leaf. It is not hard to see that e can be replaced by a star on $\Theta(n)$ edges, each of weight at most a, thus getting an instance with a monotonic cost function and the same search cost.

It turns out that the problem we study in this work is more general in the class of weighted trees – see the corresponding reduction in [8]. This reduction however does not preserve the property of monotonicity. In other words, a conversion from an instance of the edge search with a monotonic cost function produces an instance of the vertex search with a non-monotonic cost function. Thus, the fact that the edge search is NP-complete for weighted trees pointed above does not imply hardness in our case. We do believe that the argument of hardness cannot be easily obtained in a similar way as for the edge search in [6] and we leave as an open question whether the vertex search problem studied in this work is NP-complete in case of monotonic cost functions.

Also, we repeat the previously mentioned interesting and challenging open question whether a constant-factor approximation is possible for the vertex search in trees with general weight functions.

References

1 Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. In Alfred V. Aho, editor, *STOC 1987*, pages 305–314. ACM, 1987. doi:10.1145/28395.28428.

2 Haris Angelidakis. Shortest path queries, graph partitioning and covering problems in worst and beyond worst case settings. *CoRR*, abs/1807.09389, 2018. arXiv:1807.09389.

3 Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999. doi:10.1137/S009753979731858X.

4 Piotr Borowiecki, Dariusz Dereniowski, and Dorota Osula. The complexity of bicriteria tree-depth. In Evripidis Bampis and Aris Pagourtzis, editors, *FCT 2021*, volume 12867 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2021. doi:10.1007/978-3-030-86593-1_7.

5 Ferdinando Cicalese, Tobias Jacobs, Eduardo Sany Laber, and Caio Dias Valentim. Binary identification problems for weighted trees. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 255–266. Springer, 2011. doi:10.1007/978-3-642-22300-6_22.

6 Ferdinando Cicalese, Balázs Keszegh, Bernard Lidický, Dömötör Pálvölgyi, and Tomás Valla. On the tree search problem with non-uniform costs. *Theor. Comput. Sci.*, 647:22–32, 2016. doi:10.1016/j.tcs.2016.07.019.

7 Dariusz Dereniowski. Edge ranking of weighted trees. *Discret. Appl. Math.*, 154(8):1198–1209, 2006. doi:10.1016/j.dam.2005.11.005.

8 Dariusz Dereniowski, Adrian Kosowski, Przemyslaw Uznanski, and Mengchuan Zou. Approximation strategies for generalized binary search in weighted trees. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 84:1–84:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.84.

9 Dariusz Dereniowski and Marek Kubale. Cholesky factorization of matrices in parallel and ranking of graphs. In Roman Wyrzykowski, Jack J. Dongarra, Marcin Paprzycki, and Jerzy Wasniewski, editors, *PPAM 2003*, volume 3019 of *Lecture Notes in Computer Science*, pages 985–992. Springer, 2003. doi:10.1007/978-3-540-24669-5_127.

10 Dariusz Dereniowski and Marek Kubale. Efficient parallel query processing by graph ranking. *Fundam. Informaticae*, 69(3):273–285, 2006.

11 Dariusz Dereniowski, Aleksander Lukasiewicz, and Przemyslaw Uznanski. An efficient noisy binary search in graphs via median approximation. In Paola Flocchini and Lucia Moura, editors, *IWOCA 2021*, volume 12757 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2021. doi:10.1007/978-3-030-79987-8_19.

12 Dariusz Dereniowski and Adam Nadolski. Vertex rankings of chordal graphs and weighted trees. *Inf. Process. Lett.*, 98(3):96–100, 2006. doi:10.1016/j.ipl.2005.12.006.

13 Dariusz Dereniowski, Stefan Tiegel, Przemyslaw Uznanski, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *SOSA 2019*, volume 69 of *OASIcs*, pages 4:1–4:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.4.

14 Ehsan Emamjomeh-Zadeh, David Kempe, Mohammad Mahdian, and Robert E. Schapire. Interactive learning of a dynamic structure. In Aryeh Kontorovich and Gergely Neu, editors, *ALT 2020*, volume 117 of *Proceedings of Machine Learning Research*, pages 277–296. PMLR, 2020.

15 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In Daniel Wichs and Yishay Mansour, editors, *STOC 2016*, pages 519–532. ACM, 2016. doi:10.1145/2897518.2897656.

16 Mine Su Erturk and Kuang Xu. Private genetic geneaology search. Research papers, Stanford University, Graduate School of Business, 2021. URL: https://EconPapers.repec.org/RePEc:ecl:stabus:3973.

17 Ananth V. Iyer, H. Donald Ratliff, and Gopalakrishnan Vijayan. Optimal node ranking of trees. *Inf. Process. Lett.*, 28(5):225–229, 1988. doi:10.1016/0020-0190(88)90194-9.

18 Ananth V. Iyer, H. Donald Ratliff, and Gopalakrishnan Vijayan. On an edge ranking problem of trees and graphs. *Discret. Appl. Math.*, 30(1):43–52, 1991. doi:10.1016/0166-218X(91)90012-L.

19 William J. Knight. Search in an ordered array having variable probe cost. *SIAM J. Comput.*, 17(6):1203–1214, 1988. doi:10.1137/0217076.

20 Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

21 Eduardo Sany Laber, Ruy Luiz Milidiú, and Artur Alves Pessoa. On binary searching with non-uniform costs. In S. Rao Kosaraju, editor, *SODA 2001*, pages 855–864. ACM/SIAM, 2001.

22 Tak Wah Lam and Fung Ling Yue. Optimal edge ranking of trees in linear time. In Howard J. Karloff, editor, *SODA 1998*, pages 436–445. ACM/SIAM, 1998.

23 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In Shang-Hua Teng, editor, *SODA 2008*, pages 1096–1105. SIAM, 2008.

24 Gonzalo Navarro, Ricardo A. Baeza-Yates, Eduardo F. Barbosa, Nivio Ziviani, and Walter Cunto. Binary searching with nonuniform costs and its application to text retrieval. *Algorithmica*, 27(2):145–169, 2000. doi:10.1007/s004530010010.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

26 Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *FOCS 2006*, pages 379–388. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.32.

27 Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. The future is big graphs: a community view on graph processing systems. *Commun. ACM*, 64(9):62–71, 2021. doi:10.1145/3434642.

28 Alejandro A. Schäffer. Optimal node ranking of trees in linear time. *Inf. Process. Lett.*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

29 Arunabha Sen, Haiyong Deng, and Sumanta Guha. On a graph partition problem with application to VLSI layout. *Inf. Process. Lett.*, 43(2):87–94, 1992. doi:10.1016/0020-0190(92)90017-P.

On the Identity Problem for Unitriangular Matrices of Dimension Four

Ruiwen Dong ✉
Department of Computer Science, University of Oxford, UK

Abstract

We show that the Identity Problem is decidable in polynomial time for finitely generated sub-semigroups of the group $\mathsf{UT}(4, \mathbb{Z})$ of 4×4 unitriangular integer matrices. As a byproduct of our proof, we also show the polynomial-time decidability of several subset reachability problems in $\mathsf{UT}(4, \mathbb{Z})$.

2012 ACM Subject Classification Computing methodologies → Symbolic and algebraic manipulation

Keywords and phrases identity problem, matrix semigroups, unitriangular matrices

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.43

Related Version *Full Version*: https://arxiv.org/abs/2202.05225

1 Introduction

Among the most prominent algorithmic problems for matrix semigroups are the *Identity Problem* and the *Membership Problem*. For the Membership Problem, the input is a finite set of square matrices $A_1, \ldots, A_k$ and a target matrix A. The problem is to decide whether A lies in the semigroup generated by $A_1, \ldots, A_k$. The Identity Problem is the Membership Problem restricted to the case where A is the identity matrix. These two problems are closely related to each other, and, as shown in many circumstances, studying the Identity Problem is usually the first step in studying the Membership Problem.

For general matrices, the Membership Problem is undecidable by a classical result of Markov [10]. Indeed, it is one of the earliest undecidability results on algorithmic problems in matrix semigroups. Most variants of the problem remain undecidable in low dimension. For example, the *Mortality Problem*, which is the Membership Problem in which the target matrix is 0, is undecidable in dimension three [12]. In dimension four, the Membership Problem is undecidable for matrices in $\mathrm{SL}(4, \mathbb{Z})$ (see [11]), while the Identity Problem is undecidable for the set of 4×4 integer matrices $\mathcal{M}_{4\times4}(\mathbb{Z})$ (see [2]).

However, there has also been steady progress on the decidability side. The Membership Problem is shown to be decidable for $\mathrm{GL}(2, \mathbb{Z})$ in [4]. This decidability result is then extended to 2×2 integer matrices with nonzero determinant [13], and to 2×2 integer matrices with determinants equal to 0 and ±1 [14]. It remains an intricate open problem whether the Membership Problem or the Identity Problem is decidable for $\mathrm{SL}(3, \mathbb{Z})$.

Recently, there has been more progress on closing the decidability gap by restricting consideration to the class of unitriangular matrices. It has long been known that the *Group Membership Problem* is decidable for $\mathsf{UT}(n, \mathbb{Z})$, the group of unitriangular integer matrices of dimension n. The Group Membership Problem asks to decide whether a matrix A lies in the *group* generated by given matrices $A_1, \ldots, A_k$. In fact, it is decidable for all finitely generated solvable matrix groups [8]. Later, Babai et al. [1] showed that the Group Membership Problem for *commuting matrices* can be computed in polynomial time (note that commuting matrices are simultaneously upper-triangularizable). However, there are significant differences between the group case and the semigroup case. In fact, for large enough n, the *Knapsack Problem* for $\mathsf{UT}(n, \mathbb{Z})$ is undecidable [7]. Given matrices $A_1, \ldots, A_k$

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 43; pp. 43:1–43:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and A, the Knapsack Problem asks to decide whether there exist natural numbers $e_1, \dots, e_k$ such that $A_1^{e_1} \cdots A_k^{e_k} = A$. From the undecidability of the Knapsack Problem, one can deduce the undecidability of the semigroup Membership Problem for $\mathsf{UT}(n, \mathbb{Z})$ for large enough n [9].

Nevertheless, there have been some positive decidability results. The Identity Problem has been shown to be decidable for the group of 3×3 unitriangular integer matrices $\mathsf{UT}(3, \mathbb{Z})$ and the Heisenberg groups H_{2n+1} in [6]. Shortly after, the decidability result was extended to the Membership Problem [5]. Ko et al. left open the problem whether the Identity Problem in $\mathsf{UT}(n, \mathbb{Z})$ is decidable for $n \geq 4$, as well as finding the smallest n for which the Membership Problem for $\mathsf{UT}(n, \mathbb{Z})$ becomes undecidable.

The main result of this paper is that the Identity Problem is decidable in polynomial time for $\mathsf{UT}(4, \mathbb{Z})$. This further narrows the gap between decidability and undecidability and can be regarded as a first step towards the Membership Problem for $\mathsf{UT}(4, \mathbb{Z})$. The foundation of our method is the arguments developed in [5] for the Membership Problem of $\mathsf{UT}(3, \mathbb{Z})$. However, in order to pass from dimension three to four, we need to introduce additional methods from convex geometry, linear programming and even use the aid of computational algebraic geometry software. The proof for $\mathsf{UT}(3, \mathbb{Z})$ heavily relies on the fact that the subgroup generated by commutators of matrices from a given subset of $\{A_1, \dots, A_k\} \subset \mathsf{UT}(3, \mathbb{Z})$ is isomorphic to a subgroup of $\mathbb{Z}$. This is no longer the case for $\mathsf{UT}(4, \mathbb{Z})$. However, $\mathsf{UT}(4, \mathbb{Z})$ is still metabelian [15], and its derived subgroup is isomorphic to $\mathbb{Z}^3$. Given a finite set $\mathcal{G} \subseteq \mathsf{UT}(4, \mathbb{Z})$, we construct elements in $\langle \mathcal{G} \rangle$ that fall inside the derived subgroup of $\mathsf{UT}(4, \mathbb{Z})$. These elements then generate a cone in $\mathbb{Z}^3$ under the isomorphism between the derived subgroup and $\mathbb{Z}^3$. The possible shapes of this cone will determine the Identity Problem.

There is strong evidence that the new techniques introduced in this paper can help tackle the Identity Problem for $\mathsf{UT}(n, \mathbb{Z})$ with $n \geq 5$.

2 Preliminaries

Denote by $\mathsf{UT}(4, \mathbb{Z})$ the group of upper triangular integer matrices with ones on the diagonal:

$$\mathsf{UT}(4, \mathbb{Z}) \coloneqq \left\{ \begin{pmatrix} 1 & a & d & f \\ 0 & 1 & b & e \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \middle| a, b, c, d, e, f \in \mathbb{Z} \right\}.$$

Denote its normal subgroups

$$\mathsf{U}_1 \coloneqq \left\{ \begin{pmatrix} 1 & 0 & d & f \\ 0 & 1 & 0 & e \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \middle| d, e, f \in \mathbb{Z} \right\}, \quad \mathsf{U}_2 \coloneqq \left\{ \begin{pmatrix} 1 & 0 & 0 & f \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \middle| f \in \mathbb{Z} \right\}$$

in the lower central series: $\mathsf{UT}(4, \mathbb{Z}) \trianglerighteq \mathsf{U}_1 = [\mathsf{UT}(4, \mathbb{Z}), \mathsf{UT}(4, \mathbb{Z})] \trianglerighteq \mathsf{U}_2 = [\mathsf{UT}(4, \mathbb{Z}), \mathsf{U}_1]$ (see [15, Chapter 5]). In particular, U_1 and U_2 are respectively the derived subgroup and the centre of $\mathsf{UT}(4, \mathbb{Z})$. For convenience, we introduce the following notations:

$$UT(a, b, c; d, e, f) \coloneqq \begin{pmatrix} 1 & a & d & f \\ 0 & 1 & b & e \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad U_1(d, e, f) \coloneqq UT(0, 0, 0; d, e, f).$$

There are surjective group homomorphisms $\varphi_0 \colon \mathsf{UT}(4, \mathbb{Z}) \to \mathbb{Z}^3$ defined by

$$\varphi_0(UT(a, b, c; d, e, f)) = (a, b, c),$$

with $\ker(\varphi_0) = \mathsf{U}_1$, and $\varphi_1 \colon \mathsf{U}_1 \to \mathbb{Z}^2$,

$$\varphi_1(U_1(d,e,f)) = (d,e),$$

with $\ker(\varphi_1) = \mathsf{U}_2$. Moreover, U_1 is itself abelian, with a natural isomorphism $\tau \colon \mathsf{U}_1 \xrightarrow{\sim} \mathbb{Z}^3$:

$$\tau(U_1(d,e,f)) = (d,e,f).$$

Denote by τ_d the projection $U_1(d,e,f) \mapsto d$, τ_e the projection $U_1(d,e,f) \mapsto e$, and τ_f the projection $U_1(d,e,f) \mapsto f$. Then, $\varphi_1 = (\tau_d, \tau_e)$ and $\tau = (\tau_d, \tau_e, \tau_f)$.

Finally, define the subgroup of $\mathsf{UT}(4,\mathbb{Z})$:

$$\mathsf{U}_{10} \coloneqq \{U_1(0,e,f) \mid e,f \in \mathbb{Z}\} \trianglelefteq \mathsf{U}_1.$$

For a finite set of matrices $\mathcal{G} = \{A_1, \dots, A_k\}$, denote by $\langle \mathcal{G} \rangle$ the semigroup generated by $\mathcal{G}$. In this paper, we are concerned with the following problems.

▶ **Definition 1.** Let G be a monoid of matrices, and H a subset of G.

(i) The *Identity Problem* in G asks, given a finite set of matrices $\mathcal{G}$ in G, whether $I \in \langle \mathcal{G} \rangle$. If this is the case, we say that the identity matrix is *reachable.*

(ii) The *H-Reachability Problem* in G asks, given a finite set of matrices $\mathcal{G}$ in G, whether $H \cap \langle \mathcal{G} \rangle \neq \emptyset$. If this is the case, we say that H is *reachable.*

The main result of this paper is that the Identity Problem in $\mathsf{UT}(4,\mathbb{Z})$ is decidable in polynomial time, with respect to the number of bits required to encode all the entries of the matrices in $\mathcal{G}$ (each matrix $UT(a,b,c,d,e,f)$ is encoded by the entries a,b,c,d,e,f).

It turns out that the three problems: Identity Problem, U_2-Reachability and U_{10}-Reachability are interconnected and it is more convenient to devise algorithms that decide them simultaneously. A trivial observation is that, because $I \in \mathsf{U}_2 \subset \mathsf{U}_{10}$, a positive instance of the Identity Problem is also a positive instance of U_2-Reachability; and a positive instance of U_2-Reachability is also a positive instance of U_{10}-Reachability.

The following definitions will be used throughout this paper.

▶ **Definition 2** (String, product and Parikh vector)**.** Let $\mathcal{G} = \{A_1, \dots, A_k\}$ be a fixed set of matrices in $\mathsf{UT}(4,\mathbb{Z})$. A *string* of $\mathcal{G}$ is an expression $B_1 B_2 \cdots B_m$ such that $B_i \in \mathcal{G}, i = 1, \dots, m$. The *product* of a string $B_1 B_2 \cdots B_m$ is the matrix $P \in \mathsf{UT}(4,\mathbb{Z})$ such that $P = B_1 B_2 \cdots B_m$. The *Parikh vector* of a string $B_1 B_2 \cdots B_m$ is the vector $\boldsymbol{\ell} = (\ell_1, \dots, \ell_k) \in \mathbb{Z}_{\geq 0}$ where

$$\ell_j = \operatorname{card}(\{i \mid B_i = A_j\}), j = 1, \dots, k.$$

When $\mathcal{G}$ is clear from the context, we simply use the term “string” instead of “string of $\mathcal{G}$”.

For an integer $n \geq 1$, the *Heisenberg group* of dimension $2n+1$ is the group H_{2n+1} of $(n+2) \times (n+2)$ integer matrices of the form $H = \begin{pmatrix} 1 & \boldsymbol{a} & c \\ 0 & I_n & \boldsymbol{b}^\top \\ 0 & 0 & 1 \end{pmatrix}$, where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}^n$, $c \in \mathbb{Z}$.

The following result comes from [6] and [5].

▶ **Lemma 3** ([5, Theorem 7])**.** *The Identity Problem and the Membership Problem in* H_{2n+1} *are decidable for all* $n \geq 1$.

3 Identity problem, U_2- and U_{10}-Reachability in $\mathsf{UT}(4,\mathbb{Z})$

In this section, we construct algorithms that decide the Identity Problem, U_2-Reachability and U_{10}-Reachability in $\mathsf{UT}(4,\mathbb{Z})$.

3.1 Overview of decision strategy

For any set of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l \in \mathbb{R}^n$, denote by

$$\langle \boldsymbol{v}_1, \ldots, \boldsymbol{v}_l \rangle_{\mathbb{R}_{\geq 0}} \coloneqq \left\{ \sum_{i=1}^{l} r_i \boldsymbol{v}_i \,\middle|\, r_i \in \mathbb{R}_{\geq 0}, i = 1, \ldots, l \right\}$$

the $\mathbb{R}_{\geq 0}$-cone generated by $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l$, and by $\langle \boldsymbol{v}_1, \ldots, \boldsymbol{v}_l \rangle_{\mathbb{R}}$ the $\mathbb{R}$-vector space spanned by $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l$.

Let $\mathcal{G} = \{A_1, \ldots, A_k\}$ be a set of matrices in $\mathsf{UT}(4, \mathbb{Z})$, for which we want to decide the Identity Problem, U_2-Reachability and U_{10}-Reachability. Define the $\mathbb{R}_{\geq 0}$-cone

$$\mathcal{C} \coloneqq \langle \varphi_0(A_1), \ldots, \varphi_0(A_k) \rangle_{\mathbb{R}_{\geq 0}}, \tag{1}$$

and denote by $\mathcal{C}^{lin}$ its lineality space, i.e. the largest linear subspace (by inclusion) contained in $\mathcal{C}$. In particular, $\mathcal{C}^{lin} = \mathcal{C} \cap -\mathcal{C}$. A basis of $\mathcal{C}^{lin}$ can be effectively computed in polynomial time [16]. For any matrix $A_i \in \mathcal{G}$, the projection $\varphi_0(A_i)$ can be either in $\mathcal{C}^{lin}$ or in $\mathcal{C} \setminus \mathcal{C}^{lin}$. However, in order to reach U_1, which contains the identity matrix, U_2 and U_{10}, one can only use matrices A_i with $\varphi_0(A_i) \in \mathcal{C}^{lin}$. This is formally stated by the following proposition.

▶ **Proposition 4.** *If the product of a string $B_1 \cdots B_m$ is in U_1, then every $B_j, j = 1, \ldots m$, must be in the set $\{A_i \in \mathcal{G} \mid \varphi_0(A_i) \in \mathcal{C}^{lin}\}$.*

Proof. Suppose on the contrary that some B_j satisfies $\varphi_0(B_j) \in \mathcal{C} \setminus \mathcal{C}^{lin}$.

Since φ_0 is a group homomorphism, we have

$$B_1 \cdots B_m \in \mathsf{U}_1 \iff \varphi_0(B_1 \cdots B_m) = \mathbf{0} \iff \sum_{i=1}^{m} \varphi_0(B_i) = \mathbf{0}.$$

Therefore, $-\varphi_0(B_j) = \sum_{i \neq j} \varphi_0(B_i) \in \mathcal{C}$.

Hence, the linear subspace $\varphi_0(B_j)\mathbb{R} = \langle \varphi_0(B_j), -\varphi_0(B_j) \rangle_{\mathbb{R}_{\geq 0}}$ is contained in $\mathcal{C}$. This yields $\varphi_0(B_j)\mathbb{R} \subseteq \mathcal{C}^{lin}$, a contradiction to $\varphi_0(B_j) \in \mathcal{C} \setminus \mathcal{C}^{lin}$. ◀

The overall strategy for constructing our algorithm is to use induction on $\text{card}(\mathcal{G})$. If $\text{card}(\mathcal{G}) = 0$, then the answers to the Identity Problem, U_2-Reachability and U_{10}-Reachability are all negative. Suppose now that we have an algorithm that decides all three problems for every set of at most $k - 1$ matrices, we will construct an algorithm that decides them for a set of k matrices $\mathcal{G} = \{A_1, \ldots, A_k\}$. By Proposition 4, if some matrix A_i satisfies $\varphi_0(A_i) \in \mathcal{C} \setminus \mathcal{C}^{lin}$, then we can discard it without changing the answer to the Identity Problem or $\mathsf{U}_2, \mathsf{U}_{10}$-Reachability. This decreases the number of elements in $\mathcal{G}$, and an algorithm is available by the induction hypothesis on $\text{card}(\mathcal{G})$. Hence, we can suppose that every $A_i \in \mathcal{G}$ satisfies $\varphi_0(A_i) \in \mathcal{C}^{lin}$, so $\mathcal{C} = \mathcal{C}^{lin}$ is a linear space.

Since $\varphi_0(A_i) \in \mathbb{Z}^3$, $\mathcal{C}$ is a linear subspace of $\mathbb{R}^3$. We identify cases according to the dimension of $\mathcal{C}$, with each of the following four subsections treating the case of dimension 3, 1, 0, 2. The pseudocode of the decision procedure for the Identity Problem is given here as a reference point for the detailed case analysis. The decision procedures for U_2-reachability and U_{10}-reachability follow similar patterns and their pseudocode is given in the appendix of the full version of this paper. Note that the decision procedure for the Identity Problem invokes the decision procedure for U_2-reachability as a subroutine. Similarly, the decision procedure for U_2-reachability will invoke the decision procedure for U_{10}-reachability as a subroutine.

Algorithm 1 IdentityProblem(): deciding the Identity Problem for a subset of $\mathsf{UT}(4,\mathbb{Z})$.

```
Input: A set $\mathcal{G} = \{A_1, \dots, A_k\}$ of matrices in $\mathsf{UT}(4,\mathbb{Z})$.
Output: True or False.

Step 1: Compute the cone $\mathcal{C}$ and its lineality space $\mathcal{C}^{lin}$. For $i = 1, \dots, k$, if some
        $\varphi_0(A_i)$ is not in $\mathcal{C}^{lin}$, return IdentityProblem($\mathcal{G} \setminus \{A_i\}$).
Step 2: a. If $\dim(\mathcal{C}) = 3$, return True.
        b. If $\dim(\mathcal{C}) = 1$, return True if the condition in Proposition 15(i) is satisfied,
           otherwise return False.
        c. If $\dim(\mathcal{C}) = 0$, return True if $\tau(A_i), i = 1, \dots, m$ generate a semigroup
           containing $\mathbf{0}$, otherwise return False.
        d. If $\dim(\mathcal{C}) = 2$, compute a non-zero vector $(p, q, r) \in \mathbb{Q}^3$ orthogonal to $\mathcal{C}$.
             i. If $p = 0$, but $q, r$ are not zero, or $r = 0$, but $q, p$ are not zero.
                Compute $L_0$, if $\mathrm{supp}(L_0) = \{1, \dots, k\}$, return True, otherwise return
                IdentityProblem($\{A_i \mid i \in \mathrm{supp}(L_0)\}$).
            ii. If $p = r = 0$, problem reduces to Identity Problem in $\mathrm{H}_5$.
           iii. If $p = q = 0, r \neq 0$ or $r = q = 0, p \neq 0$, compute $A'_i$ as in (9). Return
                U2Reachability($A'_1, \dots, A'_k$) (see full version of paper).
```

We now give an overview of the motivation behind classifying cases according to the dimension of $\mathcal{C}$. As a convention, we always use $A_i, i = 1, \dots, k$ to denote elements of the fixed generating set $\mathcal{G}$, and Greek letters to denote their entries, i.e. $A_i = UT(\alpha_i, \beta_i, \kappa_i; \delta_i, \epsilon_i, \phi_i)$. We use $B_i, i = 1, \dots, m$ to denote arbitrary elements in $\langle \mathcal{G} \rangle$ (when appearing in *strings*, they are elements in $\mathcal{G}$), and Latin letters to denote their entries, i.e. $B_i = UT(a_i, b_i, c_i, d_i, e_i, f_i)$. The variables B_i can depend on the context.

First of all, we need some results on the structure of products in $\mathsf{UT}(4,\mathbb{Z})$. For a positive integer m, denote by S_m the permutation group of the set $\{1, \dots, m\}$. Throughout this paper, given some matrices $B_1, \dots, B_m \in \langle \mathcal{G} \rangle$, we will often be computing the product of strings of the form $B^t_{\sigma(1)} \cdots B^t_{\sigma(m)}$, where $\sigma \in \mathrm{S}_m$ and $t \in \mathbb{Z}_{\geq 0}$. The overall idea is to find various strings $B^t_{\sigma(1)} \cdots B^t_{\sigma(m)}$ whose product is in $\mathsf{U}_1 \overset{\tau}{\cong} \mathbb{Z}^3$, then use them to generate an abelian semigroup containing the identity matrix. Let us define the following important values and abbreviations that will be used throughout this paper. These complicated formulas are related to the *logarithm* of the matrices B_i, and readers can for the time being ignore their exact form and treat them as black boxes.

▶ **Notation 5.** Given a series of matrices $B_1, \dots, B_m$ where $B_i = UT(a_i, b_i, c_i; d_i, e_i, f_i), i = 1, \dots, m$, we introduce the following notation:

(i) For $\sigma \in \mathrm{S}_m, t \in \mathbb{Z}_{\geq 0}$,

$$B(\sigma, t) \coloneqq B^t_{\sigma(1)} \cdots B^t_{\sigma(m)}. \tag{2}$$

(ii) For $\sigma \in \mathrm{S}_m$,

$$D_\sigma \coloneqq \sum_{i<j} a_{\sigma(i)} b_{\sigma(j)} + \frac{1}{2} \sum_{i=1}^{m} a_i b_i, \quad E_\sigma \coloneqq \sum_{i<j} b_{\sigma(i)} c_{\sigma(j)} + \frac{1}{2} \sum_{i=1}^{m} b_i c_i,$$

$$F_\sigma \coloneqq \sum_{i<j<k} a_{\sigma(i)} b_{\sigma(j)} c_{\sigma(k)} + \frac{1}{2} \sum_{i<j} (a_{\sigma(i)} b_{\sigma(i)} c_{\sigma(j)} + a_{\sigma(i)} b_{\sigma(j)} c_{\sigma(j)}) + \frac{1}{6} \sum_{i=1}^{m} a_i b_i c_i,$$

$$G_\sigma \coloneqq \sum_{i<j}(a_{\sigma(i)}e_{\sigma(j)} + d_{\sigma(i)}c_{\sigma(j)} - \frac{1}{2}a_{\sigma(i)}b_{\sigma(j)}c_{\sigma(j)} - \frac{1}{2}a_{\sigma(i)}b_{\sigma(i)}c_{\sigma(j)}) + \frac{1}{2}\sum_{i=1}^{m}(a_ie_i + d_ic_i - a_ib_ic_i). \quad (3)$$

(iii) For $i = 1, \dots, m$,

$$D_i \coloneqq d_i - \frac{1}{2}a_ib_i, \quad E_i \coloneqq e_i - \frac{1}{2}b_ic_i, \quad F_i \coloneqq f_i - \frac{1}{2}(a_ie_i + d_ic_i) + \frac{1}{3}a_ib_ic_i. \quad (4)$$

The following proposition gives an exact expression for $B(\sigma, t)$. Because of the heavily computational nature of most of our propositions, their proofs are given in the appendix of the full version of this paper.

▶ **Proposition 6.** *Let $B_i = UT(a_i, b_i, c_i; d_i, e_i, f_i), i = 1, \dots, m$, $\sigma \in \mathrm{S}_m, t \in \mathbb{Z}_{\geq 0}$, then*

$$B(\sigma, t) = UT\left(t\sum_{i=1}^{m} a_i, t\sum_{i=1}^{m} b_i, t\sum_{i=1}^{m} c_i; \right. \left. t^2D_\sigma + t\sum_{i=1}^{m} D_i, t^2E_\sigma + t\sum_{i=1}^{m} E_i, t^3F_\sigma + t^2G_\sigma + t\sum_{i=1}^{m} F_i\right). \quad (5)$$

Notice that $B(\sigma, t) \in \mathsf{U}_1$ if and only if $\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} b_i = \sum_{i=1}^{m} c_i = 0$, a condition that does not depend on the value of t.

Proposition 6 shows that, if $B(\sigma, t)$ is in U_1, then as $t \to \infty$, the asymptotic behaviour of $\tau(B(\sigma, t))$ approaches the vector $(t^2D_\sigma, t^2E_\sigma, t^3F_\sigma)$, provided that $D_\sigma, E_\sigma, F_\sigma$ do not vanish. Therefore, the hope is that, as t, σ vary, the vectors $(t^2D_\sigma, t^2E_\sigma, t^3F_\sigma)$ can generate $\mathbb{R}^3$ as an $\mathbb{R}_{\geq 0}$-cone, barring a few degenerate cases. If these degenerate cases do not happen, then the different vectors $\tau(B(\sigma, t))$ will also generate $\mathbb{R}^3$ as an $\mathbb{R}_{\geq 0}$-cone. In particular, the identity element in $\mathbb{R}^3$ can be generated by $\tau(B(\sigma, t))$ as an additive semigroup, giving a positive answer to the Identity Problem. For the degenerate cases, they will be treated individually. As it will turn out, there are only two types of degeneracy (which may occur simultaneously):

(i) $F_\sigma = 0$ for all σ.

(ii) For some $p, r \in \mathbb{Q}$, possibly zero, we have $pD_\sigma = rE_\sigma$ for all σ.

When (i) occurs, the asymptotic behaviour of $\tau(B(\sigma, t))$ approaches the vector $(t^2D_\sigma, t^2E_\sigma, t^2G_\sigma)$, since G_σ is the second most dominant term after F_σ. This situation reminds us of the Identity Problem for H_3, and can be solved in a similar way. When (ii) occurs, the vectors $(t^2D_\sigma, t^2E_\sigma, t^3F_\sigma)$ are constrained to a strict linear subspace of $\mathbb{R}^3$. Hence, in order to describe the $\mathbb{R}_{\geq 0}$-cone generated by the vectors $\tau(B(\sigma, t))$, one needs to consider the sub-dominant terms as well, i.e. the terms $t\sum_{i=1}^{m} D_i, t\sum_{i=1}^{m} E_i$.

The rest of this paper aims to formalize this idea. We first exhibit a series of lemmas that characterise these degenerate cases. Our first lemma shows that, supposing $B(\sigma, t) \in \mathsf{U}_1$, then degenerate case (ii) happens if and only if $\langle \varphi_0(B_1), \dots, \varphi_0(B_m)\rangle_{\mathbb{R}}$ is degenerate (i.e. of dimension at most 2).

▶ **Lemma 7.** *Given $p, r \in \mathbb{R}$ and $m \geq 2$. Suppose $\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} b_i = \sum_{i=1}^{m} c_i = 0$. The two following statements are equivalent:*

(i) *For all $\sigma \in \mathrm{S}_m$, $pD_\sigma = rE_\sigma$.*

(ii) *Either $b_i = 0$ for all $i = 1, \dots, m$, or there exist $q \in \mathbb{R}$, such that $pa_i + qb_i + rc_i = 0$ for all $i = 1, \dots, m$.*

The next lemma shows that if $B(\sigma,t) \in \mathsf{U}_1$, then by "inverting" σ, we get a permutation σ' such that (D_σ, E_σ) and $(D_{\sigma'}, E_{\sigma'})$ are opposites of one another.

▶ **Lemma 8.** *Suppose $\sum_{i=1}^m a_i = \sum_{i=1}^m b_i = \sum_{i=1}^m c_i = 0$, $m \geq 2$. For every $\sigma \in \mathrm{S}_m$, there exists $\sigma' \in \mathrm{S}_m$, such that $(D_{\sigma'}, E_{\sigma'}) = -(D_\sigma, E_\sigma)$.*

We then show that, if $B(\sigma,t) \in \mathsf{U}_1$, then the value of F_σ for different $\sigma \in \mathrm{S}_m$ sums up to zero:

▶ **Lemma 9.** *Suppose $\sum_{i=1}^m a_i = \sum_{i=1}^m b_i = \sum_{i=1}^m c_i = 0$, where $m \geq 3$. Then we have $\sum_{\sigma \in \mathrm{S}_m} F_\sigma = 0$.*

The last lemma characterizes situations where the aforementioned degenerate case (i) happens. Its proof relies on the aid of a computational algebraic geometry software due to the complexity of the expressions F_σ.

▶ **Lemma 10.** *Let $m = 4$. Suppose $\sum_{i=1}^4 a_i = \sum_{i=1}^4 b_i = \sum_{i=1}^4 c_i = 0$. Then, $F_\sigma = 0$ for all $\sigma \in \mathrm{S}_4$, if and only if at least one of the following four conditions holds:*

(i) $a_1 = a_2 = a_3 = a_4 = 0$.

(ii) $b_1 = b_2 = b_3 = b_4 = 0$.

(iii) $c_1 = c_2 = c_3 = c_4 = 0$.

(iv) $\operatorname{rank} \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \end{pmatrix} \leq 1$.

A common idea of Lemma 7 and Lemma 10 is that the degeneracy of $(D_\sigma, E_\sigma, F_\sigma)$ is related to the degeneracy of $\varphi_0(B_1), \ldots, \varphi_0(B_m)$. Hence, it is natural to consider the degeneracy of the vectors $\varphi_0(A_i), i = 1, \ldots, k$, where $A_i \in \mathcal{G}$ are the elements of the generating set. This degeneracy is described by the dimension of the linear space $\mathcal{C}$ discussed at the beginning of the section. This justifies the classification according to $\dim(\mathcal{C})$. We now begin the case analysis.

3.2 $\mathcal{C}$ has dimension 3

The main idea of this case is that, for a well chosen set of matrices $B_1, B_2, B_3, B_4 \in \langle \mathcal{G} \rangle$, the vectors $(D_\sigma, E_\sigma, F_\sigma), \sigma \in \mathrm{S}_4$, are not degenerate and the asymptotic behaviour of $\tau(B(\sigma,t))$ approaches the vector $(t^2 D_\sigma, t^2 E_\sigma, t^3 F_\sigma)$, leading to a positive answer to the Identity Problem.

Let $B_1, B_2, B_3, B_4 \in \langle \mathcal{G} \rangle$ with $B_i = UT(a_i, b_i, c_i; d_i, e_i, f_i), i = 1, \ldots, 4$ be such that

$$\sum_{i=1}^{4} \varphi_0(B_i) = 0 \tag{6}$$

and

$$\langle \varphi_0(B_1), \varphi_0(B_2), \varphi_0(B_3), \varphi_0(B_4) \rangle_{\mathbb{R}_{\geq 0}} = \mathcal{C} = \mathbb{R}^3. \tag{7}$$

Equation (6) shows that $B(\sigma,t) \in \mathsf{U}_1$ for all $\sigma \in \mathrm{S}_4, t \in \mathbb{Z}_{\geq 0}$.

The following lemma shows that the d,e-coordinates of different $\tau(B(\sigma,t))$ generate $\mathbb{R}^2$ as an $\mathbb{R}_{\geq 0}$-cone.

▶ **Lemma 11.** *Assuming (6) and (7), we have $\langle \{\varphi_1(B(\sigma,t)) \mid \sigma \in \mathrm{S}_4, t \in \mathbb{Z}\} \rangle_{\mathbb{R}_{\geq 0}} = \mathbb{R}^2$.*

Proof. First, we claim that $\langle\{(D_\sigma, E_\sigma) \mid \sigma \in \mathrm{S}_4\}\rangle_{\mathbb{R}} = \mathbb{R}^2$.

In fact, suppose to the contrary that $\langle\{(D_\sigma, E_\sigma) \mid \sigma \in \mathrm{S}_4\}\rangle_{\mathbb{R}}$ has dimension at most 1. Then there exist $p, r \in \mathbb{R}$, not both zero, such that for all $\sigma \in \mathrm{S}_4$, $pD_\sigma = rE_\sigma$. By Lemma 7, this means that either $b_i = 0$ for all i or there exists some $q \in \mathbb{R}$ such that $pa_i + qb_i + rc_i = 0$ for all i. In both cases, the $\mathbb{R}$-linear subspace spanned by $\varphi_0(B_1), \varphi_0(B_2), \varphi_0(B_3), \varphi_0(B_4)$ has dimension at most 2, contradicting Equation (7). This proves the claim. Hence, there exist $\sigma_1, \sigma_2 \in \mathrm{S}_4$ such that $(D_{\sigma_1}, E_{\sigma_1})$ and $(D_{\sigma_2}, E_{\sigma_2})$ span $\mathbb{R}^2$ as an $\mathbb{R}$-linear space.

Next, by Lemma 8, there exist $\sigma_1', \sigma_2' \in \mathrm{S}_4$ such that $(D_{\sigma_1'}, E_{\sigma_1'}) = -(D_{\sigma_1}, E_{\sigma_1})$ and $(D_{\sigma_2'}, E_{\sigma_2'}) = -(D_{\sigma_2}, E_{\sigma_2})$. It follows that $(D_{\sigma_1}, E_{\sigma_1}), (D_{\sigma_2}, E_{\sigma_2}), (D_{\sigma_1'}, E_{\sigma_1'}), (D_{\sigma_2'}, E_{\sigma_2'})$ generate $\mathbb{R}^2$ as an $\mathbb{R}_{\geq 0}$-cone, and all four vectors are non-zero.

Finally, consider the products $B(\sigma, t)$ with $\sigma \in \{\sigma_1, \sigma_2, \sigma_1', \sigma_2'\}$. By Proposition 6, when $t \to +\infty$, we have $\varphi_1(B(\sigma, t)) = (D_\sigma, E_\sigma)t^2 + O(t)$. Therefore, when t is large enough, the angle between $\varphi_1(B(\sigma, t))$ and (D_σ, E_σ) tends to zero, for all $\sigma \in \{\sigma_1, \sigma_2, \sigma_1', \sigma_2'\}$. Hence, for large enough t, $\varphi_1(B(\sigma_1, t)), \varphi_1(B(\sigma_2, t)), \varphi_1(B(\sigma_1', t)), \varphi_1(B(\sigma_2', t))$ generate $\mathbb{R}^2$ as an $\mathbb{R}_{\geq 0}$-cone. This proves the Lemma. ◀

The next proposition shows that as σ, t vary, the vectors $\tau(B(\sigma, t))$ generate $\mathbb{R}^3$ as an $\mathbb{R}_{\geq 0}$-cone.

▶ **Proposition 12.** *Assuming* (6) *and* (7)*, we have* $\langle\{\tau(B(\sigma, t)) \mid \sigma \in \mathrm{S}_4, t \in \mathbb{Z}\}\rangle_{\mathbb{R}_{\geq 0}} = \mathbb{R}^3$.

Proof. First, note that all $B(\sigma, t)$ have integer coefficients. By Lemma 11, there exist elements $P_1, P_2, P_3 \in \langle\{B(\sigma, t) \mid \sigma \in \mathrm{S}_4, t \in \mathbb{Z}\}\rangle$ such that $\varphi_1(P_i), i = 1, 2, 3$ generate $\mathbb{R}^2$ as an $\mathbb{R}_{\geq 0}$-cone (see Figure 1 for an illustration.).

Next, the idea is to find two additional matrices $P_+, P_- \in \{B(\sigma, t) \mid \sigma \in \mathrm{S}_4, t \in \mathbb{Z}\}$, whose images under τ are relatively close to the f-axis in $\mathbb{R}^3$. By Lemmas 9 and 10, there exist $\sigma_+, \sigma_- \in \mathrm{S}_4$ such that $F_{\sigma_+} > 0, F_{\sigma_-} < 0$. Indeed, by condition (7), none of the four conditions of Lemma 10 hold. Thus there exists $\sigma \in \mathrm{S}_4$ such that $F_\sigma \neq 0$. Then Lemma 9 shows we can find σ_+ and σ_- such that $F_{\sigma_+} > 0$ and $F_{\sigma_-} < 0$.

By Proposition 6, when $t \to +\infty$, we have $\tau_f(B(\sigma_+, t)) = F_{\sigma_+}t^3 + O(t^2)$ and $\tau_f(B(\sigma_-, t)) = F_{\sigma_-}t^3 + O(t^2)$, whereas $\tau_d(B(\sigma_\pm, t)) = O(t^2)$ and $\tau_e(B(\sigma_\pm, t)) = O(t^2)$. Therefore, when t is large enough, the angle between $\tau(B(\sigma_+, t))$ and $(0, 0, 1)$ tends to zero, as well as the angle between $\tau(B(\sigma_-, t))$ and $(0, 0, -1)$.

Finally, we claim that there exists t such that $\tau(P_1), \tau(P_2), \tau(P_3), \tau(B(\sigma_+, t)), \tau(B(\sigma_-, t))$ generate $\mathbb{R}^3$ as an $\mathbb{R}_{\geq 0}$-cone. See Figure 1 for an illustration. To justify this claim, suppose to the contrary that for every t, the $\mathbb{R}_{\geq 0}$-cone spanned by the five vectors $\tau(P_1), \tau(P_2), \tau(P_3)$, $\tau(B(\sigma_+, t))$, $\tau(B(\sigma_-, t))$ is a proper subset of $\mathbb{R}^3$. In other words, if we denote by $\langle \cdot, \cdot \rangle$ the canonical inner product of $\mathbb{R}^3$, then there exists a vector $\boldsymbol{v}_t$ with norm 1, such that $\langle \boldsymbol{v}_t, \tau(P_i)\rangle \geq 0, i = 1, 2, 3$ and $\langle \boldsymbol{v}_t, \tau(B(\sigma_\pm, t))\rangle \geq 0$. For example, we can take $\boldsymbol{v}_t$ to be any normalized vector in the dual of the cone generated by these five vectors ([3, Chapter 2.6]). By the compactness of the unit sphere, $\{\boldsymbol{v}_t\}_{t \in \mathbb{N}}$ has a limit point $\boldsymbol{v}$. We have $\langle \boldsymbol{v}, \tau(P_i)\rangle \geq 0, i = 1, 2, 3$, so $\boldsymbol{v}$ is not orthogonal to the f-axis, otherwise $\tau(P_i), i = 1, 2, 3$ would all be on the same side of a hyperplane passing through the f-axis, contradicting the fact that their d, e-coordinates generate $\mathbb{R}^2$ as an $\mathbb{R}_{\geq 0}$-cone. Hence, $\langle \boldsymbol{v}, (0, 0, 1)\rangle \neq 0$. Without loss of generality, suppose $\langle \boldsymbol{v}, (0, 0, 1)\rangle < 0$. When $t \to \infty$, the angle between $(0, 0, 1)$ and $\tau(B(\sigma_+, t))$ tends to zero. Therefore, for all large enough t, we have $\langle \boldsymbol{v}, \tau(B(\sigma_+, t))\rangle < 0$. Since $\boldsymbol{v}$ is a limit point of $\{\boldsymbol{v}_t\}_{t \in \mathbb{N}}$, there exists a large enough t such that $\langle \boldsymbol{v}_t, \tau(B(\sigma_+, t))\rangle < 0$. This contradicts the fact that $\langle \boldsymbol{v}_t, \tau(B(\sigma_+, t))\rangle \geq 0$ for all t. ◀

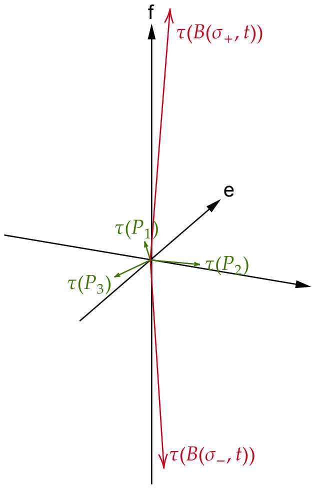


Figure 1 Illustration of the five vectors constructed in Proposition 12.

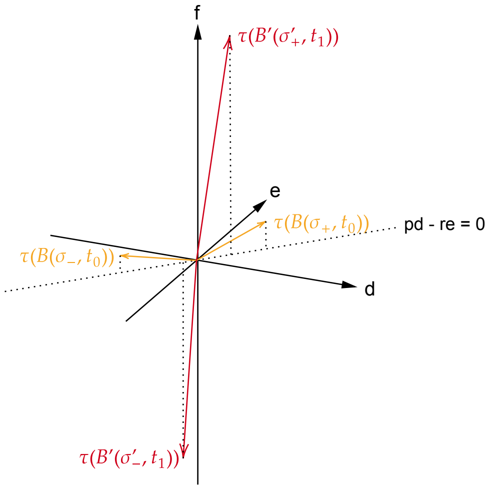


Figure 2 Illustration of the four vectors constructed in Proposition 19.

▶ **Corollary 13.** *When $\mathcal{C}$ has dimension 3, the identity matrix is reachable (and hence also* U_2 *and* U_{10}*).*

Proof. By Proposition 12, one can find $Q_1, Q_2, Q_3, Q_4 \in \langle\mathcal{G}\rangle \cap \mathsf{U}_1$ such that $\tau(Q_i), i = 1, \ldots, 4$ generate $\mathbb{R}^3$ as an $\mathbb{R}_{\geq 0}$-cone. In particular, $-\tau(Q_1) \in \langle \tau(Q_1), \tau(Q_2), \tau(Q_3), \tau(Q_4)\rangle_{\mathbb{R}_{\geq 0}}$. So there exist $x_i \in \mathbb{R}_{\geq 0}, i = 1, \ldots, 4$, not all zero, such that $\sum_{i=1}^4 x_i \tau(Q_i) = \mathbf{0}$. Since $\tau(Q_i), i = 1, \ldots, 4$ have integer entries, one can suppose $x_i \in \mathbb{Z}_{\geq 0}$. Hence, $\tau(\prod_{i=1}^4 Q_i^{x_i}) = \sum_{i=1}^4 x_i \tau(Q_i) = \mathbf{0}$, which yields $I = \prod_{i=1}^4 Q_i^{x_i} \in \langle\mathcal{G}\rangle$. ◀

3.3 $\mathcal{C}$ has dimension 1

Next, we consider the case where $\dim \mathcal{C} = 1$. The main idea of this case is that if the product of a string $B_1 \cdots B_m$ is in U_1, then all $D_\sigma, E_\sigma, F_\sigma$ vanish, so $\tau(B(\sigma, t))$ is determined by some linear terms as well as by G_σ. Recall that we write $A_i = UT(\alpha_i, \beta_i, \kappa_i; \delta_i, \epsilon_i, \phi_i)$, $i = 1, \ldots, k$. Similar to notation (4), we define the following quantities for convenience:

$$\Delta_i \coloneqq \delta_i - \frac{1}{2}\alpha_i\beta_i, \quad \mathcal{E}_i \coloneqq \epsilon_i - \frac{1}{2}\beta_i\kappa_i, \quad \Phi_i \coloneqq \alpha_i\beta_i\kappa_i - \frac{1}{2}(\alpha_i\epsilon_i + \delta_i\kappa_i) + \frac{1}{3}\phi_i. \tag{8}$$

Since $\mathcal{C}$ has dimension 1, there exist $\alpha, \beta, \kappa \in \mathbb{Z}$ such that $\varphi_0(A_i) = (\alpha, \beta, \kappa) \cdot \rho_i$ for $\rho_i \in \mathbb{Z}, i = 1, \ldots, k$.

▶ **Proposition 14.** *Suppose $\varphi_0(A_i) = (\alpha, \beta, \kappa) \cdot \rho_i$ for $\rho_i \in \mathbb{Z}, i = 1, \ldots, k$. Let $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_k)$ be the Parikh vector of a string $B_1 \cdots B_m$, with the product $P = B_1 \cdots B_m$. Then*

(i) *$P \in \mathsf{U}_{10}$ if and only if $\sum_{i=1}^k \ell_i\rho_i = 0$ and $\sum_{i=1}^k \ell_i\Delta_i = 0$.*

(ii) *$P \in \mathsf{U}_2$ if and only if $\sum_{i=1}^k \ell_i\rho_i = 0, \sum_{i=1}^k \ell_i\Delta_i = 0$ and $\sum_{i=1}^k \ell_i\mathcal{E}_i = 0$.*

The immediate consequence of Proposition 14 is that U_2-Reachability and U_{10}-Reachability are decidable using linear programming (LP). For example, U_{10}-Reachability has a positive answer if and only if the LP instance $\sum_{i=1}^k \ell_i\rho_i = 0$, $\sum_{i=1}^k \ell_i\Delta_i = 0$, $\ell_i \geq 0, i = 1, \ldots, k$, has a non-zero *integer* solution $(\ell_1, \ldots, \ell_k)$. However, because all the equations and inequalities in the LP instance are *homogeneous*, the LP instance has a non-zero *integer* solution if and only if it has a non-zero *rational* solution. Furthermore, the total bit length of $\rho_i, \Delta_i, \mathcal{E}_i$ is linear with respect to the encoding size of $\mathcal{G}$. Therefore, the existence of a non-zero rational solution is decidable in polynomial time. In particular, for $i = 1, \dots, k$, one can decide whether this LP instance has a rational solution $(\ell_1, \dots, \ell_k)$ with $\ell_i = 1$. Then, the LP instance has a non-zero rational solution if and only if it has a rational solution with $\ell_i = 1$ for some i. The decision procedure for U_2-Reachability is similar.

Next, we consider the Identity Problem. Define the set

$$\Lambda \coloneqq \left\{ (\ell_1, \dots, \ell_k) \in \mathbb{Z}_{\geq 0}^k \,\middle|\, \sum_{i=1}^k \ell_i \rho_i = \sum_{i=1}^k \ell_i \Delta_i = \sum_{i=1}^k \ell_i \mathcal{E}_i = 0 \right\}.$$

By Proposition 14, the product of a string is in U_2 if and only if its Parikh vector is in Λ. It is easy to see that Λ is additively closed, meaning $\Lambda + \Lambda \subseteq \Lambda$. Define the *support* of a Parikh vector $\boldsymbol{\ell} = (\ell_1, \dots, \ell_k)$ to be $\operatorname{supp}(\boldsymbol{\ell}) = \{i \mid \ell_i \neq 0\}$, and the support of the set Λ to be

$$\operatorname{supp}(\Lambda) \coloneqq \bigcup_{\boldsymbol{\ell} \in \Lambda} \operatorname{supp}(\boldsymbol{\ell}) = \{i \mid \exists (\ell_1, \dots, \ell_k) \in \Lambda, \ell_i \neq 0\}.$$

For $i = 1, \dots, k$, we have $i \in \operatorname{supp}(\Lambda)$ if and only if the LP instance $\sum_{i=1}^k \ell_i \rho_i = \sum_{i=1}^k \ell_i \Delta_i = \sum_{i=1}^k \ell_i \mathcal{E}_i = 0$, $\ell_i > 0$ and $\ell_j \geq 0, j \neq i$ has an *integer* solution. Again, by homogeneity, this is decidable in polynomial time by deciding the existence of a *rational* solution. Hence, $\operatorname{supp}(\Lambda)$ is computable in polynomial time by deciding whether $i \in \operatorname{supp}(\Lambda)$ for all $i = 1, \dots, k$.

If $\operatorname{supp}(\Lambda) \neq \{1, \dots, k\}$, we can discard the elements $A_i \in \mathcal{G}$ with $i \notin \operatorname{supp}(\Lambda)$, then $\operatorname{card}(\mathcal{G})$ decreases and we are done by the induction hypothesis. Hence, we only need to consider the case where $\operatorname{supp}(\Lambda) = \{1, \dots, k\}$. The following proposition answers the Identity Problem in this case. Again, the homogeneity yields a polynomial time deciding procedure.

▶ **Proposition 15.** *Suppose* $\varphi_0(A_i) = (\alpha, \beta, \kappa) \cdot \rho_i$ *for* $\rho_i \in \mathbb{Z}, i = 1, \dots, k$, *and* $\operatorname{supp}(\Lambda) = \{1, \dots, k\}$. *Define the values* $\Gamma_i = \alpha \epsilon_i - \kappa \delta_i, i = 1, \dots, k$. *Then*

- **(i)** *When* $\rho_i \Gamma_j = \rho_j \Gamma_i$ *for all* $i, j \in \{1, \dots, k\}$, *the identity matrix is reachable if and only if the set* $\{(\ell_1, \dots, \ell_k) \in \Lambda \mid \sum_{i=1}^k \ell_i \Phi_i = 0\}$ *is not equal to* $\{\mathbf{0}\}$.
- **(ii)** *When* $\rho_i \Gamma_j \neq \rho_j \Gamma_i$ *for some* $i, j \in \{1, \dots, k\}$, *the identity matrix is reachable.*

3.4 $\mathcal{C}$ has dimension 0

In this case, $\varphi_0(A_i) = \mathbf{0}$ for all i, so $\mathcal{G} \subset \mathsf{U}_1$. Since $\mathsf{U}_1 \overset{\tau}{\cong} \mathbb{Z}^3$, the Identity Problem and U_2, U_{10}-Reachability are decidable using linear programming. For example, deciding the Identity Problem amounts to deciding whether the LP instance $\sum_{i=1}^k \ell_i \cdot \tau(A_i) = \mathbf{0}$, $\ell_i \geq 0, i = 1, \dots, k$ has a non-zero integer solution. As before, by the homogeneity of the LP instance, this is decidable in polynomial time by considering solutions in $\mathbb{Q}$.

3.5 $\mathcal{C}$ has dimension 2

Suppose now that there exist $p, q, r \in \mathbb{Z}$, not all zero, such that $p\alpha_i + q\beta_i + r\kappa_i = 0, i = 1, \dots, k$. Consider the following cases on the values of p, q, r.

3.5.1 Case 1: there is at most one zero among p, q, r

The main difficulty of this case is as follows. By Lemma 7, (D_σ, E_σ) is constrained to the one dimensional subspace $\{(d, e) \mid pd - re = 0\} \subset \mathbb{R}^2$. Therefore, in order to decide whether the vectors $\tau(B(\sigma, t))$ can generate the neutral element, one needs to take into account their linear terms, i.e. $(\sum_{i=1}^m D_i, \sum_{i=1}^m E_i)$ as well. Define the additively closed set:

$$L \coloneqq \left\{ (\ell_1, \dots, \ell_k) \in \mathbb{Z}_{\geq 0}^k \,\middle|\, \sum_{i=1}^k \ell_i \varphi_0(A_i) = 0 \right\}.$$

The product P of a string $B_1 \cdots B_m$ is in U_1 if and only if its Parikh vector is in L.

▶ **Lemma 16.** *When $\mathcal{C} = \mathcal{C}^{lin}$, we have* $\operatorname{supp}(L) = \{1, \dots, k\}$.

We continue to adopt the notations from (8) for $\Delta_i, \mathcal{E}_i$. Consider the subset of L:

$$L_0 := \left\{ (\ell_1, \dots, \ell_k) \in L \,\middle|\, p \sum_{i=1}^{k} \ell_i \Delta_i - r \sum_{i=1}^{k} \ell_i \mathcal{E}_i = 0 \right\}.$$

L_0 can be described as the set of Parikh vectors whose corresponding strings have linear terms falling on the line $pd - re = 0$. Again, L_0 is additively closed. The main idea is that the quadratic term of $\varphi_1(B(\sigma, t))$ falls on the line $pd - re = 0$, therefore, if $P \in \mathsf{U}_2, \varphi_1(B(\sigma, t)) = 0$, then its linear term must also fall on the line $pd - re = 0$. This leads to the following lemma.

▶ **Lemma 17.** *Suppose* $\dim \mathcal{C} = 2$*. If the product P of a string $B_1 \cdots B_m$ is in U_2, then its Parikh vector $\boldsymbol{\ell}$ is in L_0.*

The following proposition gives a solution to the U_{10}-Reachability problem.

▶ **Proposition 18.** *Suppose* $\dim \mathcal{C} = 2$ *and at most one of p, q, r is zero.*

(i) *When $r \neq 0$, U_{10} is reachable.*

(ii) *When $r = 0, p \neq 0$, U_{10} is reachable if and only if L_0 is not equal to $\{\mathbf{0}\}$.*

In particular, whether L_0 equals $\{\mathbf{0}\}$ is decidable by linear programming, (again, by homogeneity, one can solve the linear programming instance in $\mathbb{Q}$). Hence, U_{10}-Reachability is decidable. We then treat the Identity Problem and U_2-Reachability. Consider the support of L_0. As before, $\operatorname{supp}(L_0) = \{i \mid \exists (\ell_1, \dots, \ell_k) \in L_0, \ell_i \neq 0\}$ is computable using linear programming. By Lemma 17, in order to reach U_2 (or the identity matrix), we can only use matrices with index in $\operatorname{supp}(L_0)$. By discarding matrices and using the induction hypothesis on $\operatorname{card}(\mathcal{G})$, we only need to consider the case where $\operatorname{supp}(L_0) = \{1, \dots, k\}$. The following proposition gives a positive answer to the Identity Problem and U_2-Reachability in this case.

▶ **Proposition 19.** *Suppose* $\dim \mathcal{C} = 2$ *and at most one of p, q, r is zero. If* $\operatorname{supp}(L_0) = \{1, \dots, k\}$*, then the identity matrix is reachable. (In particular, U_2 is reachable.)*

Sketch of proof. Similarly to Proposition 12, we construct four elements in $\mathsf{U}_1 \cap \langle \mathcal{G} \rangle$ whose images under τ generate the two-dimensional linear subspace $\{(d, e, f) \in \mathbb{R}^3 \mid pd - re = 0\}$ as an $\mathbb{R}_{\geq 0}$-cone (see Figure 2 for an illustration). Consequently, the $\mathbb{Z}_{\geq 0}$-cone that they generate is two-dimensional lattice in $\{(d, e, f) \in \mathbb{Z}^3 \mid pd - re = 0\}$, which contains the neutral element. ◀

3.5.2 Case 2: $p = r = 0$

In this case, $\mathcal{G} \subset \mathrm{H}_5$, so the Identity Problem is decidable by Lemma 3. U_2 and U_{10}-Reachability reduce to the Identity Problem in $\mathbb{Z}^4$ and $\mathbb{Z}^3$, respectively, which are decidable in polynomial time using linear programming. Here, we claim an additional complexity result that strengthens Lemma 3, which is crucial for a polynomial complexity algorithm for $\mathsf{UT}(4, \mathbb{Z})$.

▶ **Proposition 20.** *For a fixed n, the Identity Problem in H_{2n+1} is decidable in polynomial time.*

3.5.3 Case 3: $p = q = 0$, $r \neq 0$ or $r = q = 0$, $p \neq 0$

The main technique in this case is a reduction from the Identity Problem to U_2-Reachability, from U_2-Reachability to U_{10}-Reachability, and from U_{10}-Reachability to linear programming or to the Identity Problem in H_3. If $p = q = 0$, $r \neq 0$, then $\kappa_i = 0, i = 1, \dots, k$. If $r = q = 0$, $p \neq 0$, then $\alpha_i = 0, i = 1, \dots, k$. Define the following matrices in H_3:

$$H_i \coloneqq \begin{pmatrix} 1 & \alpha_i & \delta_i \\ 0 & 1 & \beta_i \\ 0 & 0 & 1 \end{pmatrix}, \quad i = 1, \dots, k.$$

The following proposition along with Proposition 20 provides a solution to U_{10}-Reachability.

▶ **Proposition 21.**

(i) *When $\kappa_i = 0, i = 1, \dots, k$, U_{10}-Reachability for $A_1, \dots, A_k$ is equivalent to the Identity Problem for $H_1, \dots, H_k$.*

(ii) *When $\alpha_i = 0, i = 1, \dots, k$, U_{10} is reachable for $A_1, \dots, A_k$ if and only if $\sum_{i=1}^{k} \ell_i \delta_i = \sum_{i=1}^{k} \ell_i \beta_i = \sum_{i=1}^{k} \ell_i \kappa_i = 0$ has a non-zero integer solution $(\ell_1, \dots, \ell_k) \in \mathbb{Z}_{\geq 0}^{k}$.*

Next, consider the Identity Problem and U_2-Reachability. By symmetry, we can suppose $p = q = 0$, $r \neq 0$, so $\kappa_i = 0, i = 1, \dots, k$. Define

$$A'_i \coloneqq UT(\beta_i, \alpha_i, \epsilon_i; \delta_i, \phi_i, 0), i = 1, \dots, k, \tag{9}$$

the following proposition reduces the Identity Problem and U_2-Reachability for $A_1, \dots, A_k$ to reachability problems for $A'_1, \dots, A'_k$:

▶ **Proposition 22.** *Suppose $\kappa_i = 0, i = 1, \dots, k$.*

(i) *The Identity Problem for $A_1, \dots, A_k$ is equivalent to U_2-Reachability for $A'_1, \dots, A'_k$.*

(ii) *U_2-Reachability for $A_1, \dots, A_k$ is equivalent to U_{10}-Reachability for $A'_1, \dots, A'_k$.*

Together with the previous Subsections 3.2 - 3.5.2, we have completely reduced the Identity Problem for $\mathcal{G}$ to either the problem for a set of smaller cardinality, or to U_2-reachability of another set. We have also reduced U_2-reachability for $\mathcal{G}$ to either a problem for a set of smaller cardinality, or to U_{10}-reachability of another set. By Proposition 21 and the previous subsections, U_{10}-reachability is decidable. Hence, we have now exhausted all the possible cases for the dimension of $\mathcal{C}$, and we conclude that the Identity Problem, U_2-Reachability and U_{10}-Reachability in $\mathsf{UT}(4, \mathbb{Z})$ are decidable.

4 Complexity analysis and concluding remarks

In this paper, we have shown that the Identity Problem for $\mathsf{UT}(4, \mathbb{Z})$ is decidable. A brief analysis of our algorithm shows that it terminates in polynomial time. In fact, we can first show that the algorithm for U_{10}-Reachability terminates in polynomial time. Starting with $k = \mathrm{card}(\mathcal{G})$ matrices, we need to solve at most $O(k)$ linear equations, $O(k)$ homogeneous linear programming instances and one Identity Problem in H_3 before either $\mathrm{card}(\mathcal{G})$ decreases or a conclusion on U_{10}-Reachability is reached. All these problems have $O(k)$ inputs which are of polynomial size with respect to the coefficients of the matrices in $\mathcal{G}$, and are known to have polynomial complexity. Furthermore, the number $\mathrm{card}(\mathcal{G})$ decreases at most k times. Hence, the total complexity of our algorithm for U_{10}-reachability is polynomial with respect to the input $\mathcal{G}$. Then, using the same method, we can show that the algorithm for U_2-Reachability terminates in polynomial time: since after polynomial time, either $\mathrm{card}(\mathcal{G})$

decreases, or the problem is reduced to U_{10}-Reachability, or a conclusion on U_2-Reachability is reached. At last, we can show that the algorithm for the Identity Problem terminates in polynomial time: after polynomial time, either $\mathrm{card}(\mathcal{G})$ decreases, or the problem is reduced to U_2-Reachability or the Identity Problem in H_5, or a conclusion on the Identity Problem is reached. (In particular, the polynomial complexity of the Identity Problem in H_5 is a new result of our paper, see Proposition 20.)

It is likely that our method can be adapted to study the Identity Problem for other metabelian matrix groups, for instance the direct product H_3^n. There is also evidence that the arguments in this paper can be strengthened to tackle the Identity Problem for $\mathsf{UT}(n, \mathbb{Z})$ with $n \geq 5$, even though $\mathsf{UT}(5, \mathbb{Z})$ ceases to be metabelian. In fact, one can push the convex geometry arguments down the derived series of $\mathsf{UT}(n, \mathbb{Z})$, even when the series has length greater than two. Another natural follow-up question is the Membership Problem for $\mathsf{UT}(4, \mathbb{Z})$. An interesting idea would be to adapt the Register Automata method introduced in [5] for passing from the Identity Problem to the Membership Problem.

References

1 László Babai, Robert Beals, Jin-yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 498–507, 1996.

2 Paul C. Bell and Igor Potapov. On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. *International Journal of Foundations of Computer Science*, 21(06):963–978, 2010.

3 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

4 Christian Choffrut and Juhani Karhumäki. Some decision problems on integer matrices. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 39(1):125–131, 2005.

5 Thomas Colcombet, Joël Ouaknine, Pavel Semukhin, and James Worrell. On reachability problems for low-dimensional matrix semigroups. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 44:1–44:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.44`.

6 Sang-Ki Ko, Reino Niskanen, and Igor Potapov. On the identity problem for the special linear group and the heisenberg group. 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 132:1–132:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.ICALP.2018.132`.

7 Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. *Algebra and Computer Science*, 677:138–153, 2016.

8 V. M. Kopytov. Solvability of the problem of occurrence in finitely generated soluble groups of matrices over the field of algebraic numbers. *Algebra and Logic*, 7(6):388–393, 1968.

9 Engel Lefaucheux. Private Communication, 2022.

10 A. Markov. On certain insoluble problems concerning matrices. *Doklady Akad. Nauk SSSR*, 57(6):539–542, 1947.

11 K. A. Mikhailova. The occurrence problem for direct products of groups. *Matematicheskii Sbornik*, 112(2):241–251, 1966.

12 Michael S. Paterson. Unsolvability in 3 × 3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

13 Igor Potapov and Pavel Semukhin. Decidability of the membership problem for 2 × 2 integer matrices. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–186. SIAM, 2017.

14 Igor Potapov and Pavel Semukhin. Membership problem in GL(2, Z) extended by singular matrices. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 – Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 44:1–44:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.44.

15 Joseph J. Rotman. *An introduction to the theory of groups*, volume 148. Springer Science & Business Media, 2012.

16 Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

Regular Monoidal Languages

Matthew Earnshaw
Department of Software Science, Tallinn University of Technology, Estonia

Paweł Sobociński
Department of Software Science, Tallinn University of Technology, Estonia

Abstract

We introduce regular languages of morphisms in free monoidal categories, with their associated grammars and automata. These subsume the classical theory of regular languages of words and trees, but also open up a much wider class of languages over string diagrams. We use the algebra of monoidal categories to investigate the properties of regular monoidal languages, and provide sufficient conditions for their recognizability by deterministic monoidal automata.

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

Keywords and phrases monoidal categories, string diagrams, formal language theory, cartesian restriction categories

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.44

Related Version *Full Version*: https://arxiv.org/abs/2207.00526 [6]

Funding This research was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19- 0001). The second author was additionally supported by the Estonian Research Council grant PRG1210.

Acknowledgements We would like to thank Ed Morehouse for extensive discussions concerning this work, and Tobias Heindel for discussion of his erstwhile project that partially inspired ours.

1 Introduction

Classical formal language theory has been extended to various kinds of algebraic structures, such as infinite words, rational sequences, trees, countable linear orders, graphs of bounded tree width, etc. In recent years, the essential unity of the field has been better understood [1, 16]. Such structures can often be seen as algebras for monads on the category of sets, and sufficient conditions exist [1] for formal language theory to extend to their algebras.

In this paper, we make a first step into a programme of extending language theory to higher-dimensional algebraic structures. Here we make the step from monoids to 2-monoids, better known as monoidal categories.

We introduce a categorial framework for reasoning about languages of morphisms in strict monoidal categories – including their associated grammars and automata. We show how these include classical and tree automata, but also open up a wilder world of string diagram languages. By investigating the morphisms in monoidal categories from the perspective of language theory, this work contributes to research into the computational manipulation of string diagrams, and so their usage in industrial strength applications. Omitted proofs can be found in the full version [6].

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 44; pp. 44:1–44:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Related work

Bossut [2] studied rational languages of planar acyclic graphs and proved a Kleene theorem for a class of such languages. Bossut's graph languages feature initial and final states, whereas our languages consist of scalar morphisms, which more neatly generalizes the theory of regular string and tree languages. Bossut introduces a notion of automaton for these languages, but these lack a state machine denotation – being more similar to our grammars.

In [10], Heindel recasts Bossut's approach using monoidal categories. Unfortunately the purported Myhill-Nerode result was incorrect, due to a flawed definition of syntactic congruence. We rectify this in Section 5, but a Myhill-Nerode type theorem remains open.

Zamdzhiev [18] introduced context-free languages of string diagrams using the string graph representation of string diagrams and the machinery of context-free graph grammars. In contrast, our approach does not require an intermediate representation of string diagrams as graphs: we work directly with morphisms in monoidal categories. This allows us to use the algebra of monoidal categories to reason about properties of monoidal languages.

Winfree et al. [13] use DNA self-assembly to simulate cellular automata and Wang tile models of computation. The kinds of two-dimensional languages obtained in this way can be seen quite naturally as regular monoidal languages, as illustrated in Example 12.

Walters' note [17] on regular and context-free grammars served as a starting point for our definition of regular monoidal grammar. Rosenthal [12], developing some of the ideas of Walters, defined automata as relational presheaves, which is similar in spirit to our functorial definition of monoidal automata. The framework of Colcombet and Petrişan [5] considering automata as functors is also close in spirit to our definition of monoidal automata. However, all of these papers are directed towards questions involving classical one-dimensional languages, rather than languages of diagrams as in the present paper.

Fahrenberg et al. [7] investigated languages of higher-dimensonal automata, a well-established model of concurrency. We might expect that the investigations of the present paper correspond to a detailed study of a particular low-dimensional case of such languages, but the precise correspondence between these notions is unclear.

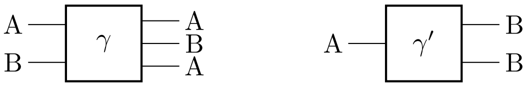
3 Regular monoidal grammars and regular monoidal languages

A *monoidal grammar* is a finite specification for the construction of string diagrams: i.e. morphisms in free monoidal categories (more specifically, free pros). We introduce *regular monoidal grammars*, an analogue of classical (right-) regular grammars, and their equivalent representation as non-deterministic *monoidal automata*. We begin by recalling the notion of monoidal graph and how they present free monoidal categories.

3.1 Monoidal graphs and free pros

▶ **Definition 1.** *A monoidal graph $\mathcal{G}$ consists of sets $E_{\mathcal{G}}, V_{\mathcal{G}}$ and functions $dom, cod : E_{\mathcal{G}} \rightrightarrows V_{\mathcal{G}}^*$ where $V_{\mathcal{G}}^*$ is the underlying set of the free monoid. The elements of $E_{\mathcal{G}}$ are called* generators*, and for a generator $\gamma \in E_{\mathcal{G}}$, $dom(\gamma), cod(\gamma)$ are the domain, codomain (resp.) types of γ.*

Diagrammatically, a monoidal graph can be pictured as a collection of boxes, labelled by elements of $E_{\mathcal{G}}$ with wires entering on the left and exiting on the right, labelled by types given by the functions dom, cod. For example, the following depicts the monoidal graph $\mathcal{G}$ with $E_{\mathcal{G}} = \{\gamma, \gamma'\}, V_{\mathcal{G}} = \{A, B\}, dom(\gamma) = AB, cod(\gamma) = ABA, dom(\gamma') = A, cod(\gamma') = BB$:



Given that we are interested in finite state machines over finite alphabets, we shall work exclusively with finite monoidal graphs, i.e. those in which $E_{\mathcal{G}}$ and $V_{\mathcal{G}}$ are both finite sets.

▶ **Definition 2.** *A morphism $\Psi : \mathcal{G}' \to \mathcal{G}$ of monoidal graphs is a pair of functions $V_\Psi : V_{\mathcal{G}} \to V_{\mathcal{G}'}, E_\Psi : E_{\mathcal{G}} \to E_{\mathcal{G}'}$ such that $dom \fatsemi V_\Psi^* = E_\Psi \fatsemi dom$ and $cod \fatsemi V_\Psi^* = E_\Psi \fatsemi cod$.*

Monoidal graphs and their morphisms form a category $\mathsf{MonGraph}$. Recall that a (coloured) pro is a strict monoidal category whose monoid of objects is free (on the set of "colours"). There is a category Pro with objects pros and morphisms strict monoidal functors whose action on objects is determined by a function between their sets of colours. We call these *pro morphisms*. (Coloured) props are pros that are also *symmetric* (strict) monoidal categories.

Pros (and props) are monadic over monoidal graphs: the forgetful functor $\mathcal{U} : \mathsf{Pro} \to \mathsf{MonGraph}$ has a left adjoint $\mathcal{F} : \mathsf{MonGraph} \to \mathsf{Pro}$, and Pro is equivalent to the category of algebras for the induced monad on $\mathsf{MonGraph}$ (see [8, §2.3]). $\mathcal{F}$ sends a monoidal graph $\mathcal{G}$ to a pro $\mathcal{F}\mathcal{G}$ whose set of objects is $V_{\mathcal{G}}^*$ and whose morphisms are *string diagrams* (see [15]).

3.2 Monoidal languages and regular monoidal grammars

Classically, a language over an alphabet Σ is a subset of the free monoid Σ^*. A *monoidal language* is defined similarly, replacing free monoids with free pros over a *monoidal alphabet*:

▶ **Definition 3.** *A monoidal alphabet Γ is a finite monoidal graph where V_Γ is a singleton.*

For a generator γ of a monoidal alphabet, we refer to $dom(\gamma), cod(\gamma)$ as the arity, coarity (resp.) of γ, writing $ar(\gamma)$, $coar(\gamma)$. Such generators are drawn with "untyped" wires.

▶ **Definition 4.** *A monoidal language L over a monoidal alphabet Γ is a subset $L \subseteq \mathcal{F}\Gamma(0,0)$ of morphisms with arity and coarity 0 in the free pro generated by Γ.*

▶ Remark 5. The restriction to arity and coarity zero (i.e. *scalar*) morphisms may appear arbitrary. However, we will see in Section 4 that this captures and explains the classical definitions of finite-state automata over words and trees. It also leads to more concise definitions in our theory.

Regular monoidal grammars specify monoidal languages that are an analogue of classical regular languages. They can be obtained by taking Walters' [17] definition of regular language and replacing the adjunction between reflexive graphs and categories with that between monoidal graphs and pros. As shown in Section 4, they include the classical definitions of regular tree and word languages as grammars over monoidal alphabets of a particular shape.

▶ **Definition 6.** *A regular monoidal grammar is a morphism of finite monoidal graphs $\Psi : \mathcal{M} \to \Gamma$ where Γ is a monoidal alphabet.*

Intuitively, a regular monoidal grammar is a labelling of the edges of $\mathcal{M}$ by generators in Γ. Indeed, the vertex function $V_\Psi : V_{\mathcal{M}} \to \{\bullet\}$ is unique, so the grammar is determined by its edge function $E_\Psi : E_{\mathcal{M}} \to E_\Gamma$, sending edges to their labels. In Section 3.4 we show that this data determines a transition system with states words $w \in V_{\mathcal{M}}^*$.

▶ Remark 7. Every regular monoidal grammar determines a pro morphism between free pros, $\mathcal{F}\Psi : \mathcal{F}\mathcal{M} \to \mathcal{F}\Gamma$, which we may also refer to as a regular monoidal grammar.

For any string diagram $s \in \mathcal{F}\Gamma$ over an alphabet Γ, we can think of the set of string diagrams $\mathcal{F}\Psi^{-1}(s)$ as a set of possible "parsings" of that diagram.

▶ Remark 8. We represent regular monoidal grammars diagrammatically by drawing the monoidal graph $\mathcal{M}$ as above, but labelling each box $e \in E_{\mathcal{M}}$ with $E_{\Psi}(e)$. The resulting diagram is not in general a diagram of a monoidal graph, since it may contain boxes with the same label but different domain or codomain types. Examples are given below.

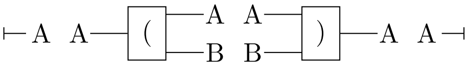
3.3 Regular monoidal languages

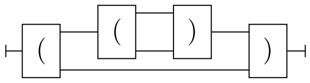
A regular monoidal grammar determines a monoidal language as follows:

▶ **Definition 9.** *Given a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, the image under $\mathcal{F}\Psi$ of the endo-hom-set of the monoidal unit ε in $\mathcal{F}\mathcal{M}$ is a monoidal language $\mathcal{F}\Psi[\mathcal{F}\mathcal{M}(\varepsilon, \varepsilon)] \subseteq \mathcal{F}\Gamma(0, 0)$.*

We call the class of languages determined by regular monoidal grammars the *regular monoidal languages*. We shall see that they are precisely the languages accepted by *non-deterministic monoidal automata* (Section 3.4). The basic idea is that a "word" is a scalar string diagram, i.e. one with no "dangling wires". The language of a monoidal grammar then consists of those scalar string diagrams that can be given a parsing. Parsings can be visually explained using the graphical notation for grammars (Remark 8). A morphism in the language defined by a grammar is any string diagram that can be built using the "typed" building blocks, such that there are no dangling wires, and then erasing the types on the wires. The following examples of regular monoidal grammars illustrate this idea:

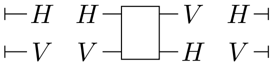
▶ **Example 10** (Balanced parentheses)**.** Recall that the Dyck language, the language of balanced parentheses, is a paradigmatic example of a non-regular word language. However, we can recognize balanced parentheses using the regular monoidal grammar shown below left. An example of a morphism in the language defined by this grammar is shown on the right.

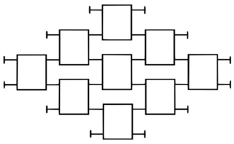




This illustrates how regular monoidal grammars permit unbounded concurrency. Here, as one scans from left to right, the (unbounded) size of the internal boundary of a string diagram keeps track of the number of open left parentheses.

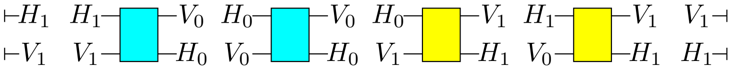
▶ **Example 11** (Brick walls)**.** A variant on the "brick wall" language introduced by [2] is given by the following grammar (left below). An example of a morphism in the language defined by this grammar is shown on the right.

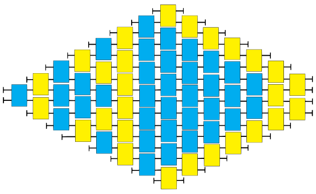




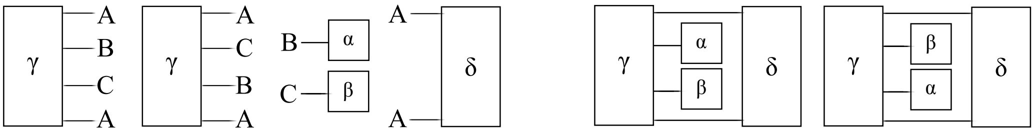
In Section 3.5 we will see how this language of "brick walls" allows us to construct the following example as an intersection of two languages:

▶ **Example 12** (Sierpiński gasket)**.** In [13], self-assembly of DNA tiles was used to realize the behaviour of a cellular automaton that computes the Sierpiński gasket fractal, based on the computation of the XOR gate. [13] implicitly depicts a monoidal grammar, and so Sierpiński gaskets of arbitrary iteration depth (e.g. right below) are in fact the monoidal language over this grammar (left below, where we use colours for the alphabet):





▶ **Example 13.** We define a grammar (left below) that will serve as a running counterexample in Section 7, as it defines a language that cannot be deterministically recognized. The connected string diagrams in this language are exactly two (right below).



▶ Remark 14. If the monoidal graph $\mathcal{M}$ has no edges whose domain is ε and no edges whose codomain is ε, a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$ will define a language containing only the identity on the monoidal unit, i.e. the empty string diagram (denoted). In fact, *every* monoidal language contains the empty string diagram.

3.4 Non-deterministic monoidal automata

Recall that a non-deterministic finite automaton (NFA) is given by a finite set Q of states, an initial state $i \in Q$, a set of final states $F \subseteq Q$, and for each $a \in \Sigma$, a function $Q \xrightarrow{\Delta_a} \mathcal{P}(Q)$. Non-deterministic *monoidal automata* do not have initial and final states; string diagrams are simply accepted or rejected depending on their shape. In Section 4, we will see that initial and final states derive from this definition, when the alphabet is of a particular form.

▶ **Definition 15.** *A non-deterministic monoidal automaton $\Delta = (Q, \Delta_\Gamma)$ over a monoidal alphabet Γ is given by a finite set Q, together with a set of transition functions indexed by generators $\Delta_\Gamma = \{Q^{ar(\gamma)} \xrightarrow{\Delta_\gamma} \mathcal{P}(Q^{coar(\gamma)})\}_{\gamma \in E_\Gamma}$.*

For classical NFAs, the assignment $a \mapsto \Delta_a$ extends uniquely to a functor $\Sigma^* \to \mathsf{Rel}$, the inductive extension of the transition structure from letters to words. We define the inductive extension of monoidal automata from generators to string diagrams. First recall the definition of the endomorphism pro of an object in a monoidal category:

▶ **Definition 16.** *Let $\mathcal{C}$ be a monoidal category, and Q an object of $\mathcal{C}$. The endomorphism pro of Q, $\mathcal{C}_Q$, has natural numbers as objects, hom-sets $\mathcal{C}_Q(n,m) := \mathcal{C}(Q^n, Q^m)$, composition and identities as in $\mathcal{C}$. The monoidal product is addition on objects, and as in $\mathcal{C}$ on morphisms.*

The codomains of our inductive extension will be endomorphism pros of finite sets Q in Rel, considered as the Kleisli category of the powerset monad $\mathcal{P}$. Since $\mathcal{P}$ is a commutative monad (with respect to the cartesian product of sets, with $\mathcal{P}X \times \mathcal{P}Y \to \mathcal{P}(X \times Y)$ given by the product of subsets), the following lemma gives us the monoidal structure on Rel:

▶ **Lemma 17** ([11], Corollary 4.3). *Let T be a commutative monad on a symmetric monoidal category $\mathcal{C}$. Then the Kleisli category $\mathsf{Kl}(T)$ has a canonical monoidal structure, which is given on objects by the monoidal product in $\mathcal{C}$, and on morphisms $f : X \to TA, g : Y \to TB$ by $X \otimes Y \xrightarrow{f \otimes g} TA \otimes TB \xrightarrow{\nabla} T(A \otimes B)$, where ∇ is given by the commutativity of T.*

▶ Remark 18. The maybe monad $(-)_\perp$ is also commutative, so its Kleisli category, equivalent to the category Par of sets and partial functions, also has a canonical monoidal structure, and for each set Q there is an endomorphism pro Par_Q. We will come back to Par_Q in Section 6.

Now we can define the inductive extension of a non-deterministic monoidal automaton:

▶ **Observation 19.** *The assignment of generators to transition functions $\gamma \mapsto \Delta_\gamma$ in Definition 15 determines a morphism of monoidal graphs $\Gamma \to |\mathsf{Rel}_Q|$. Such morphisms are in bijection with pro morphisms $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$. We will also refer to the inductive extension Δ as a non-deterministic monoidal automaton, and sometimes write Δ_α for the relation $\Delta(\alpha : n \to m)$.*

A scalar string diagram is mapped to one of the two possible nullary relations $\{\bullet\} \to \mathcal{P}(\{\bullet\})$, which represent accepting or rejecting computations, and thus can be used to define the language of the automaton:

▶ **Definition 20.** *Let $\Delta : \mathcal{F}\Gamma \to \mathit{Rel}_Q$ be a non-deterministic monoidal automaton. Then the monoidal language accepted by Δ is $\mathcal{L}(\Delta) := \{\alpha \in \mathcal{F}\Gamma(0,0) \mid \Delta_\alpha(\bullet) = \{\bullet\}\}$.*

There is an evident correspondence between regular monoidal grammars and non-deterministic monoidal automata. The graphical representation of a grammar makes this most clear: it can also be thought of as the "transition graph" of a non-deterministic monoidal automaton. More explicitly we have:

▶ **Proposition 21.** *Given a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, define a monoidal automaton with $Q = V_\mathcal{M}$, $w(\Delta_\gamma)w' \iff \exists \sigma \in E_\Psi^{-1}(\gamma)$ such that $dom(\sigma) = w, cod(\sigma) = w'$. Conversely given a monoidal automaton (Q, Δ_Γ), define a regular monoidal grammar with $V_\mathcal{M} = Q$ and take an edge $w \to w'$ over $\gamma \iff w(\Delta_\gamma)w'$. This correspondence of grammars and automata preserves the recognized language.*

▶ Remark 22. In automata theory it is often convenient to consider automata with ε-transitions, or word-labelled transitions more generally. As monoidal grammars, these correspond to arbitrary functors $\mathcal{F}\mathcal{M} \to \mathcal{F}\Gamma$, that is (by the adjunction $\mathcal{U} \dashv \mathcal{F}$), to morphisms of finite monoidal graphs $\mathcal{M} \to \mathcal{U}\mathcal{F}\Gamma$. The corresponding generalization of monoidal automata requires considering Rel_Q as a monoidal 2-category with 2-cells the inclusions. Identity on objects, strict monoidal lax 2-functors $\mathcal{F}\Gamma \to \mathsf{Rel}_Q$ (where $\mathcal{F}\Gamma$ is considered as equipped with identity 2-cells), then give the refined notion of monoidal automaton. Such a lax 2-functor need no longer send the identity on n wires to the identity relation on Q^n, but merely to a relation that includes the identity; this corresponds to allowing silent transitions. Similarly, *lax* preservation of composition corresponds to allowing "term-labelled" transitions.

3.5 Closure properties of regular monoidal languages

We record some closure properties of regular monoidal languages.

▶ **Lemma 23** (Closure under union)**.** *Let L and L' be regular monoidal languages over Γ. Then $L \cup L'$ is a regular monoidal language over Γ.*

▶ **Lemma 24** (Closure under intersection)**.** *Let L and L' be regular monoidal languages over Γ. Then $L \cap L'$ is a regular monoidal language over Γ.*

▶ Remark 25. The Sierpiński gasket language (Example 12) is the intersection of the brick wall language (Example 11) and an "XOR gate" language: this explains the origin of the states in the grammar shown in Example 12.

▶ **Lemma 26** (Closure under monoidal product and factors)**.** *Let L be a regular monoidal language. Then $\alpha, \beta \in L \iff \alpha \otimes \beta \in L$.*

▶ **Lemma 27** (Closure under images of alphabets)**.** *Let L a be regular monoidal language over Γ, and $\Gamma \xrightarrow{h} \Gamma'$ be a morphism of monoidal alphabets. Then $(\mathcal{F}h)L$ is a regular monoidal language over Γ'.*

▶ **Lemma 28** (Closure under preimages of alphabets)**.** *Let L a regular monoidal language over Γ, and $\Gamma' \xrightarrow{h} \Gamma$ be a morphism of monoidal alphabets. Then the inverse image of L, $(\mathcal{F}h)^{-1}(L)$ is a regular monoidal language over Γ'.*

Closure under complement is often held to be an important criterion for what should count as a *recognizable* language. Indeed, for the abstract monadic second order logic introduced in [1], it is a *theorem* that the class of recognizable languages relative to a monad on Set is closed under complement. However, given that every monoidal language contains the empty string diagram, we obviously have that:

▶ **Observation 29.** *Regular monoidal languages are not closed under complement.*

This suggests that there is no obvious account of regular monoidal languages in terms of monadic second order logic. On the other hand, there is no reason we should expect even the general account of monadic second order logic given in [1] to extend to monoidal categories, since these are not algebras for a monad on Set. Moreover, taking inspiration from classical examples in Section 4, one could also refine what is meant by complement, for instance focussing on the set of non-empty connected scalar diagrams – see below for more details.

4 Regular word and tree languages as regular monoidal languages

Classical non-deterministic finite-state automata and tree automata can be seen as non-deterministic monoidal automata over alphabets of a particular shape.

To make the correspondence precise, in the following we restrict monoidal languages to their *connected* string diagrams. Strictly speaking, the language of a monoidal automaton always contains only the empty diagram or is countably infinite, because if α is accepted by the automaton, so are arbitrary finite monoidal products $\alpha \otimes \cdots \otimes \alpha$. However, it is of course possible for a monoidal language to consist of a finite number of connected string diagrams.

From another perspective, without restricting to connected components, we can say that the monoidal automata corresponding to finite-state and tree automata have the power of an unbounded number of such classical automata running in parallel.

4.1 Finite-state automata

▶ **Definition 30.** *A word monoidal alphabet is a monoidal alphabet having only generators of arity and coarity 1,* $-\boxed{\sigma}-$ *, along with a single "start" generator* $\vdash\!-$ *of arity 0 and coarity 1, and "end" generator* $-\!\dashv$ *of arity 1 and coarity 0.*

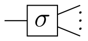
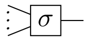
▶ **Observation 31.** *Non-deterministic monoidal automata over word monoidal alphabets correspond to classical NFAs.*

Let an NFA $A = (Q, \Sigma, \Delta, i, F)$ be given. We build a monoidal automaton as follows. Form the monoidal alphabet Σ' by starting with generators $\vdash\!-$, $-\!\dashv$ and adding generators $-\boxed{\sigma}-$ for each $\sigma \in \Sigma$. For each $-\boxed{\sigma}-$, take the transition function $\Delta_\sigma := \Delta(\sigma, -) : Q \to \mathcal{P}(Q)$. For $-\!\dashv$ take the transition function $Q \to \mathcal{P}(Q^0)$ to be the characteristic function of $F \subseteq Q$, sending elements of F to $\{\bullet\}$ and to $\varnothing$ otherwise, and for $\vdash\!-$ take the function $Q^0 \to \mathcal{P}(Q)$ to pick out the singleton $\{i\}$. This defines a monoidal automaton $A' := (Q, \Delta'_{\Sigma'})$, and a simple induction shows that $\mathcal{L}(A) = \mathcal{L}(A')$, if one restricts to connected string diagrams.

Conversely, the data of a monoidal automaton over a word monoidal alphabet corresponds to the data of an NFA, the only difference being that the transition function associated to $\vdash\!-$ picks out a *set* of initial states $\{\bullet\} \to \mathcal{P}(Q)$. We can always "normalize" such an automaton into an equivalent NFA with one initial state (see [14, §2.3.1]). This shows how NFA initial and final states are captured by this particular shape of monoidal alphabet.

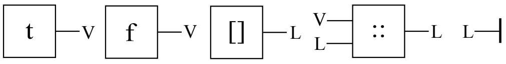
4.2 Tree automata

Recall that non-deterministic finite tree automata come in two flavours, bottom-up and top-down, depending on whether they process a tree starting at the leaves or at the root, respectively. A non-deterministic bottom-up finite tree automaton is given by a finite set of states Q, a "ranked" alphabet $(\Sigma, r : \Sigma \to \mathbb{N})$, a set of final states $F \subseteq Q$, and for each $\sigma \in \Sigma$ a transition function $\Delta_\sigma : Q^{r(\sigma)} \to \mathcal{P}(Q)$. A non-deterministic top-down tree automaton, instead, has a set of initial states $I \subseteq Q$ and transition functions $\Delta_\sigma : Q \to \mathcal{P}(Q^{r(\sigma)})$. We can recover these as non-deterministic monoidal automata over *tree monoidal alphabets*:

▶ **Definition 32.** *A top-down tree monoidal alphabet is a monoidal alphabet having only generators of arity 1 (and arbitrary coarities $\geqslant 0$),*  *, along with a single "root" generator* $\vdash\!\!-$ *. Analogously, a bottom-up tree monoidal alphabet is a monoidal alphabet having only generators of coarity 1 (and arbitrary arities $\geqslant 0$),*  *, along with a single "root" generator* $-\!\!\dashv$ *.*

▶ **Observation 33.** *Bottom-up tree automata are exactly non-deterministic monoidal automata over bottom-up tree monoidal alphabets, and likewise for top-down tree automata.*

The idea is similar to that sketched above for NFAs. For example, consider the following graph of a monoidal automaton over a bottom-up tree monoidal alphabet, recognizing trees corresponding to terms of the inductive type of lists of boolean values (a list may be empty, [], or be a boolean value "consed" onto a list via ::).

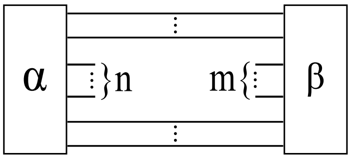


Intuitively, the connected scalar string diagrams determined by this language are trees, with leaves on the left, and the root on the right. Monoidal automata over top-down tree monoidal alphabets have a similar form, but are mirrored horizontally, and thus morphisms in the language have the root on the left, and leaves on the right, and monoidal automata read the morphism starting at the root.

5 The syntactic pro of a monoidal language

In this section we introduce the *syntactic congruence* on monoidal languages and the corresponding *syntactic pro*, by analogy with the syntactic congruence on classical regular languages and their associated syntactic monoid. In Section 7.2 we will give an algebraic property of the syntactic pro sufficient for the language to be deterministically recognizable.

▶ **Definition 34.** *A context of capacity (n, m), where $n, m \geqslant 0$, is a scalar string diagram with a hole – as illustrated below – with zero or more additional wires exiting the first box and entering the second (indicated by ellipses).*



Given a context of capacity (n, m), we can fill the hole with a string diagram $\alpha : n \to m$. Write $C[\alpha]$ for the resulting string diagram. Note that the empty diagram is a context, the empty context. Contexts allow us to define contextual equivalence of string diagrams:

▶ **Definition 35** (Syntactic congruence)**.** *Given a monoidal language $L \subseteq \mathcal{F}\Gamma(0,0)$ we define its syntactic congruence $\equiv_L$ as follows. Let α, β be morphisms in $\mathcal{F}\Gamma(n,m)$. Then $\alpha \equiv_L \beta$ whenever $C[\alpha] \in L \iff C[\beta] \in L$, for all contexts C of capacity (n,m).*

▶ **Definition 36.** *The syntactic pro of a monoidal language L is the quotient pro $\mathcal{F}\Gamma/{\equiv_L}$. The quotient functor $S_L : \mathcal{F}\Gamma \to \mathcal{F}\Gamma/{\equiv_L}$ is the syntactic morphism of L. For more details, see the full version [6].*

▶ Remark 37. The syntactic congruences for classical regular languages of words and trees are also special cases of this congruence over word and tree monoidal alphabets.

▶ **Lemma 38.** *L is the inverse image along the syntactic morphism of the equivalence class of the empty diagram.*

Proof. Let $\alpha \in L$. Then $\alpha \equiv_L \boxed{}$, since the empty diagram is in every language and if C is a context of capacity $(0,0)$ distinguishing α and $\boxed{}$, then we have a contradiction by Lemma 26. So $\alpha \in S_L^{-1}(\left[\boxed{}\right])$, and conversely. ◀

In the terminology of algebraic language theory, we say that the syntactic morphism *recognizes* L. A full investigation of algebraic recognizability of monoidal languages is a topic for future work. For now, we record the following lemma which is needed for Theorem 59:

▶ **Lemma 39.** *If a monoidal language L is regular, then its syntactic pro $\mathcal{F}\Gamma/{\equiv_L}$ is locally finite (i.e. has finite hom-sets).*

Proof. It suffices to exhibit a full pro morphism into $\mathcal{F}\Gamma/{\equiv_L}$ from a locally finite pro. Let L be a regular monoidal language recognized by $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$. Δ induces a congruence $\sim$ on $\mathcal{F}\Gamma$ defined by $\alpha \sim \beta \iff \Delta(\alpha) = \Delta(\beta)$, which implies that $\mathcal{F}\Gamma/{\sim}$ is locally finite, since Rel_Q is locally finite. Define the pro morphism $\mathcal{F}\Gamma/{\sim} \to \mathcal{F}\Gamma/{\equiv_L}$ to be identity on objects and $[\alpha]_\sim \mapsto [\alpha]_{\equiv_L}$ on morphisms. This is well-defined since if $\alpha \sim \beta$ and $C[\alpha] \in L$ for some context C, then by functoriality $C[\beta] \in L$. Clearly it is full, so $\mathcal{F}\Gamma/{\equiv_L}$ is locally finite. ◀

6 Deterministic monoidal automata

Classically, the expressive equivalence of deterministic and non-deterministic finite-state automata for string languages is well known, but already for trees, top-down deterministic tree automata are less expressive than bottom-up deterministic tree automata. Therefore we cannot expect to determinize non-deterministic monoidal automata. However, we have already seen monoidal languages that are deterministically recognizable (Examples 10, 11, 12, interpreted as the transition relations of monoidal automata, are functional relations). Here we introduce deterministic monoidal automata and show that their languages enjoy the property of *causal closure*. In Section 7 we consider the question of determinizability.

▶ **Definition 40.** *A deterministic monoidal automaton $\delta = (Q, \delta_\Gamma)$ over a monoidal alphabet Γ is given by a finite set Q, together with transition functions $\delta_\Gamma = \{Q^{ar(\gamma)} \xrightarrow{\delta_\gamma} Q_\perp^{coar(\gamma)}\}_{\gamma \in \Gamma}$.*

Recall the definition of the pro Par_Q from Remark 18. Then as in Observation 19, such assignments $\gamma \mapsto \delta_\gamma$ uniquely extend to pro morphisms $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$, and we will also refer to such pro morphisms as deterministic monoidal automata. δ maps scalar string diagrams to one of the two functions $Q^0 \to Q_\perp^0$, and we use this to define the language of the automaton:

▶ **Definition 41.** *Let $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$ be a deterministic monoidal automaton. Then the language accepted by δ is $\mathcal{L}(\delta) := \{\alpha \in \mathcal{F}\Gamma(0,0) \mid \delta_\alpha(\bullet) = \bullet\}$.*

We give a necessary condition for a monoidal language to be recognized by a deterministic monoidal automaton. The idea is to generalize the characterization of top-down deterministically recognizable tree languages as those that are closed under the operation of splitting a tree language into the set of possible paths through the trees, and reconstituting trees by grafting compatible paths [9]. For string diagrams, we call the analogue of paths through a tree the *causal histories* of a diagram (Definition 46).

First, we briefly recall the machinery of (cartesian) *restriction categories* [3], that will be necessary in the following. Restriction categories are an abstraction of the category of partial functions, and provide us with a diagrammatic calculus for reasoning about determinization of monoidal languages.

▶ **Definition 42** ([4]). *A cartesian restriction prop is a prop in which every object is equipped with a commutative comonoid structure (with the counit depicted by ─● , comultiplication by ─●< , and symmetry by ⨯) that is coherent, and for which the comultiplication is natural (for more details, see the full version [6]).*

▶ **Definition 43.** *The free cartesian restriction prop on a monoidal graph $\mathcal{M}$, denoted $\mathcal{F}_{\downarrow}\mathcal{M}$ is given by taking the free prop on the monoidal graph $\mathcal{M}$ extended with a comultiplication and counit generator for every object in $V_{\mathcal{M}}$, and quotienting the morphisms by the structural equations of cartesian restriction categories (for more details, see the full version [6]).*

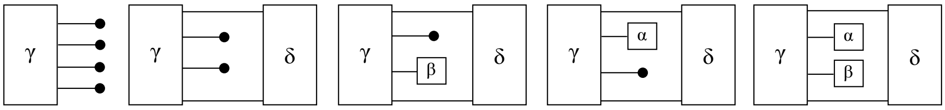
▶ Remark 44. Par is the paradigmatic example of a cartesian restriction category, with ─● on X given by the relation $X \to \{\bullet, \bot\}$ sending every element to •, and ─●< given by the diagonal relation. Par_Q inherits this structure and so is a cartesian restriction prop. Therefore deterministic monoidal automata (Q, δ_Γ) also have inductive extensions to morphisms of cartesian restriction props, $\overline{\delta} : \mathcal{F}_{\downarrow}\Gamma \to \mathsf{Par}_Q$, and these have a obvious notion of associated language, defined similarly to Definition 41. These are related by the following lemma, which follows from the universal properties of $\mathcal{F}\Gamma$ and $\mathcal{F}_{\downarrow}\Gamma$:

▶ **Lemma 45.** *If (Q, δ_Γ) is a deterministic monoidal automaton, then δ factors through $\overline{\delta}$ as $\delta = \mathcal{H}_\Gamma \,\mathring{,}\, \overline{\delta}$, where $\mathcal{H}_\Gamma : \mathcal{F}\Gamma \to \mathcal{F}_{\downarrow}\Gamma$ sends morphisms to their equivalence class in $\mathcal{F}_{\downarrow}\Gamma$.*

Recall that any restriction category is poset-enriched: $f \leqslant g$ if f is "less defined" than g, i.e. if f coincides with g on f's domain of definition. For the hom-set from the monoidal unit to itself, we have $f \leqslant g \iff f \otimes g = f$. Now we can define causal histories:

▶ **Definition 46.** *Let γ be a string diagram in $\mathcal{F}\Gamma(0,0)$. We call a string diagram h in $\mathcal{F}_{\downarrow}\Gamma(0,0)$ a causal history of γ if $\mathcal{H}_\Gamma(\gamma) \leqslant h$ in $\mathcal{F}_{\downarrow}\Gamma(0,0)$. Let $L \subseteq \mathcal{F}\Gamma(0,0)$ be a regular monoidal language. The set of causal histories of L, denoted $ch(L)$, is defined to be $\mathcal{H}_\Gamma(L)^{\uparrow}$, the upwards closure of $\mathcal{H}_\Gamma(L)$ in the poset $\mathcal{F}_{\downarrow}\Gamma(0,0)$.*

A causal history represents the possible causal influence of parts of a diagram on generators appearing "later" in the diagram. For example, the following five string diagrams are causal histories of the rightmost string diagram below (every diagram is a causal history of itself), taken from the language introduced in Example 13:

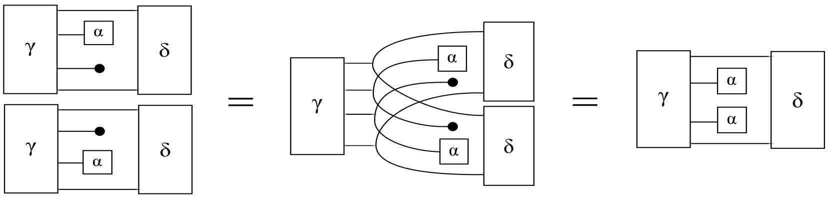


▶ **Lemma 47.** *Let $M = (Q, \delta_\Gamma)$ be a deterministic monoidal automaton, with functors $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$, $\overline{\delta} : \mathcal{F}_{\downarrow}\Gamma \to \mathsf{Par}_Q$. Then if δ accepts γ, $\overline{\delta}$ accepts all causal histories of γ.*

Proof. Since $\delta = \mathcal{H}_\Gamma \mathbin{⨟} \overline{\delta}$, if δ accepts γ, then $\overline{\delta}$ accepts $\mathcal{H}_\Gamma(\gamma)$. Let h be a causal history of γ. Then $\overline{\delta}(\mathcal{H}_\Gamma(\gamma)) = \overline{\delta}(h \otimes \mathcal{H}_\Gamma(\gamma)) = \overline{\delta}(h) \otimes \overline{\delta}(\mathcal{H}_\Gamma(\gamma))$. But then $\overline{\delta}$ accepts h by Lemma 26. ◀

▶ **Definition 48** (Causal closure of a language). *Let L be a monoidal language over a monoidal alphabet Γ. Let $\bigotimes ch(L)$ denote the closure of the set of causal histories of L under monoidal product. Then the causal closure of L is $\mathcal{H}_\Gamma^{-1} \bigotimes ch(L)$. A monoidal language is causally closed if it is equal to its causal closure.*

To illustrate causal closure, consider the following figure, which shows part of the derivation of a morphism in the causal closure of the language of Example 13:



The leftmost diagram depicts the monoidal product of two causal histories determined by the counterexample language. By the equational theory of cartesian restriction categories (see the full version [6]), this is equal to the string diagrams in the center and on the right, where we first apply the naturality of $-\!\!\bullet\!\!\!<$ (for γ), then unitality (twice), then naturality of $-\!\!\bullet\!\!\!<$ (for δ). The rightmost form of the diagram exhibits this morphism as being in the image of $\mathcal{H}_\Gamma$, and its preimage under $\mathcal{H}_\Gamma$ is the same diagram in $\mathcal{F}\Gamma$. Since this diagram is not in the original language, the language is not causally closed.

▶ **Theorem 49.** *If a monoidal language is recognized by a deterministic monoidal automaton, then it is causally closed.*

Proof. Let L be recognized by a deterministic monoidal automaton $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$. We have $\delta = \mathcal{H}_\Gamma \mathbin{⨟} \overline{\delta}$ and from Lemma 47 that $\overline{\delta}$ accepts causal histories of morphisms in L. Since languages are closed under monoidal product (Lemma 26), then by definition of the causal closure, δ must accept everything in the causal closure of L. ◀

7 Deterministically recognizable monoidal languages

Non-deterministic finite state automata for words and bottom-up trees can be determinized via the well known powerset construction. However, top-down tree automata cannot be determinized in general [9, §2.11], so general monoidal automata also cannot be determinized (Observation 33). However, there are interesting examples of deterministically recognizable monoidal languages that are not tree languages, such as the monoidal Dyck language (Example 10) and Sierpiński gaskets (Example 12), and it is an intriguing theoretical challenge to characterize such languages.

In Section 7.1 we study a class of determinizable automata called *convex* automata. In Section 7.2 we give a sufficient condition for a language to be deterministically recognizable.

7.1 Convex automata and the powerset construction

The classical powerset construction is given conceptually by composition with the functor $\mathsf{Rel} \to \mathsf{Set}$, right adjoint to the inclusion $\mathsf{Set} \hookrightarrow \mathsf{Rel}$. As remarked above, we cannot hope to obtain an analogue of this functor for monoidal automata. Thus we describe a suitable subcategory of Rel_Q for which determinization is functorial, that of convex relations.

▶ **Definition 50.** *A relation $\Delta : Q^n \to \mathcal{P}(Q^m)$ is convex if there is a morphism Δ^* such that the following square commutes:*

$$\begin{tikzcd}
(\mathcal{P}Q)^n \arrow[r, "\Delta^*"] \arrow[d, "\nabla"'] & (\mathcal{P}Q)^m \arrow[d, "\nabla"] \\
\mathcal{P}(Q^n) \arrow[r, "\Delta^{\#}"] & \mathcal{P}(Q^m)
\end{tikzcd}$$

where $\Delta^{\#}$ is the Kleisli lift of Δ, and ∇ is the monoidal multiplication given by the commutativity of the powerset monad.

▶ **Observation 51.** *If Δ is convex, the morphism Δ^* is unique, since ∇ is a monomorphism.*

▶ **Example 52.** The relation $\Delta_\gamma : Q^0 \to \mathcal{P}(Q^4)$ induced by the grammar in Example 13 is not convex, since (A, B, B, A) and (A, C, C, A), which we can think of as "convex combinations" of the other state vectors, are not included in the image of the relation.

▶ **Lemma 53.** *Convex relations determine a sub-pro $\mathsf{CRel}_Q \hookrightarrow \mathsf{Rel}_Q$.*

▶ **Definition 54.** *An automaton $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$ is convex if it factors through CRel_Q.*

The following lemma gives the powerset construction on convex automata. We use the non-empty powerset $\mathcal{P}^+$ to avoid duplication of failure state ($\varnothing$ in Rel_Q, but $\perp$ in $\mathsf{Par}_{\mathcal{P}^+(Q)}$):

▶ **Lemma 55.** *For each set Q there is a morphism of pros $\mathcal{D}_Q : \mathsf{CRel}_Q \to \mathsf{Par}_{\mathcal{P}^+(Q)}$ which is identity on objects and acts as follows on morphisms:*

$$\Delta_\alpha : Q^n \to \mathcal{P}(Q^m)$$
$$\downarrow$$
$$\mathcal{P}^+(Q)^n \xrightarrow{\eta^n} (\perp\mathcal{P}^+(Q))^n \xrightarrow{\cong} \mathcal{P}(Q)^n \xrightarrow{\Delta^*_\alpha} \mathcal{P}(Q)^m \xrightarrow{\cong} (\perp\mathcal{P}^+(Q))^m \xrightarrow{\nabla} \perp\mathcal{P}^+(Q)^m$$

where $\perp$ is the maybe monad, η is the unit of this monad, and ∇ is its monoidal multiplication with respect to the cartesian product.

Determinization of a convex automaton $\Delta : \mathcal{F}\Gamma \to \mathsf{CRel}_Q$ is now just given by post-composition with the functor $\mathcal{D}_Q$. We show that this preserves the language:

▶ **Theorem 56.** *Determinization of convex automata preserves the accepted language: let $\Delta : \mathcal{F}\Gamma \to \mathsf{CRel}_Q$ be a convex automaton, then $\mathcal{L}(\Delta) = \mathcal{L}(\Delta \,⨟\, \mathcal{D}_Q)$.*

Proof. Let $\alpha \in \mathcal{L}(\Delta)$, i.e. $\Delta_\alpha(\bullet) = \{\bullet\}$. Then we must have $\Delta^*_\alpha(\bullet) = \bullet$, and so $(\Delta \,⨟\, \mathcal{D}_Q)_\alpha(\bullet) = \bullet$. Conversely let $\alpha \in \mathcal{L}_\mathcal{D}(\Delta \,⨟\, \mathcal{D}_Q)$, i.e. $(\Delta \,⨟\, \mathcal{D}_Q)_\alpha(\bullet) = \bullet$. Then we must have that $\Delta^*_\alpha(\bullet) = \bullet$, and so $\Delta_\alpha(\bullet) = \{\bullet\}$, that is $\alpha \in \mathcal{L}(\Delta)$. ◀

▶ **Example 57.** Non-deterministic monoidal automata over word monoidal alphabets (Definition 30) are convex: for a relation $\Delta : Q \to \mathcal{P}(Q)$, Δ^* is given by the Kleisli extension of Δ. This reflects the well known determinizability of classical finite-state automata.

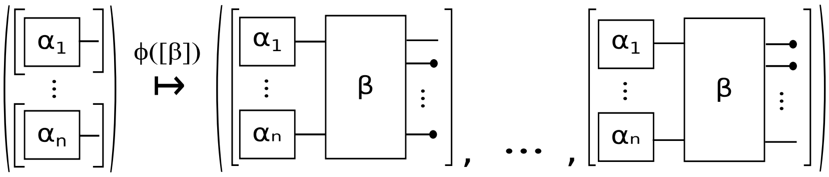
▶ **Example 58.** Similarly, non-deterministic monoidal automata over bottom-up tree monoidal alphabets (Definition 32) are convex, with $\Delta^* := \nabla \,⨟\, \Delta^{\#}$. For top-down tree monoidal alphabets, the general obstruction to convexity (and thus determinizability) is seen as the non-existence of a left inverse of ∇.

7.2 A sufficient condition for deterministic recognizability

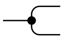
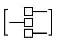
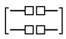
▶ **Theorem 59.** *If the syntactic pro of a regular monoidal language has the structure of a cartesian restriction prop, then the language is recognizable by a deterministic monoidal automaton.*

Proof. Let L be a monoidal language such that $\mathcal{F}\Gamma/{\equiv_L}$ has a cartesian restriction prop structure. We exhibit a pro morphism $\mathcal{F}\Gamma/{\equiv_L} \xrightarrow{\phi} \mathsf{Par}_Q$ such that $\mathcal{F}\Gamma \xrightarrow{S_L} \mathcal{F}\Gamma/{\equiv_L} \xrightarrow{\phi} \mathsf{Par}_Q$ is a deterministic monoidal automaton accepting exactly L.

Let $Q := \mathcal{F}\Gamma/{\equiv_L}(0,1)$. By Lemma 39, this is a finite set. For $m > 0$ and $[\beta] \in \mathcal{F}\Gamma/{\equiv_L}(n,m)$, define $\phi([\beta]) : n \to m$ to be the following map from $Q^n \to Q^m_\perp$:



When $m = 0$ (i.e. $[\beta]$ has coarity 0), let $\phi([\beta])([\alpha_1], ..., [\alpha_n]) = \bullet$, if $[(\alpha_1 \otimes ... \otimes \alpha_n) \mathbin{⨟} \beta] =$ [⬚], and $\phi([\beta])([\alpha_1], ..., [\alpha_n]) = \perp$ otherwise. The proof that this defines a morphism of pros is an exercise in diagrammatic reasoning using the equational theory of cartesian restriction categories, see the full version [6]. To see that this automaton accepts exactly L, let $\alpha \in \mathcal{L}(S_L \mathbin{⨟} \phi)$, then by definition we must have $S_L(\alpha) =$ [⬚], and so $\alpha \in L$ (by Lemma 38). Conversely let $\alpha \in L$, then $S_L(\alpha) =$ [⬚] and by definition ϕ ([⬚]) $(\bullet) = \bullet$, so $\alpha \in \mathcal{L}(S_L \mathbin{⨟} \phi)$. Therefore $S_L \mathbin{⨟} \phi$ is a deterministic monoidal automaton recognizing L. ◀

▶ **Example 60.** A simple example is the language L of "bones" over the monoidal alphabet $\Gamma = \{$ ▫–, –▫ $\}$, having one connected component: ▫—▫ . The syntactic pro $\mathcal{F}\Gamma/{\equiv_L}$ has a cartesian restriction prop structure, with the counit –• given by the equivalence class [–▫], comultiplication  by , and symmetry by . It is clear that $\mathcal{F}\Gamma/{\equiv_L}(0,1)$ has one equivalence class, [–▫], which becomes the state of the monoidal automaton. The construction above then gives the obvious transition functions required for each generator.

8 Conclusion and future work

The most immediate open question is to determine necessary and sufficient conditions for determinizability: causal closure is a promising candidate. Furthermore we would like to understand the relation between convexity and Theorem 59. Classical topics in the theory of regular languages such as a Myhill-Nerode theorem are also ripe for future investigation. We also plan to investigate further applications of regular monoidal languages in computer science, for example representing trace languages and look-ahead parsing.

Just as our definition of regular monoidal grammar was obtained from Walters' definition of regular grammar by replacing the adjunction $\mathsf{Cat} \to \mathsf{Graph}$ with the adjunction $\mathsf{Pro} \to \mathsf{MonGraph}$, we might consider other adjunctions and their corresponding notion of grammar. In the first instance, our theory should smoothly generalize to languages in free props, but perhaps also other (higher) categorical structures.

We plan to investigate a notion of context-free monoidal language, using a similar algebraic approach to this paper. One candidate for the algebra of such languages, inspired again by [17], are (monoidal) multicategories of n-hole contexts (in the sense of Definition 34).

References

1 Mikołaj Bojańczyk, Bartek Klin, and Julian Salamanca. Monadic monadic second order logic, 2022. doi:10.48550/ARXIV.2201.09969.

2 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.

3 J.R.B. Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1):223–259, 2002. doi:10.1016/S0304-3975(00)00382-0.

4 Robin Cockett and Stephen Lack. Restriction categories III: colimits, partial limits and extensivity. *Mathematical Structures in Computer Science*, 17(4):775–817, 2007. doi:10.1017/S0960129507006056.

5 Thomas Colcombet and Daniela Petrişan. Automata Minimization: a Functorial Approach. *Logical Methods in Computer Science*, Volume 16, Issue 1, March 2020. doi:10.23638/LMCS-16(1:32)2020.

6 Matthew Earnshaw and Paweł Sobociński. Regular monoidal languages, 2022. doi:10.48550/ARXIV.2207.00526.

7 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Languages of higher-dimensional automata. *Mathematical Structures in Computer Science*, 31(5):575–613, 2021. doi:10.1017/S0960129521000293.

8 Richard Garner and Tom Hirschowitz. Shapely monads and analytic functors. *Journal of Logic and Computation*, 28(1):33–83, November 2017. doi:10.1093/logcom/exx029.

9 Ferenc Gécseg and Magnus Steinby. Tree automata, 2015. doi:10.48550/ARXIV.1509.06233.

10 T. Heindel. A Myhill-Nerode theorem beyond trees and forests via finite syntactic categories internal to monoids. *Preprint*, 2017.

11 John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5), 1997. doi:10.1017/S0960129597002375.

12 Kimmo I. Rosenthal. Quantaloids, enriched categories and automata theory. *Applied Categorical Structures*, 3(3):279–301, 1995. doi:10.1007/bf00878445.

13 Paul W. K Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLOS Biology*, 2(12), December 2004. doi:10.1371/journal.pbio.0020424.

14 Jacques Sakarovitch. *Elements of automata theory.* Cambridge University Press, Cambridge New York, 2009.

15 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.

16 Henning Urbat, Jiri Adámek, Liang-Ting Chen, and Stefan Milius. Eilenberg Theorems for Free. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *LIPIcs*, pages 43:1–43:15, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.MFCS.2017.43.

17 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.

18 Vladimir Zamdzhiev. *Rewriting Context-free Families of String Diagrams.* PhD thesis, University of Oxford, 2016.

Oracle with $\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$, but No Many-One Completeness in UP, DisjNP, and DisjCoNP

Anton Ehrmanntraut
Julius-Maximilians-Universität Würzburg, Germany

Fabian Egidy
Julius-Maximilians-Universität Würzburg, Germany

Christian Glaßer
Julius-Maximilians-Universität Würzburg, Germany

Abstract

We construct an oracle relative to which $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$, but there are no many-one complete sets in UP, no many-one complete disjoint NP-pairs, and no many-one complete disjoint coNP-pairs.

This contributes to a research program initiated by Pudlák [33], which studies incompleteness in the finite domain and which mentions the construction of such oracles as open problem. The oracle shows that $\mathsf{NP} \cap \mathsf{coNP}$ is indispensable in the list of hypotheses studied by Pudlák. Hence one should consider stronger hypotheses, in order to find a universal one.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Oracles and decision trees; Theory of computation → Proof complexity

Keywords and phrases computational complexity, promise classes, proof complexity, complete sets, oracle construction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.45

Related Version *Full Version*: https://arxiv.org/abs/2203.11079

1 Introduction

Questions of the existence of complete sets in promise classes have a long history. They turned out to be difficult and remained open. Consider the following examples, where the questions are expressed as hypotheses.

$\mathsf{NP} \cap \mathsf{coNP}$: $\mathrm{NP} \cap \mathrm{coNP}$ does not contain many-one complete sets [23]
UP : UP does not contain many-one complete sets [22]
CON : p-optimal proof systems for TAUT do not exist [26]
SAT : p-optimal proof systems for SAT do not exist [15]
TFNP : TFNP does not contain many-one complete problems [27]
DisjNP : DisjNP does not contain many-one complete pairs [34]
$\mathsf{DisjCoNP}$: DisjCoNP does not contain many-one complete pairs [28, 32]

So far, the following implications are known: $\mathsf{DisjNP} \Rightarrow \mathsf{CON}$ [34], $\mathsf{UP} \Rightarrow \mathsf{CON}$ [25], $\mathsf{DisjCoNP} \Rightarrow \mathsf{TFNP}$ [33], $\mathsf{TFNP} \Rightarrow \mathsf{SAT}$ [5, 33], and $\mathsf{NP} \cap \mathsf{coNP} \Rightarrow \mathsf{CON} \lor \mathsf{SAT}$ [25]. This raises the question of whether further implications are provable with the currently available means. Thanks to a work by Pudlák [33], this question recently gained momentum. In fact, Pudlák's interest goes beyond: He initiated a research program to find a general principle from which the remaining hypotheses follow as special cases. This is motivated by the study of incompleteness in the finite domain, since these hypotheses can either be expressed as the non-existence of complete elements in promise classes or as statements about the unprovability of sentences of some specific form in weak theories.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 45; pp. 45:1–45:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Pudlák [33] states as open problem to construct oracles that show that the relativized conjectures are different or show that they are equivalent. Such oracles have been constructed by Verbitskii [38], Glaßer et al. [18], Khaniki [24], Dose [10, 11, 9], and Dose and Glaßer [12]. The restriction to relativizable proofs arises from the following idea: We consider the mentioned hypotheses as conjectures, hence we expect that they are equivalent. In this situation we are not primarily concerned with the question of whether two hypotheses are equivalent, but rather whether their equivalence can be *recognized* with the currently available means. An accepted formalization of this is the notion of relativizable proofs.

Our Contribution. We construct an oracle relative to which the following holds: UP, DisjNP, $\mathsf{DisjCoNP}$, but $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$, which implies $\neg\mathsf{NP} \cap \mathsf{coNP}$. Hence there is no relativizable proof for $\mathsf{NP}\cap\mathsf{coNP}$, even if we simultaneously assume all remaining hypotheses we mentioned so far. This demonstrates that $\mathsf{NP} \cap \mathsf{coNP}$ is indispensable in the list of currently viewed hypotheses and suggests to broaden the focus and include stronger statements.

Pudlák [33] ranks $\mathsf{NP} \cap \mathsf{coNP}$ as a plausible conjecture that is apparently incomparable with CON and TFNP. Our oracle supports this estimation, as it rules out relativizable proofs for "$\mathsf{CON} \Rightarrow \mathsf{NP} \cap \mathsf{coNP}$" and "$\mathsf{TFNP} \Rightarrow \mathsf{NP} \cap \mathsf{coNP}$." By Dose [10, 11], the same holds for the converse implications. Overall, we recognize a strong independence between $\mathsf{NP} \cap \mathsf{coNP}$ and all remaining hypotheses:

(i) There does not exist a relativizable proof for $\mathsf{NP} \cap \mathsf{coNP}$, even if we simultaneously assume all remaining hypotheses.

(ii) There exists a relativizable proof for the implication $\mathsf{NP} \cap \mathsf{coNP} \Rightarrow \mathsf{CON} \vee \mathsf{SAT}$ [25]. But there does not exist a relativizable proof showing that $\mathsf{NP} \cap \mathsf{coNP}$ implies one of the remaining hypotheses [10, 11].

Our oracle combines several separations with the collapse $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$. This leads to conclusions on the independence of the statement $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ from typical assumptions. For instance, the oracle shows that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ cannot be proved by relativizing means, even under the strong but likely assumption $\mathsf{UP} \wedge \mathsf{DisjNP} \wedge \mathsf{DisjCoNP}$.

Further characteristics of our oracle are, for example, $\mathrm{NE} \neq \mathrm{coNE}$, $\mathrm{NPMV} \not\subseteq_c \mathrm{NPSV}$, and the shrinking and separation properties do not hold for NP and coNP. Corollary 10 presents a list of additional properties.

Open Questions. Currently, for almost every pair A, B of hypotheses, we either know a relativizable proof for the implication $\mathsf{A} \Rightarrow \mathsf{B}$, or we know an oracle relative to which $\mathsf{A} \wedge \neg\mathsf{B}$. Only three cases are left: (1) $\mathsf{UP} \overset{?}{\Rightarrow} \mathsf{DisjNP}$, (2) $\mathsf{TFNP} \overset{?}{\Rightarrow} \mathsf{DisjCoNP}$, and (3) $\mathsf{SAT} \overset{?}{\Rightarrow} \mathsf{TFNP}$. This leads to the following task for future research: Prove these implications or construct oracles relative to which they do not hold.

Background on Connections Between Promise Classes and Proof Systems. Informally, promise classes are complexity classes that are characterized by machines that satisfy certain properties. Usually, these properties are hard or even impossible to validate. Thus, when working with an element of a promise class, one has to trust the *promise* that the respective machine has said property. We are mainly interested in the following well-studied promise classes: The class of disjoint NP-pairs $\mathrm{DisjNP} = \{(A, B) \mid A, B \in \mathrm{NP}, A \cap B = \emptyset\}$ [35, 21], the class of disjoint coNP-pairs DisjCoNP [14, 15] (defined respectively), the class of sets accepted by nondeterministic polynomial-time machines with at most one accepting computation path UP [37], the class $\mathrm{NP} \cap \mathrm{coNP}$ [13], and the class of all total polynomial search problems TFNP [27]. As an example, the machines characterizing UP promise that on every input,

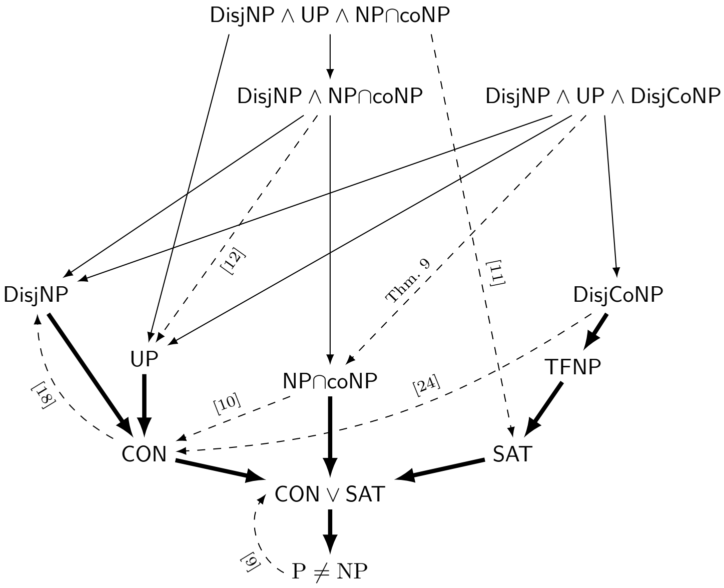


Figure 1 Solid arrows mean implications. All implications occurring in the figure have relativizable proofs. (The only nontrivial ones are DisjNP ⇒ CON [34], UP ⇒ CON [25, Cor. 4.1], DisjCoNP ⇒ TFNP ⇒ SAT [33, Prop. 5.6][5, Thm. 25][33, Prop. 5.10].) Implications between the conjectures originally considered by Pudlák (i.e., not the conjunctions) are highlighted bold. A dashed arrow from one conjecture A to another conjecture B means that there is an oracle X against the implication A ⇒ B, meaning that A ∧ ¬B holds relative to X.

they either reject on all computation paths, or accept on exactly one computation path. Furthermore, we are interested in proof systems defined by Cook and Reckhow [8], especially optimal and p-optimal proof systems for the set of satisfiable formulas SAT, for the set of tautologies TAUT.

The connections between propositional proof systems and promise classes have been studied intensively. Krajícek and Pudlák [26] linked propositional proof systems (and thus the hypothesis CON) to standard complexity classes by proving that NE = coNE implies the existence of optimal propositional proof systems and E = NE implies the existence of p-optimal propositional proof systems. These results were subsequently improved by Köbler, Messner, and Torán [25].

Glaßer, Selman, and Sengupta [17] give several characterizations of DisjNP. Some characterizations use different notions of reducibility while others use the existence of $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete functions in NPSV and the uniform enumerability of disjoint NP-pairs. Glaßer, Selman, and Zhang [19, 20] connect propositional proof systems to disjoint NP-pairs. They prove that the degree structure of DisjNP and of all canonical disjoint pairs of propositional proof systems is the same. Beyersdorff [1, 2, 3, 4] and Beyersdorff and Sadowkski [6] investigate further connections between disjoint NP-pairs and propositional proof systems.

Pudlák [30, 31, 33] draws connections between the finite consistency problem, proof systems, and promise classes like DisjNP and TFNP. Moreover, he asks for oracles that separate hypotheses regarding proof systems and promise classes. Several oracles have been constructed since Pudlák formulated his research questions. Concerning the listed hypotheses, Figure 1 summarizes all known (relativizing) implications and implications that do not hold relative to some oracle.

The paper is organized as follows: Section 2 defines the complexity classes mentioned above and presents our notations. Section 3 contains the oracle construction: the first part defines the construction, the second part proves that it is well-defined, and the last part shows the claimed properties.

2 Preliminaries

Basic Notation. Throughout this paper, let Σ be the alphabet $\{0,1\}$. The set Σ^* denotes the set of finite words over Σ. The set Σ^ω denotes the set of ω-infinite words, i.e., the ω-infinite sequences of characters from Σ. Let $\Sigma^{\leq n} := \{w \in \Sigma^* \mid |w| \leq n\}$. For a word $w \in \Sigma^* \cup \Sigma^\omega$, we denote with $w(i)$ the i-th character of w for $0 \leq i < |w| \leq \omega$. We write $v \sqsubseteq w$ when v is a prefix of w, that is, $|v| \leq |w|$ and $v(i) = w(i)$ for all $0 \leq i < |v|$. Accordingly, $v \sqsubsetneq w$ when $v \sqsubseteq w$ and $v \neq w$. The empty word is denoted by ε. For a finite set $A \subseteq \Sigma^*$, we define $\ell(A) := \sum_{w \in A} |w|$.

Let $\mathbb{N}$ denote the set of non-negative integers, and $\mathbb{N}^+$ the set of positive integers. We say that two sets X and Y *agree on set* Z when $X \cap Z = Y \cap Z$.

The finite words Σ^* can be linearly ordered by their quasi-lexicographic (i.e., "shortlex") order $\prec_{\text{lex}}$, uniquely defined by requiring $0 \prec_{\text{lex}} 1$. Under this definition, there is a unique order-isomorphism between $(\Sigma^*, \prec_{\text{lex}})$ and $(\mathbb{N}, <)$, which induces a polynomial-time computable, polynomial-time invertible bijection between Σ^* and $\mathbb{N}$. Hence, we can transfer the notations, relations, and operations for Σ^* to $\mathbb{N}$ and vice versa. In particular, $|n|$ denotes the length of the word represented by $n \in \mathbb{N}$. By definition of $\prec_{\text{lex}}$, whenever $a \leq b$, then $|a| \leq |b|$. We eliminate the ambiguity of the expressions 0^i and 1^i by always interpreting them over Σ^i. Moreover, $<$ denotes both the less-than relation for natural numbers and the quasi-lexicographic order $\prec_{\text{lex}}$ for finite words. Similarly for $\leq$ and $\preceq_{\text{lex}}$. From the properties of order-isomorphism, this is compatible with the above identification of words and numbers.

Complexity Classes. We understand P (resp., NP) as the usual complexity class of languages decidable by a deterministic (resp., nondeterministic) polynomial-time Turing machine. The class FP refers to the class of total functions that can be computed by a deterministic polynomial-time Turing transducer [29]. Valiant [37] defined UP as the set of all languages that can be recognized by a nondeterministic polynomial-time machine that, on every input, accepts on at most one computation path. For a complexity class $\mathcal{C}$ we define $\text{co}\mathcal{C} := \{\overline{A} \mid A \in \mathcal{C}\}$ as the complementary complexity class of $\mathcal{C}$. Between sets of words, we employ the usual *polynomial-time many-one reducibility*: $A \leq_{\text{m}}^{\text{p}} B$ if there exists an $f \in \text{FP}$ such that $x \in A \Leftrightarrow f(x) \in B$. The usual notion of $\leq_{\text{m}}^{\text{p}}$-completeness and -hardness follows.

A *disjoint* NP*-pair* is a pair (A, B) of disjoint sets in NP. Selman [35] and Grollmann and Selman [21] defined the class DisjNP as the set of disjoint NP-pairs. The classes DisjCoNP [14, 15], DisjUP, and DisjCoUP are defined similarly. Between two pairs, we employ the following related notion of reducibility [34, 18]: Let (A, B) and (C, D) be two disjoint pairs. We say that (A, B) is *polynomial-time many-one reducible to* (C, D), denoted by $(A, B) \leq_{\text{m}}^{\text{pp}} (C, D)$, if there is a function $h \in \text{FP}$ such that $h(A) \subseteq C$ and $h(B) \subseteq D$. The terms $\leq_{\text{m}}^{\text{pp}}$-completeness and -hardness also follow directly from this definition of reduction.

Proof Systems. We use the notion of proof systems for sets by Cook and Reckhow [8]: A function $f \in \text{FP}$ is called a *proof system for* $\text{img}(f)$. Specifically, a proof system f for TAUT is a *propositional* proof system. We say that a proof system g is *(p-)simulated* by a proof system f, denoted by $f \leq g$ (resp., $f \leq^{\text{p}} g$), if there exists a total function π (resp.,

$\pi \in \mathrm{FP}$) and a polynomial p such that $|\pi(x)| \leq p(|x|)$ and $f(\pi(x)) = g(x)$ for all x. We call a proof system f *(p-)optimal* for the set $\mathrm{img}(f)$, if $g \leq f$ (resp., $g \leq^{\mathrm{p}} f$) for all $g \in \mathrm{FP}$ with $\mathrm{img}(g) = \mathrm{img}(f)$.

Relativizations. We can relativize each complexity and function class to some oracle O, by equipping all machines corresponding to the respective class with oracle access to O. That is, e.g., $\mathrm{UP}^O := \{L(M^O) \mid M$ is a nondeterministic polynomial-time oracle Turing machine, and for all inputs x, $M^O(x)$ accepts on at most one path$\}$. The classes P^O, NP^O and so on are defined similarly. We can also relativize our notions of reducibility by using functions from FP^O instead of FP. In other words, we allow the reduction functions to access the oracle in relativized instances. This results in polynomial-time many-one reducibilities relative to an oracle O, which we denote as $\leq_{\mathrm{m}}^{\mathrm{p},O}$ for sets and $\leq_{\mathrm{m}}^{\mathrm{pp},O}$ for pairs of disjoint sets. In the same way, we can relativize (p-)simulation of proof systems to some oracle O, and denote the relativized simulation as $\leq^O$ resp. $\leq^{\mathrm{p},O}$. When it is clear from context that some statements refer to the relativized ones relative to some fixed oracle O, we sometimes omit the indication of O in the superscripts.

We define $p_i(n) := n^i + i$. Let $\{M_i\}_{i\in\mathbb{N}}$ and $\{F_i\}_{i\in\mathbb{N}}$ be, respectively, standard enumerations of nondeterministic polynomial-time (oracle) Turing machines resp. deterministic polynomial-time (oracle) Turing transducers, having the property that runtime of M_i, F_i is bounded by p_i relative to any oracle. Note that $\{L(M_i^O) \mid i \in \mathbb{N}\} = \mathrm{NP}^O$, $\{F_i^O \mid i \in \mathbb{N}\} = \mathrm{FP}^O$.

Specific Notation Used in our Oracle Construction. We now take on the notations proposed by Dose and Glaßer [12] designed for the construction of oracles. The domain of definition, image, and support for partial function $t\colon A \to \mathbb{N}$ are defined as $\mathrm{dom}(t) := \{x \in A \mid t(x)$ defined$\}$, $\mathrm{img}(t) := \{t(x) \mid x \in A, t(x)$ defined$\}$, $\mathrm{supp}(t) := \{x \in A \mid t(x)$ defined and $t(x) > 0\}$. We say that t is *injective on its support* if, for any $a, b \in \mathrm{supp}(t)$, $t(a) = t(b)$ implies $a = b$. If t is not defined at point x, then $t \cup \{x \mapsto y\}$ denotes the extension t' of t that at x has value y and satisfies $\mathrm{dom}(t') = \mathrm{dom}(t) \cup \{x\}$.

For a set A, we denote with $A(x)$ the characteristic function at point x, i.e., $A(x)$ is 1 if $x \in A$, and 0 otherwise. We can identify an oracle $A \subseteq \mathbb{N}$ with its characteristic ω-word $A(0)A(1)A(2)\cdots$ over Σ^ω. In this way, $A(i)$ denotes both the characteristic function at point i and the i-th character of its characteristic word. Similarly, for a finite word $w \in \Sigma^*$, we also understand w as the set $\{i \mid w(i) = 1\}$ and, e.g., we write $A = w \cup B$ where A and B are sets. (However, we understand $|w|$ as the length of the word w, and not the cardinality of set $\{i \mid w(i) = 1\}$.) Thus, a finite word w describes an oracle which is partially defined, i.e., only defined for natural numbers (or equivalently words) $x < |w|$. Being able to interpret a word w as a set and partial oracle is very useful for the oracle construction. In most construction steps we decide the membership of the smallest undefined word of a partial oracle w, which is simply $|w|$. This gives access to very concise notation.

In particular, for oracle machines M, the notation $M^w(x)$ refers to $M^{\{i \mid w(i)=1\}}(x)$ (that is, oracle queries that w is not defined for are negatively answered). This also allows us to define the following notion: we say that $M^w(x)$ is *definite* if all queries on all computation paths are $< |w|$ (or equivalently: $w(q)$ is defined for all queries q on all computation paths); we say that $M^w(x)$ *definitely accepts* (resp., *definitely rejects*) if $M^w(x)$ is definite and accepts (resp., rejects). Intuitively, the term definite describes computations that do not change when extending the respective oracle, because the queries are too short. This allows the following observation:

▶ **Observation 1.**

(i) *When $M^w(x)$ is a definite computation, and $v \sqsupseteq w$, then $M^v(x)$ is definite. Computation $M^v(x)$ accepts if and only if $M^w(x)$ accepts.*

(ii) *When w is defined for all words of length $p_i(|x|)$, then $M_i^w(x)$ is definite.*

(iii) *When $M^w(x)$ accepts on some computation path with set of oracle queries Q, and w, v agree on Q, then $M^v(x)$ accepts on the same computation path and with the same set of oracle queries Q.*

For an oracle w, a transducer F, and a machine M, we occasionally understand the notation $M^w(F^w(x))$ as the single computation of the machine $M \circ F$ on input x relative to w. Consequently, we say that $M^w(F^w(x))$ definitely accepts (resp., rejects) when $M \circ F$ definitely accepts (resp., rejects) input x relative to w.

In our oracle construction, we want to injectively reserve and assign countably infinitely many *levels* n, that are, words of same length n, for a countably infinite family of witness languages, with increasingly large gaps. For this, let $e(0) := 2$, $e(i) := 2^{e(i-1)}$. There is a polynomial-time computable, polynomial-time invertible injective function f, mapping $(m, h) \in \mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. Now define $H_m := \{e(f(m, h)) \mid h \in \mathbb{N}\}$ as the set of levels reserved for witness language m. This definition ensures

▶ **Observation 2.**

(i) *The set H_m is countably infinite, a subset of the even numbers, and all $H_0, H_1, \ldots$ are pairwise disjoint.*

(ii) *The sequence $\min H_0, \min H_1, \ldots$ is unbounded.*

(iii) *When $n \in H_m$, then $n < n' < 2^n$ implies $n' \notin H_0, H_1, \ldots$.*

(iv) *Every set $H_m \in \mathrm{P}$ for all $m \in \mathbb{N}$.*

3 Oracle Construction

We are primarily interested in an oracle O with the property that relative to that oracle, UP, DisjNP, DisjCoNP, and ¬NP ∩ coNP hold, but our construction yields the following slightly stronger statements:

(i) NP ∩ coNP = P (implying ¬NP ∩ coNP).

(ii) DisjNP does not contain $\leq_{\mathrm{m}}^{\mathrm{pp}}$-hard pairs for DisjUP (implying DisjNP).

(iii) UP does not contain $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete languages (i.e., UP).

(iv) DisjCoNP does not contain $\leq_{\mathrm{m}}^{\mathrm{pp}}$-hard pairs for DisjCoUP (implying DisjCoNP).

Given a (possibly partial) oracle O and $m \in \mathbb{N}$, we define the following witness languages:

$$A_m^O := \{0^n \mid n \in H_m, \text{there exists } x \in \Sigma^n \text{ such that } x \in O \text{ and } x \text{ ends with } 0\}$$
$$B_m^O := \{0^n \mid n \in H_m, \text{there exists } x \in \Sigma^n \text{ such that } x \in O \text{ and } x \text{ ends with } 1\}$$
$$C_m^O := \{0^n \mid n \in H_m, \text{there exists } x \in \Sigma^n \text{ such that } x \in O\}$$
$$D_m^O := \{0^n \mid n \in H_m, \text{for all } x \in \Sigma^n, x \in O \rightarrow x \text{ ends with } 0\}$$
$$E_m^O := \{0^n \mid n \in H_m, \text{for all } x \in \Sigma^n, x \in O \rightarrow x \text{ ends with } 1\}$$

Their purpose is to be a “witness” that an element of DisjNP (resp., UP, DisjCoNP) is not complete by admitting no reduction to this element. This only works if the witness languages themselves belong to the respective classes. The following observation shows how the membership of the witness languages to the respective classes depends on the oracle.

▶ **Observation 3.**

(i) *If for all $n \in H_m$, $|O \cap \Sigma^n| \leq 1$, then (A_m^O, B_m^O) is in* DisjUP^O*, and C_m^O is in* UP^O.

(ii) *If for all $n \in H_m$, $O \cap \Sigma^n$ contains at least one word but not two words with the same parity, (i.e., there exists $\alpha \in \Sigma^{n-1}0$, $\beta \in \Sigma^{n-1}1$ such that the set $O \cap \Sigma^n$ is equal to $\{\alpha\}$ or $\{\beta\}$ or $\{\alpha, \beta\}$), then (D_m^O, E_m^O) is in* $\mathrm{DisjCoUP}^O$.

Preview of the Construction. The construction is quite technical, since the oracle has to satisfy several properties which simultaneously demand structure (i.e., property (i)) and freedom (i.e., properties (ii), (iii) and (iv)) during the construction. This leads to several dependencies and special cases that need to be addressed, mostly by combinatorial arguments and various extensions of the oracle constructed so far. To keep track of the progress of the construction, it is divided into tasks corresponding to the desired properties (i)–(iv). Each task contributes to the goal of satisfying its corresponding property.

1. Work towards $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$: For all $a \neq b$, the construction tries to achieve that M_a, M_b do not accept complementary. (*Accepting complementary* should mean that for each input x, precisely one of $M_a(x), M_b(x)$ accepts and the other rejects.) If this is not possible, (M_a, M_b) inherently accept complementary, and thus $L(M_a) \in \mathrm{NP} \cap \mathrm{coNP}$. Then, we start to *encode* into the oracle, whether M_a accepts some inputs or not. Thus, the final oracle will contain the encodings for almost all inputs, thus allowing to recover the accepting behavior of M_a and hence to decide $L(M_i)$ in P using oracle queries.
2. Work towards (ii), which implies DisjNP: For all $i \neq j$, the construction tries to achieve that M_i, M_j both accept some input x, hence $x \in L(M_i) \cap L(M_j)$ and $(L(M_i), L(M_j)) \notin \mathrm{DisjNP}$. If this is not possible, (M_i, M_j) inherently is a disjoint NP-pair. In this case, we fix some m, make sure that (A_m, B_m) is a disjoint UP-pair and diagonalize against every transducer F_r, so that F_r does not realize the reduction $(A_m, B_m) \leq_{\mathrm{m}}^{\mathrm{pp}} (L(M_i), L(M_j))$. This is achieved by, (i) for all $n \in H_m$, insert at most one word of length n into O (and thus $(A_m, B_m) \in \mathrm{DisjUP}$), and (ii) for every r there is an $n \in H_m$ such that $0^n \in A_m$ but $M_i(F_r(0^n))$ rejects (or analogously $0^n \in B_m$ but $M_j(F_r(0^n))$ rejects).
3. Work towards (iii), i.e., UP: Try to make M_i accept on two separate paths. If this is not possible, then $L(M_i)$ inherently is a UP-language. In this case, we fix some m, make sure that C_m is a language in UP and diagonalize against every transducer F_r so that F_r does not realize the reduction $C_m \leq_{\mathrm{m}}^{\mathrm{p}} L(M_i)$. This is achieved by, (i) for all $n \in H_m$, insert at most one word of length n into O (and thus $C_m \in \mathrm{UP}$), and (ii) for every r there is an $n \in H_m$ such that $0^n \in C_m$ if and only if $M_i(F_r(0^n))$ rejects.
4. Work towards (iv), which implies DisjCoNP: Try to achieve that M_i, M_j both reject some input x, hence $x \in \overline{L(M_i)} \cap \overline{L(M_j)}$ and $(L(M_i), L(M_j)) \notin \mathrm{DisjNP}$. If this is not possible, (M_i, M_j) inherently is a disjoint coNP-pair. In this case, we fix some m, make sure that (D_m, E_m) is a disjoint coUP-pair and diagonalize against every transducer F_r, so that F_r does not realize the reduction $(D_m, E_m) \leq_{\mathrm{m}}^{\mathrm{pp}} (L(M_i), L(M_j))$. This is achieved by, (i) for all $n \in H_m$, insert at least one word of length n into O but not two words with same parity (and thus $(D_m, E_m) \in \mathrm{DisjCoUP}$), and (ii) for every r there is an $n \in H_m$ such that $0^n \in D_m$ but $M_i(F_r(0^n))$ accepts (or analogously $0^n \in E_m$ but $M_j(F_r(0^n))$ accepts).

To these requirements, we assign the following symbols representing tasks: $\tau_{a,b}^1$, $\tau_{i,j}^2$, $\tau_{i,j,r}^2$, τ_i^3, $\tau_{i,r}^3$, $\tau_{i,j}^4$, $\tau_{i,j,r}^4$ for all $a, b, i, j, r \in \mathbb{N}, i \neq j$, $a \neq b$. The symbol $\tau_{a,b}^1$ represents the coding or the destruction of $\mathrm{NP} \cap \mathrm{coNP}$-pairs. The symbol $\tau_{i,j}^2$ represents the destruction of a disjoint NP-pair, $\tau_{i,j,r}^2$ the diagonalization of that pair against transducer F_r. Analogously for UP and $\tau_i^3, \tau_{i,r}^3$. Analogously for DisjCoNP and $\tau_{i,j}^4, \tau_{i,j,r}^4$.

For the coding, we injectively define the code word $c(a,b,x) := 0^a10^b10^l10^p1x$ with $p = p_a(|x|) + p_b(|x|)$, $l \in \mathbb{N}$ minimal such that $l \geq 7/8|c(a,b,x)|$ and $c(a,b,x)$ has odd length. By this, a code word contains the word x as information and is padded to sufficient length. We call any word of the form $c(\cdot,\cdot,\cdot)$ a *code word.* This ensures the following properties:

▷ Claim 4. For all $a, b \in \mathbb{N}$, $x \in \Sigma^*$, the following holds:

(i) $|c(a,b,x)| \notin H_m$ for any m.

(ii) For fixed a, b, the function $x \mapsto c(a,b,x)$ is polynomial-time computable, and polynomial-time invertible with respect to $|x|$.

(iii) Relative to any oracle, the running times of $M_a(x)$ and $M_b(x)$ are both bounded by $< |c(a,b,x)|/8$.

(iv) For every partial oracle $w \in \Sigma^*$, if $c(a,b,x) \leq |w|$, then $M_a^w(x)$ and $M_b^w(x)$ are definite.

During the construction we successively add requirements that we maintain. To keep track of these requirements, we use a partial function t belonging to some set $\mathcal{T}$, which we define below. In particular, the partial function t maps some of the above task symbols to $\mathbb{N}$. In fact, these requirements determine to a large extent how tasks are treated and are mainly responsible that the oracle satisfies the desired properties. To add a requirement in the construction, we can extend the function t.

Define $\mathcal{T}$ as the set of all partial functions t mapping $\tau^1_{a,b}$, $\tau^2_{i,j}$, τ^3_i, $\tau^4_{i,j}$, $i \neq j$, $a \neq b$ to $\mathbb{N}$, and $\mathrm{dom}(t)$ is finite, and t is injective on its support.

To now link the maintenance of the requirements with the oracle construction, we introduce the notion of *validity.* A partial oracle $w \in \Sigma^*$ is called *t-valid* for $t \in \mathcal{T}$ if it satisfies the following properties:

V1 If $t(\tau^1_{a,b}) = 0$, then there exists an x such that $M_a^w(x)$, $M_b^w(x)$ both definitely accept or both definitely reject.
(Meaning: if $t(\tau^1_{a,b}) = 0$, then for every extension of the oracle, M_a, M_b do not accept complementary.)

V2 If $0 < t(\tau^1_{a,b}) \leq c(a,b,x) < |w|$, then $M_a^w(x)$ is definite. Additionally, the computation $M_a^w(x)$ accepts when $c(a,b,x) \in w$, and rejects when $c(a,b,x) \notin w$. When the above conditions are met by $c(a,b,x)$ we sometimes refer to these code words as *mandatory* code words with respect to some t-valid partial oracle w. Note that when the previous conditions are not met ($\tau^1_{a,b} \notin \mathrm{dom}(t)$ or $t(\tau^1_{a,b}) = 0$ or $t(\tau^1_{a,b}) > c(a,b,x)$) then the code word $c(a,b,x)$ may be a member of oracle w, independent of M_a, M_b.
(Meaning: if $t(\tau^1_{a,b}) > 0$, then from $t(\tau^1_{a,b})$ on, we encode $L(M_a)$ into the oracle. That is, $L(M_a^O) = (\{x \mid c(a,b,x) \in O\} \cup \text{some finite set}) \in \mathrm{P}^O$.)

V3 If $t(\tau^2_{i,j}) = 0$, then there exists an x such that $M_i^w(x)$, $M_j^w(x)$ both definitely accept.
(Meaning: if $t(\tau^2_{i,j}) = 0$, then for every extension of the oracle, $(L(M_i), L(M_j)) \notin \mathrm{DisjNP}$.)

V4 If $t(\tau^2_{i,j}) = m > 0$, then for every $n \in H_m$ it holds that $|\Sigma^n \cap w| \leq 1$.
(Meaning: if $t(\tau^2_{i,j}) = m > 0$, then ensure that $(A_m, B_m) \in \mathrm{DisjUP}$ relative to the final oracle.)

V5 If $t(\tau^3_i) = 0$, then there exists an x such that $M_i^w(x)$ is definite and accepts on two different paths.
(Meaning: if $t(\tau^3_i) = 0$, then for every extension of the oracle, $L(M_i) \notin \mathrm{UP}$.)

V6 If $t(\tau^3_i) = m > 0$, then for every $n \in H_m$ it holds that $|\Sigma^n \cap w| \leq 1$.
(Meaning: if $t(\tau^3_i) = m > 0$, then ensure that $C_m \in \mathrm{UP}$ relative to the final oracle.)

V7 If $t(\tau^4_{i,j}) = 0$, then there exists an x such that $M_i^w(x)$, $M_j^w(x)$ both definitely reject.
(Meaning: if $t(\tau^4_{i,j}) = 0$, then for every extension of the oracle, $(\overline{L(M_i)}, \overline{L(M_j)}) \notin \mathrm{DisjCoNP}$.)

V8 If $t(\tau^4_{i,j}) = m > 0$, then for every $n \in H_m$ it holds that all words in $\Sigma^n \cap w$ have pairwise different parity. If additionally w is defined for all words of length n, then $|\Sigma^n \cap w| > 0$. (Meaning: if $t(\tau^4_{i,j}) = m > 0$, then ensure that $(D_m, E_m) \in \mathrm{DisjCoUP}$ relative to the final oracle.)

Intuitively, a t-valid oracle is a possibly partial oracle which has the desired properties (i)–(iv) "partially satisfied". This notion of validness helps in the oracle construction, since our oracle is defined inductively, the induction step deals with a partial oracle and therefore t-validity fits great as part of an induction hypothesis which states that the partial oracle is constructed properly so far. Observe that V4, V6, V8 do not (pairwise) contradict each other, since t is injective on its support and all $H_1, H_2, \dots$ are pairwise disjoint, by Observation 2(i). Also observe that V2 and V4 (resp., V2 and V6, V2 and V8) do not contradict each other, as $c(\cdot,\cdot,\cdot)$ has odd length, but all n in all H_m are even by Observation 2(i).

Oracle Construction. Let T be a countable enumeration of

$$\begin{aligned}\{\tau^1_{a,b} \mid a, b \in \mathbb{N}, a \neq b\} \cup \{\tau^2_{i,j} \mid i, j \in \mathbb{N}, i \neq j\} &\cup \{\tau^2_{i,j,r} \mid i, j, r \in \mathbb{N}, i \neq j\}\\ \cup \{\tau^3_i \mid i \in \mathbb{N}\} \qquad\qquad &\cup \{\tau^3_{i,r} \mid i, r \in \mathbb{N}\}\\ \cup \{\tau^4_{i,j} \mid i, j \in \mathbb{N}, i \neq j\} &\cup \{\tau^4_{i,j,r} \mid i, j, r \in \mathbb{N}, i \neq j\}\end{aligned}$$

with the property that $\tau^2_{i,j}$ appears earlier than $\tau^2_{i,j,r}$, τ^3_i appears earlier than $\tau^3_{i,r}$, $\tau^4_{i,j}$ earlier than $\tau^4_{i,j,r}$.

We inductively define an infinite sequence $\{(w_s, t_s)\}_{s \in \mathbb{N}}$, where the s-th term of the sequence is a pair (w_s, t_s) of a partial oracle and a function in $\mathcal{T}$. We call the s-th term the *stage* s. We ensure that for all $s \in \mathbb{N}$, w_s is a t_s-valid partial oracle.

In each stage, we treat the smallest task in the order specified by T, and after treating a task we remove it and possibly other higher tasks from T. In the next stage, we continue with the next task not already removed from T. (In every stage, there always exists a task not already removed, as we never remove *all* remaining tasks from T in any stage.)

We start with the nowhere defined function $t_0 \in \mathcal{T}$ and the t_0-valid oracle $w_0 := \varepsilon$ as 0-th stage. Then we begin treating the tasks.

Thus, for stage $s > 0$, we have that $w_0, w_1, \dots, w_{s-1}$ and $t_0, t_1, \dots, t_{s-1}$ are defined. With this, we define the s-th stage (w_s, t_s) such that (a) $w_{s-1} \sqsubsetneq w_s$, and $t_s \in \mathcal{T}$ is a (not necessarily strict) extension of t_{s-1}, and (b) w_s is t_s-valid, and (c) the earliest task τ still in T is treated and removed in some way.

So for each task we strictly extend the oracle and are allowed to add more requirements, by extending the valid function, that have to be maintained in the further construction. Finally, we choose $O := \bigcup_{i \in \mathbb{N}} w_i$. (Note that O is totally defined since in each step we strictly extend the oracle.) Also, every task in T is assigned some stage s where it was treated (or removed from T).

We now define stage $s > 0$, which starts with some $t_{s-1} \in \mathcal{T}$ and a t_{s-1}-valid oracle w_{s-1} and treats the first task that still is in T choosing an extension $t_s \in \mathcal{T}$ of t_{s-1} and a t_s-valid $w_s \sqsupsetneq w_{s-1}$. Let us recall that each task is immediately deleted from T after it is treated. There are seven cases depending on the form of the task that is treated in stage s:

- Task $\tau^1_{a,b}$: Let $t' := t_{s-1} \cup \{\tau^1_{a,b} \mapsto 0\}$. If there exists a t'-valid $v \sqsupsetneq w_{s-1}$, then assign $t_s := t'$ and let $w_s := v$.

  Otherwise, let $t_s := t_{s-1} \cup \{\tau^1_{a,b} \mapsto n\}$ with $n \in \mathbb{N}^+$ sufficiently large such that $n > |w_s|, \max \mathrm{img}(t_{s-1})$. Thus t_s is injective on its support, and w_{s-1} is t_s-valid. Let $w_s := w_{s-1}y$ with $y \in \{0, 1\}$ such that w_s is t_s-valid. Lemma 5 shows that such y does indeed exist.

(Meaning: try to ensure that M_a, M_b do not accept complementary, cf. V1. If that is impossible, require that from now on the computations of M_a are encoded into the oracle, cf. V2.)

- Task $\tau^2_{i,j}$: Let $t' := t_{s-1} \cup \{\tau^2_{i,j} \mapsto 0\}$. If there exists t'-valid $v \sqsupsetneq w_{s-1}$, then assign $t_s := t'$ and $w_s := v$. Besides task $\tau^2_{i,j}$, also remove all tasks $\tau^2_{i,j,0}, \tau^2_{i,j,1}, \dots$ from T.

  Otherwise, let $t_s := t_{s-1} \cup \{\tau^2_{i,j} \mapsto m\}$ with $m \in \mathbb{N}^+$ sufficiently large such that $m \notin \mathrm{img}(t_{s-1})$ and that w_{s-1} defines no word of length $\min H_m$. Thus t_s is injective on its support, and w_{s-1} is t_s-valid. Let $w_s := w_{s-1}y$ with $y \in \{0, 1\}$ such that w_s is t_s-valid. Again, Lemma 5 shows that such y does indeed exist.

  (Meaning: try to ensure that M_i, M_j do not accept disjointly, cf. V3. If that is impossible, choose a sufficiently large “fresh” m and require for the further construction that $(A_m, B_m) \in$ DisjUP (cf. V4). The treatment of the tasks $\tau^2_{i,j,0}, \tau^2_{i,j,1}, \dots$ defined below makes sure that (A_m, B_m) cannot be reduced to $(L(M_i), L(M_j))$.)
- Task τ^3_i: Defined symmetrically to task $\tau^2_{i,j}$. Let $t' := t_{s-1} \cup \{\tau^3_i \mapsto 0\}$. If there exists t'-valid $v \sqsupsetneq w_{s-1}$, then assign $t_s := t'$ and $w_s := v$. Besides task τ^3_i, also remove all tasks $\tau^3_{i,0}, \tau^3_{i,1}, \dots$ from T.

  Otherwise, let $t_s := t_{s-1} \cup \{\tau^3_i \mapsto m\}$ with $m \in \mathbb{N}^+$ sufficiently large such that $m \notin \mathrm{img}(t_{s-1})$ and that w_{s-1} defines no word of length $\min H_m$. Thus t_s is injective on its support, and w_{s-1} is t_s-valid. Let $w_s := w_{s-1}y$ with $y \in \{0, 1\}$ such that w_s is t_s-valid. Again, Lemma 5 shows that such y does indeed exist.

  (Meaning: try to ensure that M_i does accept on two different paths, cf. V5. If that is impossible, choose a sufficiently large “fresh” m and require for the further construction that $C_m \in$ UP (cf. V6). The treatment of the tasks $\tau^3_{i,0}, \tau^3_{i,1}, \dots$ defined below makes sure that C_m cannot be reduced to $L(M_i)$.)
- Task $\tau^4_{i,j}$: Defined symmetrically to task $\tau^2_{i,j}$. (Meaning: try to ensure that M_i, M_j do not reject disjointly, cf. V7. If that is impossible, choose a sufficiently large “fresh” m and require for the further construction that $(D_m, E_m) \in$ DisjCoUP (cf. V8). The treatment of the tasks $\tau^4_{i,j,0}, \tau^4_{i,j,1}, \dots$ defined below makes sure that (D_m, E_m) cannot be reduced to $(\overline{L(M_i)}, \overline{L(M_j)})$.)
- Task $\tau^2_{i,j,r}$: We have $t_{s-1}(\tau^2_{i,j}) = m \in \mathbb{N}^+$. Let $t_s := t_{s-1}$ and choose a t_s-valid $w_s \sqsupsetneq w_{s-1}$ such that there is some $n \in \mathbb{N}$ and at least one of the following holds:
  - $0^n \in A^v_m$ for all $v \sqsupseteq w_s$ and $M^{w_s}_i(F^{w_s}_r(0^n))$ definitely rejects.
  - $0^n \in B^v_m$ for all $v \sqsupseteq w_s$ and $M^{w_s}_j(F^{w_s}_r(0^n))$ definitely rejects.

  In Theorem 6 we show that such w_s does exist.

  (Meaning: ensure that F_r does not reduce (A_m, B_m) to $(L(M_i), L(M_j))$.)
- Task $\tau^3_{i,r}$: We have $t_{s-1}(\tau^3_i) = m \in \mathbb{N}^+$. Let $t_s := t_{s-1}$ and choose a t_s-valid $w_s \sqsupsetneq w_{s-1}$ such that there is some $n \in \mathbb{N}$ and at least one of the following holds:
  - $0^n \in C^v_m$ for all $v \sqsupseteq w_s$ and $M^{w_s}_i(F^{w_s}_r(0^n))$ definitely rejects.
  - $0^n \notin C^v_m$ for all $v \sqsupseteq w_s$ and $M^{w_s}_i(F^{w_s}_r(0^n))$ definitely accepts.

  In Theorem 7 we show that such w_s does exist.

  (Meaning: ensure that F_r does not reduce C_m to $L(M_i)$.)
- Task $\tau^4_{i,j,r}$: Defined symmetrically to $\tau^2_{i,j,r}$. Choose a t_s-valid $w_s \sqsupsetneq w_{s-1}$ such that for some $n \in \mathbb{N}$, one of the two holds:
  - $0^n \in D^v_m$ for all $v \sqsupseteq w_s$ and $M^{w_s}_i(F^{w_s}_r(0^n))$ definitely accepts.
  - $0^n \in E^v_m$ for all $v \sqsupseteq w_s$ and $M^{w_s}_j(F^{w_s}_r(0^n))$ definitely accepts.

  In Theorem 8 we show that such w_s does exist.

  (Meaning: ensure that F_r does not reduce (D_m, E_m) to $(\overline{L(M_i)}, \overline{L(M_j)})$.)

Observe that t_s is always defined to be in $\mathcal{T}$. Remember that the treated task is immediately deleted from T. This completes the definition of stage s, and thus, the entire sequence $\{(w_s, t_s)\}_{s \in \mathbb{N}}$. We now show that this construction is indeed possible. The proofs of the theorems/lemma announced in the definition are either roughly sketched or omitted. For detailed proofs, we refer to the full version of the paper. It is not difficult to see that a valid oracle can be extended by one bit such that it remains valid:

▶ **Lemma 5.** *Let $s \in \mathbb{N}$, $(w_0, t_0), \dots, (w_s, t_s)$ defined, and let $w \in \Sigma^*$ be a t_s-valid oracle with $w \sqsupseteq w_s$, and $z := |w|$. (Think of z as the next word we need to decide its membership to the oracle, i.e., $z \notin w0$ or $z \in w1$.) Then there exists $y \in \{0, 1\}$ such that wy is t_s-valid. Specifically:*

(i) *If $z = c(a, b, x)$ and $0 < t_s(\tau^1_{a,b}) \leq z$, then $w1$ is t_s-valid if $M^w_a(x)$ accepts (or when $M^w_b(x)$ rejects), and $w0$ is t_s-valid if $M^w_a(x)$ rejects (or when $M^w_b(x)$ accepts).*
(Meaning: if we are at a position of some mandatory code word, add the word as appropriate for the NP ∩ coNP*-pair.)*

(ii) *If there exists $\tau = \tau^2_{i,j}$ or $\tau = \tau^3_i$ with $m = t_s(\tau) > 0$ and $n \in H_m$ such that $|z| = n$, $w \cap \Sigma^n \neq \emptyset$, then $w0$ is t_s-valid.*
(Meaning: if we are on a level n belonging to a DisjUP*-pair or a* UP*-language, ensure that there is no more than one word on that level.)*

(iii) *If there exists $\tau^4_{i,j}$, $m = t_s(\tau^4_{i,j}) > 0$ and $n \in H_m$ such that $|z| = n$ and there is some other word $x \in w \cap \Sigma^n$ with same parity as z, then $w0$ is t_s-valid. (Meaning: if we are on a level n belonging to a* DisjCoUP*-pair, ensure that on that level, there are no two words with the same parity.)*

(iv) *If there exists $\tau^4_{i,j}$, $m = t_s(\tau^4_{i,j}) > 0$ and $n \in H_m$ such that $|z| = n$, $|z + 1| > n$, $w \cap \Sigma^n = \emptyset$, then $w1$ is t_s-valid.*
(Meaning: if we finalize level n belonging to a DisjCoUP *witness pair, ensure that there is at least one word on that level.)*

(v) *In all other cases, $w0$ and $w1$ are t_s-valid.*

Lemma 5 shows that the construction is possible for the tasks $\tau^1_{a,b}$, $\tau^2_{i,j}$, τ^3_i and $\tau^4_{i,j}$. Now we show that the construction is possible for $\tau^2_{i,j,r}$, $\tau^3_{i,r}$ and $\tau^4_{i,j,r}$, respectively. We first consider task $\tau^2_{i,j,r}$.

▶ **Theorem 6.** *Let $s \in \mathbb{N}^+$, $(w_0, t_0), \dots, (w_{s-1}, t_{s-1})$ defined. Consider task $\tau^2_{i,j,r}$.*

Suppose that $t_s = t_{s-1}$, $t_s(\tau^2_{i,j}) = m > 0$. Then there exists a t_s-valid $w \sqsupsetneq w_{s-1}$ and $n \in \mathbb{N}$ such that one of the two holds:

(i) *$0^n \in A^v_m$ for all $v \sqsupseteq w$ and $M^w_i(F^w_r(0^n))$ definitely rejects.*

(ii) *$0^n \in B^v_m$ for all $v \sqsupseteq w$ and $M^w_j(F^w_r(0^n))$ definitely rejects.*

Proof sketch. We prove the Theorem by contradiction. Let $\hat{s} < s$ be the stage that treated $\tau^2_{i,j}$ with $t_{\hat{s}} = t_{\hat{s}-1} \cup \{\tau^2_{i,j} \mapsto m\}$ and $m > 0$. We construct a suitable alternative oracle $u' \sqsupsetneq w_{\hat{s}-1}$, which is valid with respect to $t' := t_{\hat{s}-1} \cup \{\tau^2_{i,j} \mapsto 0\}$. Then, by definition, we obtain that u' is one possible t'-valid extension of $w_{\hat{s}-1}$ in stage $\hat{s}$, hence $t_{\hat{s}} = t'$ and $t_{\hat{s}}(\tau^2_{i,j}) = 0$, contradicting $t_s(\tau^2_{i,j}) = m > 0$ in the hypothesis of this Theorem 6.

Assume that (i) and (ii) do not hold. Fix some sufficiently large $n \in H_m$. This is some level that belongs to the witness NP-pair (A_m, B_m). For every $\xi \in \Sigma^n$, one can provisionally keep extending w_{s-1} bitwise with Lemma 5 while inserting precisely the word ξ into level n. Continue extending until a sufficiently long but fixed length n' such that $M_i(F_r(0^n))$ and $M_j(F_r(0^n))$ are definite (relative to any oracle), and call the resulting oracle u_ξ. By construction, this oracle is t_s-valid. By assumption, when $\alpha \in \Sigma^{n-1}0$, then $M_i(F_r(0^n))$ accepts relative to u_α, and when $\beta \in \Sigma^{n-1}1$, then $M_j(F_r(0^n))$ accepts relative to u_β.

We want to maintain these accepting computations relative to an oracle containing, in level n, exactly one $\alpha \in \Sigma^{n-1}0$ and one $\beta \in \Sigma^{n-1}1$. The accepting behavior for any such computation, say $M_i(F_r(0^n))$ relative to u_α, depends on the oracle queries posed on that path. However, this computation might also query certain mandatory code words $c(a, b, x)$, whose memberships depend on further, shorter queries of computations $M_a(x)$, $M_b(x)$. Continuing this recursively, we obtain a set of queries Q_α^+ the original computation $M_i(F_r(0^n))$ transitively depends on.

Similarly, for each $\beta \in \Sigma^{n-1}1$ we can define a set of queries Q_β^+ an accepting path of the computation $M_j(F_r(0^n))$ relative to u_β depends on. One can verify that Q_α^+ and Q_β^+ only have polynomially many elements.

The crucial idea that completes the proof is to find suitable $\alpha \in \Sigma^{n-1}0$ and $\beta \in \Sigma^{n-1}1$ such that u_α and u_β agree on the set $Q_\alpha^+ \cap Q_\beta^+$. Such a pair of words α and β exists; we skip here the combinatorial argument, but intuitively this assertion holds from the fact that there are exponentially many choices for α, β but for each choice, Q_α^+ and Q_β^+ create only polynomially many dependencies. With this property, it is possible to construct a $t_{\hat{s}-1}$-valid oracle $u' \sqsupsetneq w_{s-1}$ such that $\alpha, \beta \in u'$ holds, u' and u_α agree on Q_α^+, and symmetric u' and u_β agree on Q_β^+. By assumption and Observation 1(iii), this means that *both* $M_i(F_r(0^n))$ and $M_j(F_r(0^n))$ definitely accept relative to u'. Thus u' is also t'-valid, as desired. ◀

With only slight modifications, one can give the same Theorem concerning tasks $\tau_{i,r}^3$. We omit the specific details.

▶ **Theorem 7.** *Let $s \in \mathbb{N}^+$, $(w_0, t_0), \ldots, (w_{s-1}, t_{s-1})$ defined. Consider task $\tau_{i,r}^3$.*

Suppose that $t_s = t_{s-1}$, $t_s(\tau_i^3) = m > 0$. Then there exists a t_s-valid $w \sqsupsetneq w_{s-1}$ and $n \in \mathbb{N}$ such that one of the following holds:

(i) *$0^n \in C_m^v$ for all $v \sqsupseteq w$ and $M_i^w(F_r^w(0^n))$ definitely rejects.*

(ii) *$0^n \notin C_m^v$ for all $v \sqsupseteq w$ and $M_i^w(F_r^w(0^n))$ definitely accepts.*

Lastly, handling task $\tau_{i,j,r}^4$ is also possible. For this we use techniques similar to previous theorems. In this setting, it is in fact possible to explicitly construct a suitable extension for the task. Due to many additional technical details required, we omit the proof of this theorem and refer to the full version of the paper.

▶ **Theorem 8.** *Let $s \in \mathbb{N}^+$, (w_{s-1}, t_{s-1}) defined. Consider task $\tau_{i,j,r}^4$.*

Suppose that $t_s = t_{s-1}$, $t_s(\tau_{i,j}^4) = m > 0$. Then there exists a t_s-valid $w \sqsupsetneq w_{s-1}$ and $n \in \mathbb{N}$ such that one of the following holds:

(i) *$0^n \in D_m^v$ for all $v \sqsupseteq w$ and $M_i^w(F_r^w(0^n))$ definitely accepts.*

(ii) *$0^n \in E_m^v$ for all $v \sqsupseteq w$ and $M_j^w(F_r^w(0^n))$ definitely accepts.*

We have now completed the proofs showing that the oracle construction can be performed as desired. The following theorem confirms the desired properties of $O := \bigcup_{i \in \mathbb{N}} w_i$. Remember that $|w_0| < |w_1| < \ldots$ is unbounded, hence for any z there is a sufficiently large s such that $|w_s| > z$. Also remember that w_s is t_s-valid for all $s \in \mathbb{N}$. Using these facts and the properties V1–V8 of t_s-valid oracles, one can easily state the following result for the final oracle.

▶ **Theorem 9.** *Relative to $O = \bigcup_{i \in \mathbb{N}} w_i$, the following holds:*

(i) $\mathrm{NP} \cap \mathrm{coNP} = \mathrm{P}$*, which implies* $\neg\mathsf{NP} \cap \mathsf{coNP}$.

(ii) *No pair in* DisjNP *is* $\leq_{\mathrm{m}}^{\mathrm{pp}}$*-hard for* DisjUP*, which implies* DisjNP.

(iii) *No language in* UP *is* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete for* UP*, i.e.,* UP.

(iv) *No pair in* DisjCoNP *is* $\leq_{\mathrm{m}}^{\mathrm{pp}}$*-hard for* DisjCoUP*, which implies* $\mathsf{DisjCoNP}$.

From Theorem 9 and known relativizable results, we obtain the following additional properties that hold relative to the oracle. See, e.g., the work of Fenner et al. [15] for a definition of the mentioned function classes $\mathrm{NPSV}, \mathrm{NPbV}, \mathrm{NP}k\mathrm{V}, \mathrm{NPMV}$ and their total variants $\mathrm{NPSV_t}, \mathrm{NPbV_t}, \mathrm{NP}k\mathrm{V_t}, \mathrm{NPMV_t}$. Here, NEE means $\mathrm{NTIME}\left(2^{O(2^n)}\right)$.

▶ **Corollary 10.** *The following holds relative to the oracle O constructed in this section.*

(i) $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP} \subsetneq \mathrm{UP} \subsetneq \mathrm{NP}$
(ii) UP, NP, NE, *and* NEE *are not closed under complement.*
(iii) $\mathrm{UP} \not\subseteq \mathrm{coNP}$
(iv) $\mathrm{NEE} \cap \mathrm{TALLY} \not\subseteq \mathrm{coNEE}$
(v) $\mathrm{NPSV_t} \subseteq \mathrm{PF}$
(vi) $\mathrm{NPbV_t} \not\subseteq_c \mathrm{NPSV_t}$
(vii) $\mathrm{NP}k\mathrm{V_t} \not\subseteq_c \mathrm{NPSV_t}$ *for all $k \geq 2$*
(viii) $\mathrm{NPMV_t} \not\subseteq_c \mathrm{NPSV_t}$
(ix) $\mathrm{NPMV} \not\subseteq_c \mathrm{NPSV}$
(x) $\mathrm{TFNP} \not\subseteq_c \mathrm{PF}$
(xi) $\mathrm{NP} \cap \mathrm{coNP}$ *has $\leq_\mathrm{m}^\mathrm{p}$-complete sets, i.e.,* $\neg\mathsf{NP} \cap \mathsf{coNP}$.
(xii) UP *has no $\leq_\mathrm{m}^\mathrm{p}$-complete sets, i.e.,* UP.
(xiii) DisjNP *has no $\leq_\mathrm{m}^\mathrm{pp}$-complete pairs, i.e.,* DisjNP.
(xiv) DisjCoNP *has no $\leq_\mathrm{m}^\mathrm{pp}$-complete pairs, i.e.,* $\mathsf{DisjCoNP}$.
(xv) *No pair in* DisjNP *is $\leq_\mathrm{m}^\mathrm{pp}$-hard for* DisjUP.
(xvi) *No pair in* DisjCoNP *is $\leq_\mathrm{m}^\mathrm{pp}$-hard for* DisjCoUP.
(xvii) *There are no p-optimal proof systems for* TAUT, *i.e.,* CON.
(xviii) *There are no optimal proof systems for* TAUT.
(xix) *There are no p-optimal proof systems for* SAT, *i.e.,* SAT.
(xx) TFNP *has no $\leq_\mathrm{m}^\mathrm{p}$-complete problems, i.e.,* TFNP.
(xxi) $\mathrm{NPMV_t}$ *has no $\leq_\mathrm{m}^\mathrm{p}$-complete functions.*
(xxii) NP *and* coNP *do not have the shrinking property. [7, 16]*
(xxiii) NP *and* coNP *do not have the separation property. [16]*
(xxiv) DisjNP *and* DisjCoNP *contain* P*-inseparable pairs.*

References

1 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.

2 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.

3 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.

4 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.

5 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.

6 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.

7 A. Blass and Y. Gurevich. Equivalence relations, invariants, and normal forms. *SIAM Journal on Computing*, 13(4):682–689, 1984. doi:10.1137/0213042.

8 S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979. doi:10.2307/2273702.

9 T. Dose. *Balance Problems for Integer Circuits and Separations of Relativized Conjectures on Incompleteness in Promise Classes.* PhD thesis, Fakultät für Mathematik und Informatik, Universität Würzburg, 2020. doi:10.25972/OPUS-22220.

10 T. Dose. Further oracles separating conjectures about incompleteness in the finite domain. *Theoretical Computer Science*, 847:76–94, 2020. doi:10.1016/j.tcs.2020.09.040.

11 T. Dose. An oracle separating conjectures about incompleteness in the finite domain. *Theoretical Computer Science*, 809:466–481, 2020. doi:10.1016/j.tcs.2020.01.003.

12 T. Dose and C. Glaßer. NP-completeness, proof systems, and disjoint NP-pairs. In C. Paul and M. Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.9.

13 J. Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards*, 69B:67–72, 1965. doi:10.6028/JRES.069B.004.

14 S. Fenner, L. Fortnow, A. Naik, and J. Rogers. On inverting onto functions. In *Proceedings 11th Conference on Computational Complexity*, pages 213–223. IEEE Computer Society Press, 1996.

15 S. A. Fenner, L. Fortnow, A. V. Naik, and J. D. Rogers. Inverting onto functions. *Information and Computation*, 186(1):90–103, 2003. doi:10.1016/S0890-5401(03)00119-6.

16 C. Glaßer, C. Reitwießner, and V. L. Selivanov. The shrinking property for NP and coNP. *Theoretical Computer Science*, 412(8-10):853–864, 2011. doi:10.1016/j.tcs.2010.11.035.

17 C. Glaßer, A. L. Selman, and S. Sengupta. Reductions between disjoint NP-pairs. *Information and Computation*, 200:247–267, 2005. doi:10.1016/j.ic.2005.03.003.

18 C. Glaßer, A. L. Selman, S. Sengupta, and L. Zhang. Disjoint NP-pairs. *SIAM Journal on Computing*, 33(6):1369–1416, 2004. doi:10.1137/S0097539703425848.

19 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.

20 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.

21 J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988. doi:10.1137/0217018.

22 J. Hartmanis and L. A. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58:129–142, 1988. doi:10.1016/0304-3975(88)90022-9.

23 Kannan, 1979. Sipser [36] cites an unpublished work by Kannan for asking if there is a set complete for NP ∩ coNP.

24 E. Khaniki. New relations and separations of conjectures about incompleteness in the finite domain. *ArXiv preprint*, 2019. arXiv:1904.01362.

25 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.

26 J. Krajícek 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.

27 N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. doi:10.1016/0304-3975(91)90200-L.

28 J. Messner. *On the Simulation Order of Proof Systems.* PhD thesis, Universität Ulm, 2000.

29 C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

30 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.

31 P. Pudlák. On reducibility and symmetry of disjoint NP pairs. *Theoretical Computer Science*, 295:323–339, 2003. doi:10.1016/S0304-3975(02)00411-5.

32 P. Pudlák. On some problems in proof complexity. In O. Beyersdorff, E. A. Hirsch, J. Krajícek, and R. Santhanam, editors, *Optimal algorithms and proofs (Dagstuhl Seminar 14421)*, volume 4, pages 63–63, Dagstuhl, Germany, 2014. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/DagRep.4.10.51.

33 P. Pudlák. Incompleteness in the finite domain. *The Bulletin of Symbolic Logic*, 23(4):405–441, 2017. doi:10.1017/bsl.2017.32.

34 A. Razborov. On provably disjoint NP-pairs. *BRICS Report Series*, 36, 1994. doi:10.7146/brics.v1i36.21607.

35 A. L. Selman. Promise problems complete for complexity classes. *Information and Computation*, 78:87–98, 1988. doi:10.1016/0890-5401(88)90030-2.

36 M. Sipser. On relativization and the existence of complete sets. In *Proceedings 9th ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 523–531. Springer Verlag, 1982. doi:10.1007/BFb0012797.

37 L. G. Valiant. Relative complexity of checking and evaluation. *Information Processing Letters*, 5:20–23, 1976. doi:10.1016/0020-0190(76)90097-1.

38 O. V. Verbitskii. Optimal algorithms for coNP-sets and the EXP=?NEXP problem. *Mathematical notes of the Academy of Sciences of the USSR*, 50(2):796–801, 1991. doi:10.1007/BF01157564.

Exact Matching in Graphs of Bounded Independence Number

Nicolas El Maalouly
Department of Computer Science, ETH Zürich, Switzerland

Raphael Steiner
Department of Computer Science, ETH Zürich, Switzerland

Abstract

In the *Exact Matching Problem* (EM), we are given a graph equipped with a fixed coloring of its edges with two colors (red and blue), as well as a positive integer k. The task is then to decide whether the given graph contains a perfect matching exactly k of whose edges have color red. EM generalizes several important algorithmic problems such as *perfect matching* and restricted minimum weight spanning tree problems.

When introducing the problem in 1982, Papadimitriou and Yannakakis conjectured EM to be **NP**-complete. Later however, Mulmuley et al. presented a randomized polynomial time algorithm for EM, which puts EM in **RP**. Given that to decide whether or not **RP**=**P** represents a big open challenge in complexity theory, this makes it unlikely for EM to be **NP**-complete, and in fact indicates the possibility of a *deterministic* polynomial time algorithm. EM remains one of the few natural combinatorial problems in **RP** which are not known to be contained in **P**, making it an interesting instance for testing the hypothesis **RP**=**P**.

Despite EM being quite well-known, attempts to devise deterministic polynomial algorithms have remained illusive during the last 40 years and progress has been lacking even for very restrictive classes of input graphs. In this paper we push the frontier of positive results forward by proving that EM can be solved in deterministic polynomial time for input graphs of bounded independence number, and for bipartite input graphs of bounded bipartite independence number. This generalizes previous positive results for complete (bipartite) graphs which were the only known results for EM on dense graphs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Perfect Matching, Exact Matching, Independence Number, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.46

Related Version *Full Version*: https://arxiv.org/abs/2202.11988

Funding *Raphael Steiner*: supported by an ETH Zurich Postdoctoral Fellowship.

1 Introduction

The problem of deciding whether a given graph contains a perfect matching, as well as the related problem of computing a maximum (minimum) weight perfect matching in a given graph are amongst the foundational problems in algorithmic graph theory and beyond, and the fact that they can be solved in polynomial time [4] is an integral part of many efficient algorithms in theoretical computer science.

In 1982, Papadimitriou and Yannakakis [17] studied a decision problem related to perfect matchings in edge-colored graphs as follows: Given as input a graph G whose edges come with a given fixed two-edge coloring (say, with colors red and blue), then the task is to decide whether for a given integer k there exists a perfect matching M of G such that exactly k of the edges in M are red. Clearly, in the special case when all edges are colored red and $k = \frac{n}{2}$,

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 46; pp. 46:1–46:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

this problem is simply to decide whether there exists a perfect matching in a given graph. For a heterogeneous coloring of the edges, however, the difficulty of the problem seems to change quite dramatically (see below).

The original motivation of Papadimitriou and Yannakakis [17] to study the above problem, which from now on will be called *Exact Matching* and abbreviated by EM, was their investigation of *restricted minimum weight spanning tree problems*. In the usual minimum weight spanning tree problem, we are given a graph with non-negative edge-weights and seek to find a spanning tree minimizing the total edge-weight, and this is well-known to be solvable in polynomial time using for instance Kruskal's algorithm [12]. Papadimitriou and Yannakakis considered what happens if we restrict the shape of the spanning trees allowed in the output, and obtained several results. For instance, the problem is easily seen to be **NP**-hard if the considered spanning trees are constrained to be paths, by a reduction from the Hamiltonian Path problem, but it is polynomial-time solvable if the tree shapes are restricted to stars or 2-stars. While for many classes of trees, Papadimitriou and Yannakakis [17] classified the complexity of the above problem, some cases remained unsettled. In particular, they proved that the restricted minimum weight spanning tree problem for so-called *double* 2-*stars* is equivalent to EM, and left it as an open problem to decide its computational complexity. In fact, they stated the conjecture that EM is **NP**-complete. Up until today, neither has this conjecture been confirmed, nor is it known whether EM can be solved in polynomial time by a deterministic algorithm. Yet, there have been some interesting results and developments regarding the problem in the past, which we summarize in the following.

Only few years after the introduction of the problem, in a breakthrough result Mulmuley, Vazirani and Vazirani [16] developed their so-called *isolation lemma*, and demonstrated its power by using it to prove that EM can be solved by a randomized polynomial time algorithm, i.e. it is contained in **RP**. This makes it unlikely to be **NP**-hard. In fact, deciding whether **RP**=**P** remains one of the big challenges in complexity theory. This means that problems such as EM, for which we know containment in **RP** but are not aware of deterministic polynomial time algorithms, are interesting candidates for testing the hypothesis **RP**=**P**. Indeed, due to this, EM is cited in several papers as an open problem. This includes recent breakthrough papers such as the seminal work on the parallel computation complexity of the matching problem [19], works on planarizing gadgets for perfect matchings [8], works on more general constrained matching problems [1, 14, 15, 18] and on multicriteria optimization problems [7] among others. Even though EM has caught the attention of many researchers from different areas, there seems to be a substantial lack of progress on the problem even when restricted to very special subclasses of input graphs as we will see next. This highlights the surprising difficulty of the problem given how simple it may seem at first glance.

Previous results for EM on restricted classes of graphs. It may surprise some readers that EM is even non-trivial if the input graphs are complete or complete bipartite graphs: In fact, at least four different articles have appeared on resolving these two special cases of EM [10, 20, 6, 9], which are now known to be solvable in deterministic polynomial time. Another positive result follows from the existence of Pfaffian orientations and their analogues on planar graphs and $K_{3,3}$-minor free graphs [21], EM is solvable in polynomial time on these classes via a derandomization of the techniques used in [16]. Considering a generalization of Pfaffian orientations, it was further proved in [5] that EM can be solved in polynomial time for graphs embeddable on a surface of bounded genus. Finally, from the well-known meta-theorem of Courcelle [2], one easily obtains that EM can be efficiently solved on classes of bounded tree-width.

Our contribution. In this paper, we generalize the known positive results for EM on very dense graphs such as complete and complete bipartite graphs to graphs of independence number at most α and to bipartite graphs of bipartite independence number at most β, for all fixed integers $\alpha, \beta \geq 1$. The *independence number* of a graph G is defined as the largest number α such that G contains an *independent set* of size α. The *bipartite independence number* of a bipartite graph G equipped with a bipartition of its vertices is defined as the largest number β such that G contains a *balanced independent set* of size 2β, i.e., an independent set using exactly β vertices from both color classes.

▶ **Theorem 1.** *There is a deterministic algorithm for EM on graphs of independence number α running in time $n^{O(f(\alpha))}$, for $f(\alpha) = 2^{O(\alpha)}$.*

▶ **Theorem 2.** *There is a deterministic algorithm for EM on bipartite graphs of bipartite independence number β running in time $n^{O(f(\beta))}$, for $f(\beta) = 2^{O(\beta)}$.*

The special cases $\alpha = 1$ and $\beta = 1$ of the above results correspond exactly to the previously studied cases of complete and complete bipartite graphs. We emphasize that even though bounding the independence number might seem like a big restriction on the input graphs, already for $\alpha = 2, \beta = 2$ our results cover rich and complicated classes of graphs, for instance every complement of a triangle-free graph belongs to the class of independence number at most 2, and every bipartite complement of a C_4-free bipartite graph belongs to the class of bipartite independence number at most 2.

Another interesting observation in support of the above is the following: So far, for all classes of graphs on which EM was known to be solvable in polynomial time (including planar graphs, $K_{3,3}$-minor-free graphs, graphs of bounded genus, complete and complete bipartite graphs), the number of perfect matchings was also known to be countable in polynomial time (cf. [11, 13, 5, 21]), and one may wonder about whether tractability of EM aligns with the tractability of corresponding counting problems for perfect matchings. However, even for graphs of independence number 2 we are not aware that polynomial schemes for counting perfect matchings exist, and in fact conjecture that this problem is computationally hard, therefore putting our result into nice contrast with previous positive results on EM.

▶ **Conjecture 3.** *The problem of counting perfect matchings in input graphs of independence number 2 is $\#\boldsymbol{P}$-complete.*

Organization of the paper. The remainder of this paper is organized as follows: In Section 2 we present the basic definitions and conventions we use throughout the paper. In Section 3 we prove Theorem 1, i.e., showing the existence of an XP algorithm parameterized by the independence number of the graph. In Section 4 we consider the bipartite graphs case and prove Theorem 2. In Section 5 we discuss distance-d independence number parameterizations and in Section 6 we conclude the paper and provide some open problems.

2 Preliminaries

Due to space restrictions, proofs of statements marked $\star$ have been deferred to the appendix. All graphs considered are simple. For a graph $G = (V, E)$ we let $n = |V(G)|$, i.e. the number of vertices in G. Given an instance of EM and a perfect matching[1] (abbreviated PM) M, we

[1] A perfect matching of a graph is a matching (i.e., an independent edge set) in which every vertex of the graph is incident to exactly one edge of the matching.

define edge weights as follows: blue edges get weight 0, matching red edges get weight -1 and non-matching red edges get weight $+1$. For G' a subgraph[2] of G, we define $R(G')$ (resp. $B(G')$) to be the set of red (resp. blue) edges in G', $r(G') := |R(G')|$ and $w_M(G')$ to be the sum of the weights of edges in G'. For ease of notation, we will use $w(G')$ for $w_M(G')$ and will make the matching explicit whenever it is not M.

Whenever we consider a set of cycles or paths, it is always assumed that they are vertex disjoint and alternating with respect to the current matching M (unless specified otherwise). Define an x-path to be an alternating path of weight x. Undirected cycles are considered to have an arbitrary orientation. For a cycle C and $u, v \in C$, $C[u, v]$ is defined as the path from u to v along C (in the fixed but arbitrarily chosen orientation). For simplicity, a cycle is also considered to be a path i.e. a closed path (its starting vertex is chosen arbitrarily). $Ram(r, s)$ refers to the Ramsey number, i.e. every graph on $Ram(r, s)$ vertices contains either a clique of size r or an independent set of size s. For simplicity we will use the following upper bound $Ram(s + 1, s + 1) < 4^s$ [3]. For two sets of edges M and M', $M\Delta M'$ refers to the symmetric difference between the two sets (i.e. the edges that appear in exactly one of the two sets). Note that if M and M' are two PMs, then $M\Delta M'$ forms a set of cycles (each alternating with respect to both matchings) and will be use as such. Also note that with the above defined edge weights we have $r(M') = r(M) + w(M\Delta M')$.

3 Bounded Independence Number Graphs

The algorithm relies on a 2 phase process. The first phase is an algorithm that outputs a PM M with $|k - r(M)|$ bounded (by a function of α), i.e. with a number of red edges that only differs from k by a function of α. This algorithm is also of independent interest since it provides a solution that is close to optimal (for small independence number) while its running time is polynomial and independent of the independence number.

▶ **Theorem 4.** *Given a "Yes" instance of EM, there exists a deterministic polynomial time algorithm that outputs a PM M with $k - 2 \cdot 4^\alpha \leq r(M) \leq k$.*

▶ Remark. Note that a standalone proof of Theorem 4 can be made quite simple but would require additional notions and definitions. Our main focus however, is on the proof of Theorem 1, so the proof structure is tailored towards that end and the proof of Theorem 4 will come as result along the way.

The second phase is an algorithm that outputs a solution matching with a running time that depends on the size of the smallest color class in a symmetric difference between a given matching and a solution matching. It is also of independent interest as it can be more generally useful for the study of other parameterizations of EM as well as other matching problems with color constraints.

▶ **Proposition 5.** *Let M and M' be two PMs in G s.t. $|B(M\Delta M')| \leq L$ or $|R(M\Delta M')| \leq L$. Then there exists a deterministic algorithm running in time $n^{O(L)}$ such that given M it outputs a PM M'' with $r(M'') = r(M')$.*

Proof. Suppose w.l.o.g. $|R(M\Delta M')| \leq L$ (the other case is similar by swapping the colors). Guess $R(M\Delta M')$ in time $n^{O(L)}$ by trying all possibilities (the rest of the algorithm should succeed for at least one such possibility). Compute $R(M') = R(M)\Delta R(M\Delta M')$.

[2] Note that the subgraph can also be a set of edges or cycles.

Remove the edges $R(M')$ and their endpoints from the graph as well as all remaining red edges. Compute a PM M_1 on the rest of the graph (such a PM must exist since $M' \backslash R(M')$ is one such example) and let $M'' := M_1 \cup R(M')$. Observe that M'' is a PM with $r(M'') = |R(M'')| = |R(M')| = r(M')$. ◀

For this phase to run in polynomial time for bounded independence number, we need to show that there exists a PM M^* with exactly k red edges, where $M \Delta M^*$ (M being the PM we get after the first phase) has a bounded (by a function of α) number of edges of some color class. The main technical challenge is to show that for this to be the case it is sufficient to have $|k - r(M)|$ bounded (which is guaranteed by the first phase). The rest of this section is devoted to this proof. Along the way we will also prove Theorem 4. Before going into the technical details, we give a quick overview.

3.1 Proof Overview

In order to apply Proposition 5, we will consider the solution matching M^* which minimizes the number of edges in $M \Delta M^*$ (M being the PM we get after the first phase) and aim to show that it contains a bounded number of edges of some color class. Towards this end, we want to show that if the set of alternating cycles $M \Delta M^*$ contains a large number of edges from both color classes, then there be must another set of alternating cycles $\mathcal{C}$ with the same total weight as $M \Delta M^*$, but containing strictly less edges. This contradicts the minimality of $M \Delta M^*$ since $M \Delta \mathcal{C}$ is also a solution matching with $|E(M \Delta (M \Delta \mathcal{C}))| = |E(\mathcal{C})| < |E(M \Delta M^*)|$. In other words, we want to show that unless one color class in $M \Delta M^*$ is bounded, we can reduce the size of one or more of the cycles in $M \Delta M^*$ while keeping the total weight unchanged.

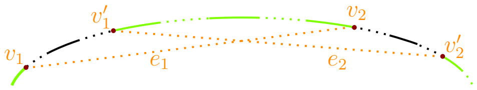


Figure 1 A skip formed by two non-matching edges e_1 and e_2 (in orange). Matching edges are represented by full lines and non-matching edges by dotted lines. The paths removed by the skip are depicted in black.

Skips. The main tool we use to show the existence of smaller alternating cycles is something we call a skip (see Figure 1). At a high level, a skip is simply a pair of edges that creates a new alternating cycle C' by replacing two paths of an alternating cycle C. If those paths have total length more than 2 then $|C'| < |C|$. This means that if a solution matching M^{**} exists, such that $M \Delta M^{**}$ is the same as $M \Delta M^*$ but with C replaced by C', it would contradict the minimality of $M \Delta M^*$. For M^{**} to be a solution matching, we also need $w(C) = w(C')$ so that M^{**} also has k red edges. For this reason we look for skips that do not change the total weight (we call them 0 skips). It can happen however, that even though no 0 skip exists, a collection of skips exists, that can be used independently, and their total weight change is zero (we call them 0 skip sets). Also observe that these skips can come from different cycles of $M \Delta M^*$ and still be used to reduce its total number of edges (i.e. we can modify multiple cycles in $M \Delta M^*$ simultaneously to preserve the total weight change). So by taking $M \Delta M^*$ to be minimal (in terms of total number of edges), we are guaranteed that no such skip sets can exist.

Skips from Paths. To show the existence of skips (which will lead to the desired contradiction), we rely on Ramsey theory to show that if we take a large enough (with respect to α) collection of disjoint paths on an alternating cycle, starting and ending with non-matching edges, then they must form skips. Now if these paths have certain desired weights, then we could make sure that we get a 0 skip set as desired.

Paths from Edge Pairs. To prove the existence of paths of desired weight, we analyze the cycles in $M \Delta M^*$ by looking at their edge pairs, i.e. pairs of consecutive matching and non-matching edges. These edge pairs can have 3 configurations from which we can extract the paths. (1) Consecutive same sign pairs (sign here refers to the weight of the pair), (2) consecutive different sign pairs and (3) consecutive 0 pairs. We show that we can extract paths of the desired properties from all of these configurations, and the types of skips we get is dependent on the weights of the cycles and the sizes of their color classes.

Bounding the Cycle Weights. Next, we show that if $M \Delta M^*$ is minimal, all of its cycles have bounded weight. This is mainly achieved by showing that cycles of large weight must have skips that reduce their weight. This changes the total weight of $M \Delta M^*$ however, and must be compensated for either by skips on a cycle of opposite sign weight, or by removing some of the cycles in $M \Delta M^*$.

Bounding one color class. After bounding the weights of the cycles in $M \Delta M^*$ (by a function of α), we will also bound their number given that their total weight is bounded. With these properties (bounded cycle weights and number of cycles), we can show that if $M \Delta M^*$ has enough edges from both colors, then at least one of its cycles contains enough positive skips and one of its cycles contains enough negative skips, together forming a 0 skip set, i.e. $M \Delta M^*$ is not minimal. So choosing $M \Delta M^*$ minimal implies a bound on the size of one of its color classes.

3.2 Detailed Proof

Skips. We start by formally defining a skip and its properties.

▶ **Definition 6.** *Let C be an alternating cycle. A skip S is a set of two non-matching edges $e_1 := (v_1, v_2)$ and $e_2 := (v_1', v_2')$ with $e_1, e_2 \notin C$ and $v_1, v_1', v_2, v_2' \in C$ (appearing in this order along C) s.t. $C' = e_1 \cup e_2 \cup (C \setminus C[v_1, v_1'] \cup C[v_2, v_2'])$ is an alternating cycle, $|C| - |C'| > 0$ and $|w(S)| \leq 4$ where $w(S) := w(C') - w(C)$ is called the weight of the skip.*

Note that we require a skip to have weight at most 4. This is mainly to simplify the analysis since it is enough to only consider such skips.

If $P \subseteq C$ is a path and $C[v_1, v_2'] \subseteq P$, then we say that P contains the skip S. We say using S to mean replacing C by C'. If $C \in M \Delta M'$ for some PM M', then by using S we also modify M' accordingly (i.e. s.t. $M \Delta M'$ now contains C' instead of C). Observe that a positive skip (where positive refers to the weight of the skip) increases the cycle weight, a negative skip decreases it and a 0 skip does not change the cycle weight. Using a skip always results in a cycle of smaller cardinality. Two skips $\{(v_1, v_2), (v_1', v_2')\}$ and $\{(u_1, u_2), (u_1', u_2')\}$ are called disjoint if they are contained in disjoint paths along the cycle. Note that two disjoint skips can be used independently.

▶ **Definition 7.** *Let $\mathcal{C}$ be a set of alternating cycles. A 0 skip set is a set of disjoint skips on cycles of $\mathcal{C}$ s.t. the total weight of the skips is 0.*

Observe that finding a skip with some desired properties can be done in polynomial time by trying all possible combinations of 2 edges, every time checking if the edges form a skip with the desired properties (i.e. checking if the resulting cycle C' is alternating, has strictly less edges then C and the weight change is as desired, which can all be done in polynomial time).

Skips from Paths. Next, we show that if a cycle contains a lot of disjoint paths then it must contain a skip that replaces 2 of these paths by its 2 edges.

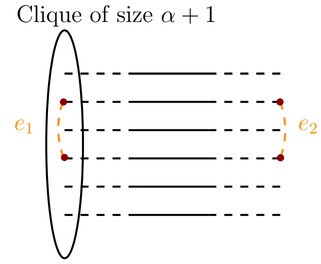


Figure 2 A subset of $\mathcal{P}$ of size $\alpha + 1$ whose starting vertices form a clique.

▶ **Lemma 8.** *Let P be an alternating path containing a set $\mathcal{P}$ of disjoint paths, each of length at least 3 and starting and ending at non-matching edges, of size $|\mathcal{P}| \geq 4^{\alpha}$. Then P contains a skip. If all paths in $\mathcal{P}$ have the same weight x, then if x is one of the following values, we get the following types of skips:*

- $x = 2$: *negative skip.*
- $x = 1$: *negative or 0 skip.*
- $x = 0$: *positive or 0 skip.*
- $x = -1$: *positive skip.*

Proof. The set of starting vertices of the paths in $\mathcal{P}$ must contain a clique Q of size $\alpha + 1$ since $|\mathcal{P}| > Ram(\alpha + 1, \alpha + 1)$ (and the independence number of the graph is α). Let $\mathcal{P}'$ be the set of paths from $\mathcal{P}$ starting with vertices in Q and Q' their set of ending vertices. Since $|Q'| = \alpha + 1$, there must be an edge connecting two of its vertices, call it e_2. Let P_1 and P_2 be the two paths in $\mathcal{P}'$ connected by e_2. Let e_1 be the edge connecting the starting vertices of P_1 and P_2 (which must exist since Q is a clique). Note that e_1 and e_2 must be non-matching edges since they are chords of the alternating cycle C so their endpoints are matched to edges of C. Now observe that e_1 and e_2 form a skip S (see Figure 2) and $w(S) = w(e_1) + w(e_2) - w(P_1) - w(P_2)$. Finally, suppose P_1 and P_2 have weight x and note that $w(e_1), w(e_2) \in \{0, 1\}$ since they are non-matching edges. We get $-2x \leq w(S) \leq 2 - 2x$ thus proving the lemma. ◀

The above lemma only shows the existence of a skip of a certain sign, and does not guarantee the existence of 0 skips, i.e. skips that do not change the cycle weight. The next lemma shows that if there are enough disjoint positive and negative skips we can still obtain a 0 skip set (i.e. we can still reduce the cardinality of $M \Delta M'$ without changing its weight).

▶ **Lemma 9** (⋆). *Let $\mathcal{S}$ be a collection of disjoint skips. If $\mathcal{S}$ contains at least 4 positive skips and at least 4 negative skips (all mutually disjoint), then $\mathcal{S}$ must contain a 0 skip set.*

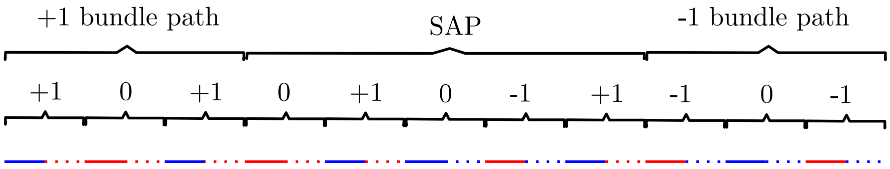


Figure 3 An example of an alternating path containing a $+1$ bundle, a -1 bundle and an SAP. Matching edges are represented by full lines and non-matching edges by dotted lines. The colors of the edges correspond to their color in the graph.

Edge Pairs. For a given alternating cycle, our goal is to find paths of some desired weight in order to apply Lemma 8. To make finding these paths easier, we look at pairs of edges. Each pair consists of two consecutive edges (the first a matching-edge and the second a non-matching edge). We label the pairs according to their weight (see in Figure 3 the label above each pair of edges).

▶ **Definition 10.** *A $+1$ pair (resp. -1 pair and 0 pair) is a pair of consecutive edges (the first a matching-edge and the second a non-matching edge) along an alternating cycle such that their weight sums to 1 (resp. -1 and 0).*

Two $+1$ (resp. -1) pairs are called consecutive if there is an alternating path between them on the cycle which only contains 0 pairs.

▶ **Definition 11.** *A $+1$ (resp. -1) bundle is a pair of edge-disjoint consecutive $+1$ (resp. -1) pairs. The path starting at the first pair and ending at the second one (including both pairs) is referred to as the bundle path (see Figure 3 for an example of such bundles).*

Note that a $+1$ (resp. -1) bundle path has weight $+2$ (resp. -2). Two bundles are called disjoint if their bundle paths are edge disjoint.

▶ **Definition 12.** *A Sign Alternating Path (SAP) is an alternating path P formed by edge pairs, such that it does not contain any bundles (see Figure 3 for an example of such path).*

Note that for an SAP P, $|w(P)| \leq 1$.

Paths from Edge Pairs. Our goal is to bound the number of edges from some color class in $M\Delta M^*$, when the latter is chosen to contain a minimum number of edges. To this end, we aim to show that a large number of edges of both color classes implies the existence of a 0 skip set (which would contradict the minimality of $M\Delta M^*$). By Lemma 9 it suffices to show the existence of many positive and negative skips which in turn can be a result of many paths of certain weight (by Lemma 8).

In the next two lemmas, we first show that a large number of edges of some color class implies the existence of either many bundles, a long SAP or many 0-paths starting with an edge of that color class. Then we show that all of these structures result in paths of the desired weights.

▶ **Lemma 13** ($\star$). *Let P be an alternating path containing at least $10t^3$ blue (resp. red) edges. Then one of the following properties must hold:*
(a) *P contains at least t disjoint bundles.*
(b) *P contains an SAP with at least t non-zero pairs.*
(c) *P contains at least t edge-disjoint 0-paths of length at least 4 starting with a blue (resp. red) matching edge.*

▶ **Lemma 14** (⋆). *A path P, satisfying one of the following properties, must contain t disjoint paths each of length at least 3, starting and ending with non-matching edges and having specific weights that depend on the satisfied property:*

(a) *P contains t disjoint $+1$ bundles: paths of weight $+2$.*
(b) *P contains t disjoint -1 bundles: paths of weight -1.*
(c) *P contains t edge-disjoint 0-paths of length at least 4 starting with a red matching edge: paths of weight $+1$.*
(d) *P contains t edge-disjoint 0-paths of length at least 4 starting with a blue matching edge: paths of weight 0.*
(e) *P contains an SAP with at least $2t+1$ non-zero pairs: paths of weight $+1$.*
(f) *P contains an SAP with at least $2t+1$ non-zero pairs: paths of weight 0.*

While the above lemmas would be enough to show the existence of many skips whenever $M\Delta M^*$ contains many edges from both color classes, these skips can still be of the same sign (e.g. all positive) which is not enough to use Lemma 9. We will later show that this cannot happen if the cycles in $M\Delta M^*$ have bounded weight.

Bounding Cycle Weights. In this part, we will deal with cycles of unbounded weight. We start by showing that a large cycle weight implies the existence of many skips that can be used to reduce it.

▶ **Lemma 15** (⋆). *Let P be an alternating path with $w(P) \geq 2t \cdot 4^{\alpha}$ (resp. $w(P) \leq -2t \cdot 4^{\alpha}$), then P contains at least t disjoint negative (resp. positive) skips.*

The above lemma also allows for simple proof of Theorem 4.

Proof of Theorem 4. Let M_1 be a PM containing a minimum number of red edges and M_2 a PM with a maximum number of red edges (should be at least k). Note that M_1 (resp. M_2) can be computed in polynomial time by simply using a maximum weight perfect matching algorithm with -1 (resp. $+1$) weights assigned to red edges and 0 weights assigned to blue edges.

Now as long as $r(M_1) \leq k - 2 \cdot 4^{\alpha}$ and $r(M_2) > k$ we will apply the following procedure (otherwise we output $M := M_1$): Let $C \in M_1\Delta M_2$ with $w(C) > 0$ (such a cycle must exist since $r(M_1) < r(M_2)$). If $w(C) \leq 2 \cdot 4^{\alpha}$ then we replace M_1 by $M_1\Delta C$ and iterate (note that $r(M_1) < r(M_1\Delta C) \leq k$). Otherwise, by Lemma 15, C contains a negative skip. We find it in polynomial time and use it to reduce the cycle weight, and iterate the whole procedure (note that $r(M_2)$ decreases). If at any point $r(M_2)$ drops below k, we simply output $M := M_2$. In all cases $w(M_1\Delta M_2)$ decreases after every iteration. So there can be at most n iterations (since the PMs have at most $n/2$ edges each, so $w(M_1\Delta M_2) \leq n$ and we only iterate as long as it is bigger than 0), each running in polynomial time. ◀

▶ Remark. Note that the proof only relies on Lemma 15 which in turn only relies on Lemma 8 and the part of Lemma 14 that deals with bundles. Most of the previously defined notions are not needed for this standalone result.

From Lemma 15 we get that if $M\Delta M^*$ contains both a positive cycle of unbounded weight and a negative cycle of unbounded weight, we can find a 0 skip set using Lemma 9. It could be the case however, that we have only one of the two, say a positive cycle of unbounded weight (with respect to α), and many negative weight cycles (which would be required if $|w(M\Delta M^*)|$ is bounded, which is guaranteed by the first phase of the algorithm). In this case we can get many negative skips from the positive weight cycle of unbounded weight but we are not guaranteed to find positive skips, so we need another way to compensate for the total weight change. Notice that this can be achieved by removing negative cycles from $M\Delta M^*$. So we will combine the use of negative skips with the removal of some of the negative cycles in order to get a zero total weight change. We call this a 0 skip-cycle set.

▶ **Definition 16.** *Let $\mathcal{C}$ be a set of alternating cycles. A 0 skip-cycle set is a set of disjoint skips on cycles of $\mathcal{C}$ and/or cycles from $\mathcal{C}$, s.t. the total weight of the skips minus the total weight of the cycles is 0.*

We say that we use a skip-cycle set $\mathcal{S}$ to mean that we use all skips in $\mathcal{S}$ and remove all cycles in $\mathcal{S}$ from $\mathcal{C}$. Note that a 0 skip set is a 0 skip-cycle set. Also a cycle $C \in M\Delta M^*$ with $w(C) = 0$ is a 0 skip-cycle set. The following lemma shows that if a set of alternating cycles has bounded weight but one of its cycles has an unbounded weight then it must contain a 0 skip-cycle set.

▶ **Lemma 17** ($\star$). *Let $t \geq 8 \cdot 4^{\alpha}$ and $t' = 4t^2$. Let $\mathcal{C}$ be a set of alternating cycles and $C \in \mathcal{C}$ s.t. $|w(\mathcal{C})| \leq t'$ and $|w(C)| \geq 2t'$, then $\mathcal{C}$ contains a 0 skip-cycle set.*

Bounding one color class. So far we have shown that we can bound the weight of the cycles in $M\Delta M^*$ (if $M\Delta M^*$ is minimal). What we want to show next is that if $M\Delta M^*$ contains many blue (resp. red) edges and all of its cycles have bounded weight, then it also contains many positive (resp. negative) skips. This way we show that having many edges of both color classes results in a 0 skip set.

First we deal with the case when the number of cycles in $M\Delta M^*$ is unbounded. The following lemma shows that if the number of cycles is large enough compared to their individual and total weights, then there must be a subset of them of 0 total weight (i.e., a 0 skip-cycle set).

▶ **Lemma 18** ($\star$). *Let $t \geq 3$. Let $\mathcal{C}$ be a set of alternating cycles s.t. $|w(\mathcal{C})| \leq t$, $|w(C)| \leq 2t$ for all $C \in \mathcal{C}$ and $|\mathcal{C}| \geq 10t^3$, then $\mathcal{C}$ contains a 0 skip-cycle set.*

Now we deal with the case when the number of cycles in $M\Delta M^*$ is bounded. In this case, for the number of edges of some color class to be unbounded, it has to be unbounded on at least one of the cycles. Lemma 21 deals with this case by first using Lemma 13 to show the existence of many bundles, many 0-paths starting with a red edge and many 0-paths starting with a blue edge, or a long SAP. In the latter two cases, we can prove the existence of both positive and negative skips resulting in a 0 skip set. In the case of bundles however, we need to have both many $+1$ and many -1 bundles for this to work. In Lemma 19 we show that if the weight of an alternating cycle is bounded, then the difference between the number of $+1$ and -1 bundles is also bounded. This in turn allows us to prove that the existence of many bundles results in a 0 skip set as well (see Lemma 20).

▶ **Lemma 19** ($\star$). *Let C be a cycle with $|w(C)| \leq l$. If C contains $3t + l$ disjoint -1 (resp. $+1$) bundles, then C also contains at least t disjoint $+1$ (resp. -1) bundles.*

▶ **Lemma 20** ($\star$). *Let $t \geq 8 \cdot 4^{\alpha}$. Let C be a cycle with $|w(C)| \leq 2t$. If C contains more than $10t$ disjoint bundles then it must contain a 0 skip set.*

▶ **Lemma 21** ($\star$). *Let $t \geq 8 \cdot 4^{\alpha}$. Let $\mathcal{C}$ be a collection of cycles s.t. $|\mathcal{C}| \leq 10t^3$, $|w(C)| \leq 2t$ for all $C \in \mathcal{C}$ and $\mathcal{C}$ contains at least $1000t^6$ blue edges and $1000t^6$ red edges, then $\mathcal{C}$ contains a 0 skip set.*

Proof of Theorem 1. Use the algorithm of Theorem 4 to get a matching M s.t. $k - 2 \cdot 4^{\alpha} \leq r(M) \leq k$. Let M^* be a PM with k red edges that minimizes $|E(M \Delta M^*)|$. Consider the set of cycles $M \Delta M^*$. Observe that it cannot contain a 0 skip-cycle set (by minimality of its number of edges) and $|w(M \Delta M^*)| \leq |k - r(M)| \leq 2 \cdot 4^{\alpha}$. Let $t = 256 \cdot 4^{2\alpha}$ (so t is large enough to apply all the previous lemmas). If some cycle $C \in M \Delta M^*$ has $|w(C)| \geq 2t$, by Lemma 17 we get a 0 skip-cycle set. So we consider the case when all cycles $C \in M \Delta M^*$ have $|w(C)| < 2t$. If $M \Delta M^*$ contains at least $10t^3$ cycles, by Lemma 18 we get a 0 skip set. So we consider the case when $|M \Delta M^*| \leq 10t^3$. By Lemma 21, since $M \Delta M^*$ does not contain a 0 skip set, it must contain at most $f(\alpha)$ edges of some color class (for $f(\alpha) = 1000 \cdot (256 \cdot 4^{2\alpha})^6 = 2^{O(\alpha)}$). By Proposition 5 we can find a PM with exactly k red edges in $n^{O(f(\alpha))}$ time if one exists. ◀

4 Bipartite Graphs

In this section, we consider Bipartite graphs, which contain very large independent sets ($\geq n/2$). For this reason, we instead parameterize by the bipartite independence number β. Note that for the proof of Theorem 1 the only time we used the bounded independence number is in the proof of Lemma 8. So we need an analogue of it that works for bounded bipartite independence number, which will be given in Lemma 23. We will also need a new notion of a skip that better fits the bipartite case. We call it a biskip (see Definition 22 and Figure 4). We will also rely on an orientation of the edges of the graph defined as follows. Given a bipartite graph G with bipartition (A, B) and a matching M, we transform G into a directed graph G_M by orienting every matching edge from A to B and every non-matching edge from B to A.

▶ **Definition 22.** *Let C be a directed alternating cycle. A biskip S is a set of 2 arcs $a_1 := (v_1, v_2)$ and $a_2 := (v_1', v_2')$ with $a_1, a_2 \notin C$ and $v_1, v_2', v_1', v_2 \in C$ (appearing in this order along C) s.t. $C_1 := C[v_2, v_1] \cup a_1$ and $C_2 := C[v_2', v_1'] \cup a_2$ are vertex disjoint alternating cycles, $|C| - |C_1| - |C_2| > 0$ and $|w(S)| \leq 4$ where $w(S) := w(C_1) + w(C_2) - w(C)$ is called the weight of the biskip.*

If $P \subseteq C$ is a path and $C[v_1, v_2] \subseteq P$, then we say that P contains the biskip S. We say using S to mean replacing C by C_1 and C_2. If $C \in M \Delta M'$ for some PM M', then by using S we also modify M' accordingly (i.e. s.t. $M \Delta M'$ now contains C_1 and C_2 instead of C). Two skips $\{(v_1, v_2), (v_1', v_2')\}$ and $\{(u_1, u_2), (u_1', u_2')\}$ are called disjoint if they are contained in disjoint paths.

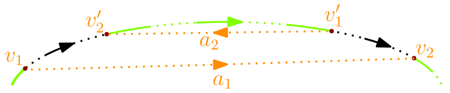


Figure 4 A biskip formed by two non-matching arcs a_1 and a_2 (in orange). Matching edges are represented by full lines and non-matching edges by dotted lines. The paths removed by the biskip are depicted in black.

▶ Remark. Note that the biskip could have been defined with one arc instead of two (since in this case one arc is enough to shorten an alternating cycle), which would have made the definition simpler. Definition 22 is however, very similar to the definition of the skip (see Definition 6) and this in turn allows us to prove Theorem 2 in an analogous way to Theorem 1 instead of requiring a completely different proof.

▶ **Lemma 23.** *Let $P \subseteq G_M$ be a directed alternating path containing a set $\mathcal{P}$ of disjoint directed paths, each of length at least 3 and starting and ending at a non-matching edge, s.t. $|\mathcal{P}| \geq 4^{2\beta+2}$. Then P contains a biskip. If all paths in $\mathcal{P}$ have the same weight x, then if x is one of the following values, we get the following types of biskips:*

- *$x = 2$: negative biskip.*
- *$x = 1$: negative or 0 biskip.*
- *$x = 0$: positive or 0 biskip.*
- *$x = -1$: positive biskip.*

Proof. Consider the graph G_C defined as follows: $V(G_C) = \mathcal{P}$, there is an edge between two vertices if their corresponding paths have an arc that goes from the start vertex of the first path to the end vertex of the second.

▷ Claim. G_C has independence number bounded by $2\beta + 1$.

Proof. Take any subset Q of vertices of G_C of size $2\beta + 2$. Let Q_1 be $\beta + 1$ consecutive (along C) vertices of Q and Q_2 the rest. Let V_1 be the set of start vertices of the paths corresponding to Q_1 in G_M and V_2 be the set of end vertices of the paths corresponding to Q_2 in G_M. Observe that $V_1 \cup V_2$ is a balanced set of size $2\beta + 2$, so there must be an arc connecting two of its vertices. Observe that the arc must be going from V_1 to V_2 since it corresponds to a non-matching edge. So G_C contains an edge corresponding to this arc, i.e. Q is not an independent set. ◁

G_C must contain a clique Q of size $2\beta + 2$ since $|V(G_C)| = |\mathcal{P}| \geq \mathit{Ram}(2\beta + 2, 2\beta + 2)$. Let Q_1 be $\beta + 1$ consecutive (along C) vertices of Q and Q_2 the rest. Let V_1 be the set of end vertices of the paths corresponding to Q_1 in G_M and V_2 be the set of start vertices of the paths corresponding to Q_2 in G_M. Observe that $V_1 \cup V_2$ is a balanced set of size $2\beta + 2$, so there must be an arc (call it a_1) connecting two of its vertices. Observe that the arc must be going from V_2 to V_1 since it corresponds to a non-matching edge. Let P_1 and P_2 be the paths corresponding to its start and end vertices. P_1 and P_2 must be connected by an edge in Q, let a_2 be the corresponding arc in G_M. So a_2 connects the start of P_1 to the end of P_2 and a_1 connects the start of P_2 to the end of P_1. Observe that a_1 and a_2 form a biskip S and $w(S) = w(a_1) + w(a_2) - w(P_1) - w(P_2)$. Let $x = w(P_1) = w(P_2)$ (x depends on the type of paths considered) and note that $w(e_1), w(e_2) \in \{0, 1\}$. We get $-2x \leq w(S) \leq 2 - 2x$ thus proving the lemma. ◀

The rest of the proof of Theorem 2 follows the same structure as that of Theorem 1 while using biskips instead of skips. Due to lack of space we will defer all the details to the appendix where we will restate all the definitions and lemmas that need to be adapted.

5 Distance-d Independence Number

In this section we show that the algorithms developed for small independence number graphs cannot be generalized to distance-d independence number, for $d > 2$, unless they can be used to solve EM on any graph. A distance-d independent set is a set of vertices at pairwise distance at least d (i.e. the shortest path between any two of them contains at least d edges) and the distance-d independence number is the size of the largest such set. Note that for $d = 2$ we get the normal independence number. Let $\alpha_d(G)$ be the distance-d independence number of a graph G.

▶ **Theorem 24.** *EM can be reduced to EM on graphs with $\alpha_d(G) = 1$, for any $d > 2$, in deterministic polynomial time.*

Proof. Given a graph $G = (V, E)$ we construct another graph $G' = (V', E')$ by adding two new vertices u and v s.t. $V' = V \cup \{u, v\}$ and $E' := E \cup (u, v) \cup \{(u, x) : x \in V\}$. All edges in E keep their colors while new edges get color blue. Observe that any PM on G' must contain (u, v) since it is the only edge connected to v, so by removing this edge from the PM we get a PM for G. Also note that G' has distance-d independence number of 1, for any $d > 2$, since any two vertices are connected to u, i.e. have distance 2. Now if there exists a PM M with exactly k red edges in G', we know that $M\backslash(u, v)$ is a PM with exactly k red edges in G. ◀

A similar theorem applies for bipartite graphs. Note that here we do not need to consider balanced independent sets (a similar result holds if we do).

▶ **Theorem 25.** *EM on bipartite graphs can be reduced to EM on bipartite graphs with $\alpha_d(G) = 2$, for any $d > 2$, in deterministic polynomial time.*

Proof. Given a bipartite graph $G = (U, V, E)$ we construct another bipartite graph $G' = (U', V', E')$ s.t. $U' = U \cup \{u, u'\}$, $V' = V \cup \{v, v'\}$ and $E' := E \cup \{(u, x) : x \in V'\} \cup \{(v, x) : x \in U'\}$. All edges in E keep their colors while new edges get color blue. Observe that any PM on G' must contain (u, v') and (v, u') since they are the only edges connected to u' and v', so by removing these edges from the PM we get a PM for G. Also note that G' has distance-d independence number of 2, for any $d \geq 2$, since it can contain at most one vertex from each of U' and V' (any two vertices of U' are connected to v, and any two vertices of V' are connected to u). Now if there exists a PM M with exactly k red edges in G', we know that $M\backslash((u, v') \cup (v, u'))$ is a PM with exactly k red edges in G. ◀

6 Conclusion and Open Problems

In this paper we initiated the study of the parameterized complexity of EM by showing that it can be solved in deterministic polynomial time on graphs of bounded independence number and bipartite graphs of bounded bipartite independence number (i.e. we developed XP algorithms). This is an important step towards finding the right complexity class of the problem in general graphs as it generalizes the only previously known results on dense graph classes which were restricted to complete (bipartite) graphs.

A natural next step would be to design corresponding FPT-algorithms in which the exponent in the running time is independent of the independence number. Another future direction would be to study the parameterized complexity of EM for other graph parameters. As we showed, parameterizing by higher distance independence numbers does not provide any additional structure, so it would be interesting to find other parameters that could yield non-trivial structure. Finally, it would be interesting to prove our conjecture on the hardness of counting PMs in graphs of independence number 2 or to find deterministic polynomial time algorithms for EM that work on graph classes for which counting PMs is #**P**-hard.

References

1 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.

2 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

3 Norman Do. Party problems and ramsey theory. *Vinculum*, 56(2):18–19, 2019.

4 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

5 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6:R6, 1999.

6 Hans-Florian Geerdes and Jácint Szabó. A unified proof for karzanov's exact matching theorem. Technical Report QP-2011-02, Egerváry Research Group, Budapest, 2011. URL: `https://egres.elte.hu/`.

7 Fabrizio Grandoni and Rico Zenklusen. Optimization with more than one budget. *arXiv preprint*, 2010. `arXiv:1002.2147`.

8 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.

9 Rohit Gurjar, Arpita Korwar, Jochen Messner, and Thomas Thierauf. Exact perfect matching in complete graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(2):1–20, 2017.

10 AV Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics*, 23(1):8–13, 1987.

11 Pieter W Kasteleyn. Graph theory and crystal physics. *Harary, F. (ed.), Graph Theory and Theoretical Physics*, pages 43–110, 1967.

12 Joseph B Kruskal. Paths, trees, and flowers. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

13 Charles H C Little. Kasteleyn's theorem and arbitrary graphs. *Canadian Journal of Mathematics*, 25(4):758–764, 1973.

14 Monaldo Mastrolilli and Georgios Stamoulis. Constrained matching problems in bipartite graphs. In *International Symposium on Combinatorial Optimization*, pages 344–355. Springer, 2012.

15 Monaldo Mastrolilli and Georgios Stamoulis. Bi-criteria and approximation algorithms for restricted matchings. *Theoretical Computer Science*, 540:115–132, 2014.

16 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

17 Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.

18 Georgios Stamoulis. Approximation algorithms for bounded color matchings via convex decompositions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 625–636. Springer, 2014.

19 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. Ieee, 2017.

20 Tongnyoul Yi, Katta G Murty, and Cosimo Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121(1-3):261–277, 2002.

21 Raphael Yuster. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.

CNF Encodings of Parity

Gregory Emdin
St. Petersburg State University, Russia

Alexander S. Kulikov
Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia
St. Petersburg State University, Russia

Ivan Mihajlin
Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia

Nikita Slezkin
Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia

Abstract

The minimum number of clauses in a CNF representation of the parity function $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ is 2^{n-1}. One can obtain a more compact CNF encoding by using non-deterministic variables (also known as guess or auxiliary variables). In this paper, we prove the following lower bounds, that almost match known upper bounds, on the number m of clauses and the maximum width k of clauses: 1) if there are at most s auxiliary variables, then $m \geq \Omega\left(2^{n/(s+1)}/n\right)$ and $k \geq n/(s+1)$; 2) the minimum number of clauses is at least $3n$. We derive the first two bounds from the Satisfiability Coding Lemma due to Paturi, Pudlák, and Zane using a tight connection between CNF encodings and depth-3 circuits. In particular, we show that lower bounds on the size of a CNF encoding of a Boolean function imply depth-3 circuit lower bounds for this function.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Circuit complexity

Keywords and phrases encoding, parity, lower bounds, circuits, CNF

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.47

Related Version *Full Version*: https://arxiv.org/abs/2203.01082

Acknowledgements Research is partially supported by Huawei (grant TC20211214628).

1 Overview

1.1 Motivation

A popular approach for solving a difficult combinatorial problem in practice is to encode it in conjunctive normal form (CNF) and to invoke a SAT-solver. There are two main reasons why this approach works well for many hard problems: the state-of-the-art SAT-solvers are extremely efficient and many combinatorial problems are expressed naturally in CNF. At the same time, a CNF encoding is not unique and one usually determines a good encoding empirically. Moreover, there is no such thing as the best encoding of a given problem as it also depends on a SAT-solver at hand. Prestwich [10] gives an overview of various ways to translate problems into CNF and discusses their desirable properties, both from theoretical and practical points of view.

Already for such simple functions as the parity function $x_1 \oplus x_2 \oplus \cdots \oplus x_n$, it is not immediate how to encode them in CNF (to make it efficiently handled by SAT-solvers). Parity function is used frequently in cryptography (hash functions, stream ciphers, etc.). It is known that the minimum number of clauses in a CNF computing parity is 2^{n-1}. This becomes impractical quickly as n grows. A standard way to reduce the size of an encoding is by using non-deterministic variables (also known as guess or auxiliary variables). Namely, one introduces s non-deterministic variables $y_1, \dots, y_s$ and partitions the set of input variables into $s+1$ blocks of size at most $\lceil n/(s+1) \rceil$: $\{x_1, x_2, \dots, x_n\} = X_1 \sqcup X_2 \sqcup \dots \sqcup X_{s+1}$. Then, one writes down the following $s+1$ parity functions in CNF:

$$\left(y_1 = \bigoplus_{x \in X_1} x\right), \left(y_2 = y_1 \oplus \bigoplus_{x \in X_2} x\right), \dots,$$
$$\left(y_s = y_{s-1} \oplus \bigoplus_{x \in X_s} x\right), \left(1 = y_s \oplus \bigoplus_{x \in X_{s+1}} x\right). \quad (1)$$

The value for the parameter s is usually determined experimentally. For example, Prestwich [9] reports that taking $s = 10$ gives the best results when solving the minimal disagreement parity learning problem using local search based SAT-solvers.

The simple construction above implies several upper bounds on the number m of clauses, the number s of non-deterministic variables, and the width k of clauses:

Limited non-determinism: using s non-deterministic variables, one can encode parity either as a CNF with at most

$$m \le (s+1)2^{\lceil n/(s+1) \rceil + 2 - 1} \le 4(s+1)2^{n/(s+1)}$$

clauses or as a k-CNF, where

$$k = 2 + \lceil n/(s+1) \rceil \le 3 + n/(s+1)\,.$$

Unlimited non-determinism: one can encode parity as a CNF with at most $4n$ clauses (to do this, use $s = n-1$ non-deterministic variables; then, each of n functions in (1) can be written in CNF using at most four clauses).

1.2 Results

In this paper, we show that the upper bounds mentioned above are essentially optimal.

▶ **Theorem 1.** *Let F be a CNF-encoding of PAR_n with m clauses, s non-deterministic variables, and maximum clause width k.*

1. *The parameters s and m cannot be too small simultaneously:*

$$m \ge \Omega\left(\frac{s+1}{n} \cdot 2^{n/(s+1)}\right). \quad (2)$$

2. *The parameters s and k cannot be too small simultaneously:*

$$k \ge n/(s+1)\,. \quad (3)$$

3. *The parameter m cannot be too small:*

$$m \ge 3n - 9\,. \quad (4)$$

1.3 Techniques

We derive a lower bound $m \ge \Omega((s+1)2^{n/(s+1)}/n)$ from the Satisfiability Coding Lemma due to Paturi, Pudlák, and Zane [8]. This lemma allows to prove a $2^{\sqrt{n}}$ lower bound on the size of depth-3 circuits computing the parity function. Interestingly, the lower bound $m \ge \Omega((s+1)2^{n/(s+1)}/n)$ implies a lower bound $2^{\Omega(\sqrt{n})}$ almost immediately, though it is not clear whether a converse implication can be easily proved.

To prove a lower bound $m \ge 3n - 9$, we analyze carefully the structure of a CNF encoding.

1.4 Related work

Many results for various computational models with limited non-determinism are surveyed by Goldsmith, Levy, and Mundhenk [2]. An overview of known approaches for CNF encodings is given by Prestwich [10]. Two recent results that are close to the results of this paper are the following. Morizumi [7] proved that non-deterministic inputs do not help in the model of Boolean circuits over the U_2 basis (the set of all binary functions except for the binary parity and its complement) for computing the parity function: with and without non-deterministic inputs, the minimum size of a circuit computing parity is $3(n-1)$. Kucera, Savický, Vorel [5] prove almost tight bounds on the size of CNF encodings of the at-most-one Boolean function ($[x_1+\dots+x_n \le 1]$). Sinz [11] proves a linear lower bound on the size of CNF encodings of the at-most-k Boolean function.

2 General setting

2.1 Computing Boolean functions by CNFs

For a Boolean function $f(x_1,\dots,x_n)\colon \{0,1\}^n \to \{0,1\}$, we say that a CNF $F(x_1,\dots,x_n)$ *computes* f if $f \equiv F$, that is, for all $x_1,\dots,x_n \in \{0,1\}$, $f(x_1,\dots,x_n) = F(x_1,\dots,x_n)$. We treat a CNF as a set of clauses and by the *size* of a CNF we mean its number of clauses. It is well known that for every function f, there exists a CNF computing it. One way to construct such a CNF is the following: for every input $x \in \{0,1\}^n$ such that $f(x)=0$, populate a CNF with a clause of length n that is falsified by x.

This method does not guarantee that the produced CNF has the minimal number of clauses: this would be too good to be true as the problem of finding a CNF of minimum size for a given Boolean function (specified by its truth table) is NP-complete as proved by Masek [6] (see also [1] and references herein). For example, for a function $f(x_1,x_2)=x_1$ the method produces a CNF $(\overline{x_1} \lor x_2) \land (\overline{x_1} \lor \overline{x_2})$ whereas the function x_1 is already in CNF format.

2.2 Parity

It is well known that for many functions, the minimum size of a CNF is exponential. The canonical example is the parity function $\mathrm{PAR}_n(x_1,\dots,x_n) = x_1 \oplus \dots \oplus x_n$. The property of PAR_n that prevents it from being computable by short CNF's is its high *sensitivity*: by flipping *any* bit in *any* input $x \in \{0,1\}^n$, one flips the value of $\mathrm{PAR}_n(x)$.

▶ **Lemma 2.** *The minimum size of a CNF computing* PAR_n *has size* 2^{n-1}.

Proof. An upper bound follows from the method above by noting that $|\mathrm{PAR}_n^{-1}(0)| = 2^{n-1}$.

A lower bound is based on the fact that any clause of a CNF F computing PAR_n must contain all variables $x_1,\dots,x_n$. Indeed, if a clause $C \in F$ did not depend on x_i, one could find an input $x \in \{0,1\}^n$ that falsifies C (hence, $F(x) = \mathrm{PAR}_n(x) = 0$) and remains to be falsifying even after flipping x_i. As any clause of F has exactly n variables, it rejects exactly one $x \in \{0,1\}^n$. Hence, F must contain at least $|\mathrm{PAR}_n^{-1}(0)| = 2^{n-1}$ clauses. ◀

2.3 Encoding Boolean functions by CNFs

We say that a CNF F *encodes* a Boolean function $f(x_1,\dots,x_n)$ if the following two conditions hold.

1. In addition to the input bits $x_1, \dots, x_n$, F also depends on s bits $y_1, \dots, y_s$ called *guess inputs* or *non-deterministic inputs*.
2. For every $x \in \{0,1\}^n$, $f(x) = 1$ iff there exists $y \in \{0,1\}^s$ such that $F(x,y) = 1$. In other words, for every $x \in \{0,1\}^n$,

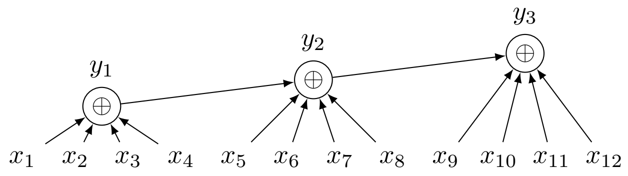
$$f(x) = \bigvee_{y \in \{0,1\}^s} F(x,y)\,. \tag{5}$$

Such representations of Boolean functions are widely used in practice when one translates a problem to SAT. For example, the following CNF encodes PAR$_4$:

$$(x_1 \vee x_2 \vee \overline{y_1}) \wedge (x_1 \vee \overline{x_2} \vee y_1) \wedge (\overline{x_1} \vee x_2 \vee y_1) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{y_1}) \wedge (y_1 \vee x_3 \vee \overline{y_2}) \wedge \\ (y_1 \vee \overline{x_3} \vee y_2) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge (\overline{y_1} \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{x_4} \vee y_2) \wedge (x_4 \vee \overline{y_2})\,. \tag{6}$$

2.4 Boolean Circuits and Tseitin Transformation

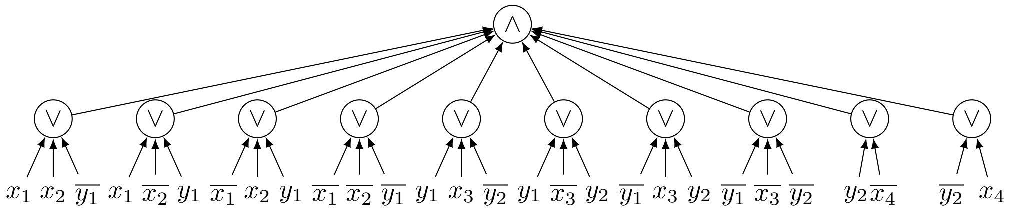
A natural way to get a CNF encoding of a Boolean function f is to take a circuit computing f and apply Tseitin transformation [12]. We describe this transformation using a toy example. The following circuit computes PAR$_{12}$ with three gates. It has 12 inputs, 3 gates (one of which is an output gate), and has depth 3.


y_1
y_2
y_3
$\oplus$
$\oplus$
$\oplus$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}

$$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_2 = y_1 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8$$
$$y_3 = y_2 \oplus x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12}$$

To the right of the circuit, we show the functions computed by each gate. One can translate each line into CNF. Adding a clause (y_3) to the resulting CNF gives a CNF encoding of the function computed by the circuit. In fact, the CNF (1) can be obtained this way (after propagating the value of the output gate).

A CNF can be viewed as a depth-2 circuit where the output gate is an AND, all other gates are ORs, and the inputs are variables and their negations. For example, the following circuit corresponds to a CNF (6). Such depth-2 circuits are also denoted as AND $\circ$ OR circuits.


$\wedge$
$\vee$ $\vee$ $\vee$ $\vee$ $\vee$ $\vee$ $\vee$ $\vee$ $\vee$ $\vee$
x_1 x_2 $\overline{y_1}$ x_1 $\overline{x_2}$ y_1 $\overline{x_1}$ x_2 y_1 $\overline{x_1}$ $\overline{x_2}$ $\overline{y_1}$ y_1 x_3 $\overline{y_2}$ y_1 $\overline{x_3}$ y_2 $\overline{y_1}$ x_3 y_2 $\overline{y_1}$ $\overline{x_3}$ $\overline{y_2}$ y_2 $\overline{x_4}$ $\overline{y_2}$ x_4

2.5 Depth-3 circuits

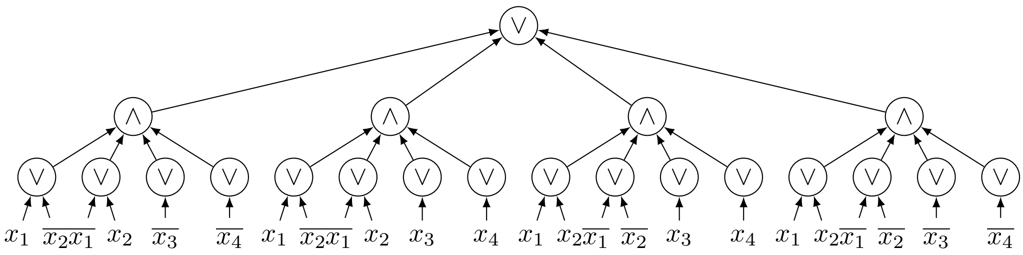
Depth-3 circuits is a natural generalization of CNFs: a Σ_3-*circuit* is simply an OR of CNFs. In a circuit, these CNFs are allowed to share clauses. A Σ_3-*formula* is a Σ_3-circuit whose CNFs do not share clauses (in other words, it is a circuit where the out-degree of every gate is equal to one).

On the one hand, this computation model is still simple enough. On the other hand, proving lower bounds against this model is much harder: getting a $2^{\omega(n)}$ lower bound for an explicit function (say, from NP or E^{NP}) is a major challenge. Proving a lower bound $2^{\omega(n/\log\log n)}$ would resolve another open question, through Valiant's depth reduction [13]: proving a superlinear lower bound on the size of logarithmic depth circuits. We refer the reader to Jukna's book [4, Chapter 11] for an exposition of known results for depth-3 circuits. For the parity function, the best known lower bound on depth-3 circuits is $\Omega(2^{\sqrt{n}})$ [8]. If one additionally requires that a circuit is a formula, i.e., that every gate has out-degree at most 1, then the best lower bound is $\Omega(2^{2\sqrt{n}})$ [3]. Both lower bounds are tight up to polynomial factors.

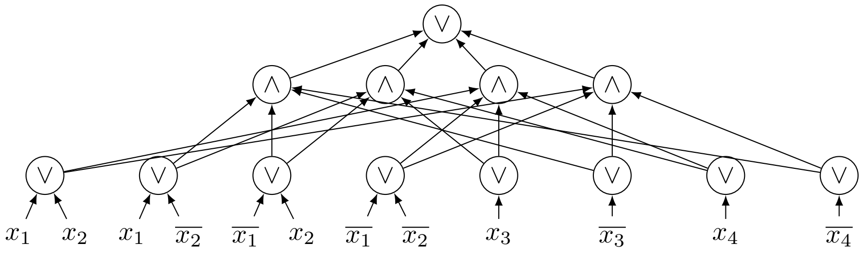
Equation (5) shows a tight connection between CNF encodings and depth-3 circuits of type $\mathrm{OR} \circ \mathrm{AND} \circ \mathrm{OR}$. Namely, let $F(x_1, \dots, x_n, y_1, \dots, y_s) = \{C_1, \dots, C_m\}$ be a CNF encoding of a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$. Then, $f(x) = \vee_{y \in \{0,1\}^s} F(x, y)$. By assigning y's in all 2^s ways, one gets an Σ_3-formula that computes f:

$$f(x) = \bigvee_{j \in [2^s]} F_j(x)\,, \tag{7}$$

where each F_j is a CNF. We call this an *expansion* of F. For example, an expansion of a CNF (6) looks as follows. It is an OR of four CNFs.


x_1 $\overline{x_2}$ $\overline{x_1}$ x_2 $\overline{x_3}$ $\overline{x_4}$ x_1 $\overline{x_2}$ $\overline{x_1}$ x_2 x_3 x_4 x_1 x_2 $\overline{x_1}$ $\overline{x_2}$ x_3 x_4 x_1 x_2 $\overline{x_1}$ $\overline{x_2}$ $\overline{x_3}$ $\overline{x_4}$

An expansion is a formula: it is an OR of CNFs, every gate has out-degree one. One can also get a *circuit-expansion*: in this case, gates are allowed to have out-degree more than one; alternatively, CNFs are allowed to share clauses. For example, this is a circuit-expansion of (6).


x_1 x_2 x_1 $\overline{x_2}$ $\overline{x_1}$ x_2 $\overline{x_1}$ $\overline{x_2}$ x_3 $\overline{x_3}$ x_4 $\overline{x_4}$

Below, we show that CNF encodings and depth-3 circuits can be easily transformed one into the other. It will prove convenient to define the size of a circuit as its number of gates *excluding* the output gate. This way, the size of a CNF formula equals its number of clauses (a CNF is a depth-2 formula). By a $\Sigma_3(t, r)$-circuit we denote a Σ_3-circuit having at most t ANDs on the second layer and at most r ORs on the third layer (hence, its size is at most $t + r$).

▶ **Lemma 3.** *Let $F(x_1, \dots, x_n, y_1, \dots, y_s)$ be a CNF encoding of size m of a function $f\colon \{0,1\}^n \to \{0,1\}$. Then, f can be computed by a $\Sigma_3(2^s, m \cdot 2^s)$-formula and by a $\Sigma_3(2^s, m)$-circuit.*

Proof. Let $F = \{C_1, \dots, C_m\}$. To expand F as $\bigvee_{j \in [2^s]} F_j$, we go through all 2^s assignments to non-deterministic variables $y_1, \dots, y_s$. Under any such assignment, each clause C_i is either satisfied or becomes a clause $C'_i \subseteq C_i$ resulting from C_i by removing all its non-deterministic variables. Thus, for each $j \in [2^s]$, $F_j \subseteq \{C'_1, \dots, C'_m\}$. The corresponding Σ_3-formula contains at most $2^s + m2^s$ gates: there are 2^s gates for F_j's, each F_j contains no more than m clauses. The corresponding Σ_3-circuit contains no more than $2^s + m$ gates: there are 2^s gates for F_j's and m gates for $C'_1, \dots, C'_m$ (each F_j selects which of these m clauses to contain). ◀

Interestingly, the upper bounds on depth-3 circuits resulting from this simple transformation cannot be substantially improved. Indeed, by plugging in a CNF encoding of PAR_n with $s = \sqrt{n}$ and $m = O(\sqrt{n}2^{\sqrt{n}})$ (see (1)), one gets a Σ_3-formula and a Σ_3-circuit of size $2^{2\sqrt{n}}$ and $2^{\sqrt{n}}$, respectively, up to polynomial factors. As discussed above, these bounds are known to be optimal.

Below, we show a converse transformation.

▶ **Lemma 4.** *Let C be a $\Sigma_3(t, r)$-formula (circuit) computing a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$. Then, f can be encoded as a CNF with $\lceil \log t \rceil$ non-deterministic variables of size r ($2rt$, respectively).*

Proof. Let $C = F_1 \vee \dots \vee F_t$ be a Σ_3-formula (hence, $r = \mathrm{size}(F_1) + \dots + \mathrm{size}(F_t)$). Introduce $s = \lceil \log t \rceil$ non-deterministic variables $y_1, \dots, y_s$. Then, for every assignment to $y_1, \dots, y_s$, take the corresponding CNF F_i ($1 \le i \le 2^s$ is the unique integer corresponding to this assignment) and add y_i's with the corresponding signs to every clause of F_i. Call the resulting CNF F'_i. Then, $F = F'_1 \wedge \dots \wedge F'_{2^s}$ encodes f and F has at most r clauses.

If C is a Σ_3-circuit, we need to create a separate copy of every gate corresponding to a clause in each of 2^s CNFs. Hence, the size of the resulting CNF encoding is at most $r2^s \le 2rt$. ◀

Finally, we show that proving strong lower bounds on the size of CNF encodings is not easier than proving strong lower bounds on the size of depth-3 circuits. Let C be a $\Sigma_3(t, r)$-formula computing PAR_n. Lemma 4 guarantees that PAR_n can be encoded as a CNF of size r with $\lceil \log t \rceil$ non-deterministic variables. Then, by the inequality (2),

$$\mathrm{size}(C) = t + r \ge t + \Omega\left(\frac{1}{n} \cdot 2^{\frac{n}{\log t + 2}}\right) \ge \frac{1}{n}\left(t + \Omega\left(2^{\frac{n}{\log t + 2}}\right)\right) \ge \Omega\left(\frac{2^{\sqrt{n}}}{n}\right) .$$

Similarly, if C is a $\Sigma_3(t, r)$-circuit, Lemma 4 guarantees that PAR_n can be encoded as a CNF of size $2rt$ with $\lceil \log t \rceil$ non-deterministic variables. Then,

$$\mathrm{size}(C) = t + r \ge t + \Omega\left(\frac{1}{2tn} \cdot 2^{\frac{n}{\log t + 2}}\right) \ge \Omega\left(\frac{2^{\sqrt{n/2}}}{n}\right) .$$

3 Lower bounds for CNF encodings of parity

In this section, we prove Theorem 1. The essential property of the parity function used in the proof is its high sensitivity (every satisfying assignment is isolated): for any $i \in [n]$ and any $x, x' \in \{0,1\}^n$ that differ in the i-th position only, $\mathrm{PAR}(x) \neq \mathrm{PAR}(x')$. This means that if a CNF F computes PAR and $F(x) = 1$, then F must contain a clause that is satisfied by x_i only. Following [8], we call such a clause *critical* with respect to (x, i). This notion extends to CNF encodings in a natural way. Namely, let $F(x, y)$ be a CNF encoding of PAR. Then, for any (x, y) such that $F(x, y) = 1$ and any $i \in [n]$, F contains a clause that becomes falsified if one flips the bit x_i. We call it critical w.r.t. (x, y, i).

3.1 Limited non-determinism

To prove a lower bound $m \geq \Omega((s+1)2^{n/(s+1)}/n)$, we adapt a proof of the $\Omega(n^{1/4}2^{\sqrt{n}})$ lower bound for depth-3 circuits computing PAR_n by Paturi, Pudlák, and Zane [8]. Let $F(x_1, \ldots, x_n)$ be a CNF. For every isolated satisfying assignment $x \in \{0,1\}^n$ of F and every $i \in [n]$, fix a shortest critical clause w.r.t. (x, i) and denote it by $C_{F,x,i}$. Then, for an isolated satisfying assignment x, define its weight w.r.t. F as

$$w_F(x) = \sum_{i=1}^{n} \frac{1}{|C_{F,x,i}|} \,.$$

▶ **Lemma 5** (Lemma 5 in [8]). *For any μ, F has at most $2^{n-\mu}$ isolated satisfying assignments of weight at least μ.*

Proof of (2), $m \geq \Omega\left(\frac{s+1}{n} \cdot 2^{n/(s+1)}\right)$**.** Let $F(x_1, \ldots, x_n, y_1, \ldots, y_s)$ be a CNF encoding of size m of PAR_n. Consider its expansion:

$$\mathrm{PAR}_n(x) = \bigvee_{j \in [2^s]} F_j(x) \,.$$

We extend the definitions of $C_{F,x,i}$ and $w(x)$ to CNFs with non-deterministic variables as follows. Let $x \in \mathrm{PAR}_n^{-1}(1)$ and let $j \in [2^s]$ be the smallest index such that $F_j(x) = 1$. For $i \in [n]$, let $C'_{F,x,i} = C_{F_j,x,i}$ (that is, we simply take the first F_j that is satisfied by x and take its critical clause w.r.t. (x, i)). Then, the weight $w'_F(x)$ of x w.r.t. to F is defined simply as $w_{F_j}(x)$. Clearly,

$$w'_F(x) = \sum_{i \in [n]} \frac{1}{|C'_{(F,x,i)}|} \,.$$

For $l \in [n]$, let also $N_{l,F}(x) = |\{i \in [n] \colon |C'_{F,x,i}| = l\}|$ be the number of critical clauses (w.r.t. x) of length l. Clearly,

$$w'_F(x) = \sum_{l \in [n]} \frac{N_{l,F}(x)}{l} \,. \tag{8}$$

For a parameter $0 < \varepsilon < 1$ to be chosen later, split $\mathrm{PAR}_n^{-1}(1)$ into light and heavy parts:

$$H = \{x \in \mathrm{PAR}_n^{-1}(1) \colon w'_F(x) \geq s + 1 + \varepsilon\} \,,$$
$$L = \{x \in \mathrm{PAR}_n^{-1}(1) \colon w'_F(x) < s + 1 + \varepsilon\} \,.$$

We claim that

$$|H| \leq 2^s \cdot 2^{n-s-1-\varepsilon} \,.$$

Indeed, for every $x \in H$, $w'_F(x) = w_{F_j}(x)$ for some $j \in [2^s]$, and by Lemma 5, F_j cannot accept more than $2^{n-s-1-\varepsilon}$ isolated solutions of weight at least $s + 1 + \varepsilon$. Since $|H| + |L| = |\mathrm{PAR}_n^{-1}(1)| = 2^{n-1}$, we conclude that

$$|L| = 2^{n-1} - |H| \geq (1 - 2^{-\varepsilon})2^{n-1} \,. \tag{9}$$

Let $F = \{C_1, \dots, C_m\}$. For every $k \in [m]$, let $C'_k \subseteq C_k$ be the clause C_k with all non-deterministic variables removed. Hence, for every $j \in [2^s]$, $F_j \subseteq \{C'_1, \dots, C'_m\}$. For $l \in [n]$, let $m_l = |\{k \in [m] \colon |C'_k| = l\}|$ be the number of such clauses of length l. Consider a clause C'_k and let $l = |C'_k|$. Then, there are at most $l2^{n-l}$ pairs (x, i), where $x \in \mathrm{PAR}^{-1}(1)$ and $i \in [n]$, such that $C'_{F,x,i} = C'_k$: there are at most l choices for i, fixing i fixes the values of all l literals in C'_k (all of them are equal to zero except for the i-th one), and there are no more than 2^{n-l} choices for the other bits of x. Recall that $N_{l,F}(x)$ is the number of critical clauses w.r.t. x of length l. Thus, we arrive at the following inequality:

$$m_l \cdot l \cdot 2^{n-l} \geq \sum_{x \in \mathrm{PAR}^{-1}(1)} N_{F,l}(x) \geq \sum_{x \in L} N_{F,l}(x) \,.$$

Then,

$$m = \sum_{l \in [n]} m_l \geq \sum_{l \in [n]} \frac{\sum_{x \in L} N_{F,l}(x)}{l2^{n-l}} = \sum_{x \in L} \sum_{l \in [n]} \frac{N_{F,l}(x)}{l2^{n-l}} = \sum_{x \in L} n2^{-n} \sum_{l \in [n]} \frac{N_{F,l}(x)}{n} \cdot \frac{2^l}{l} \,. \tag{10}$$

To estimate the last sum, let

$$T(x) = \sum_{l \in [n]} \frac{N_{F,l}(x)}{n} \cdot \frac{2^l}{l} = \sum_{l \in [n]} \frac{N_{F,l}(x)}{n} \cdot g(l) \,,$$

where $g(l) = \frac{2^l}{l}$. Since $g(l)$ is convex (for $l > 0$) and $\sum_{l \in [n]} \frac{N_{F,l}(x)}{n} = 1$, Jensen's inequality gives

$$T(x) \geq g\left(\sum_{l \in [n]} \frac{N_{F,l}(x)}{n} \cdot l \right) \,. \tag{11}$$

Further, Sedrakyan's inequality[1] (combined with (8) and $\sum_{l \in [n]} N_{F,l}(x) = n$) gives

$$\sum_{l \in [n]} l N_{F,l}(x) = \sum_{l \in [n]} \frac{N^2_{F,l}(x)}{N_{F,l}(x)/l} \geq \frac{\left(\sum_{l \in [n]} N_{F,l}(x)\right)^2}{\sum_{l \in [n]} N_{F,l}(x)/l} = \frac{n^2}{w'_F(x)} \,. \tag{12}$$

Since $g(l)$ is monotonically increasing for $l \geq 1/\ln 2$ and $w'_F(x) < s + 1 + \varepsilon$ for every $x \in L$, combining (11) and (12), we get

$$T(x) \geq g\left(\frac{n}{w'_F(x)}\right) \geq g\left(\frac{n}{s+1+\varepsilon}\right) \,, \tag{13}$$

for $s \leq n \ln 2 - 1 - \varepsilon$. (If $s > n \ln 2 - 1 - \varepsilon$, then the lower bound $m \geq \Omega(2^{n/(s+1)}/n)$ is trivial.)

[1] Sedrakyan's inequality is a special case of Cauchy–Schwarz inequality: for all $a_1, \dots, a_n \in \mathbb{R}$ and $b_1, \dots, b_n \in \mathbb{R}_{>0}$, $\sum_{i=1}^n a_i^2/b_i \geq \left(\sum_{i=1}^n a_i\right)^2 / \sum_{i=1}^n b_i$.

Thus,

$$\begin{aligned}
m &\geq \sum_{x \in L} n2^{-n}T(x) \geq && \text{(10 and 13)}\\
&\geq \sum_{x \in L} n2^{-n} g\left(\frac{n}{s+1+\varepsilon}\right) = && \text{(definition of } g\text{)}\\
&= |L|2^{-n}2^{\frac{n}{s+1+\varepsilon}}(s+1+\varepsilon) \geq && \text{(9)}\\
&\geq \left(\frac{1}{2} - \frac{1}{2^{\varepsilon+1}}\right)(s+1+\varepsilon)2^{\frac{n}{s+1+\varepsilon}} = && \text{(rewriting)}\\
&= \left(\frac{1}{2} - \frac{1}{2^{\varepsilon+1}}\right)(s+1+\varepsilon)2^{\frac{n}{s+1}}2^{\frac{-n\varepsilon}{(s+1)(s+1+\varepsilon)}} .
\end{aligned}$$

Set $\varepsilon = 1/n$. Then,

$$\left(\frac{1}{2} - \frac{1}{2^{\frac{1}{n}+1}}\right) = \Theta\left(\frac{1}{n}\right) .$$

Also,

$$\frac{1}{2} \leq 2^{\frac{-1}{(s+1)(s+1+1/n)}} \leq 1 ,$$

as $2^{-1/x}$ is increasing for $x > 0$. This finally gives a lower bound

$$m \geq \Omega\left(\frac{s+1}{n} \cdot 2^{\frac{n}{s+1}}\right) .$$ ◀

3.2 Width of clauses

To prove the lower bound $k \geq n/(s+1)$, we use the following corollary of the Satisfiability Coding Lemma.

▶ **Lemma 6** (Lemma 2 in [8]). *Any k-CNF $F(x_1, \ldots, x_n)$ has at most $2^{n-n/k}$ isolated satisfying assignments.*

Proof of (3), $k \geq n/(s+1)$**.** Consider a k-CNF $F(x_1, \ldots, x_n, y_1, \ldots, y_s)$ that encodes PAR_n. Expand F to an OR of 2^s k-CNFs:

$$\mathrm{PAR}_n(x) = \bigvee_{j \in [2^s]} F_j(x) .$$

By Lemma 6, each F_j accepts at most $2^{n-n/k}$ isolated solutions. Hence,

$$2^s \geq \frac{2^{n-1}}{2^{n-n/k}} = 2^{n/k-1}$$

and thus, $k \geq n/(s+1)$. ◀

3.3 Unlimited non-determinism

In this section, we prove the lower bound $m \geq 3n - 9$.

Proof of (4), $m \geq 3n - 9$. We use induction on n. The base case $n \leq 3$ is clear. To prove the induction step, assume that $n > 3$ and consider a CNF encoding $F(x_1, \ldots, x_n, y_1, \ldots, y_s)$ of PAR_n with the minimum number of clauses. Below, we show that one can find k deterministic variables (where $k = 1$ or $k = 2$) such that assigning appropriately chosen constants to them reduces the number of clauses by at least $3k$, respectively. The resulting function computes PAR_{n-k} or its negation. It is not difficult to see that the minimum number of clauses in encodings of PAR and its negation are equal (by flipping the signs of all occurrences of any deterministic variable in a CNF encoding of PAR, one gets a CNF encoding of the negation of PAR, and vice versa). Hence, one can proceed by induction and conclude that F contains at least $3(n-k) - 9 + 3k = 3n - 9$ clauses.

To find the required k deterministic variables, we go through a number of cases. In the analysis below, by a d-literal we mean a literal that appears exactly d times in F, a d^+-literal appears at least d times. A (d_1, d_2)-literal occurs d_1 times positively and d_2 times negatively. Other types of literals are defined similarly. We treat a clause as a set of literals (that do not contain a literal together with its negation) and a CNF formula as a set of clauses.

Note that for all $i \in [s]$, y_i must be a $(2^+, 2^+)$-literal. Indeed, if y_i (or $\overline{y_i}$) is a 0-literal, one can assign $y_i \leftarrow 0$ ($y_1 \leftarrow 1$, respectively). It is not difficult to see that the resulting formula still encodes PAR. If y_i is a $(1, t)$-literal, one can eliminate it using resolution: for all pairs of clauses $C_0, C_1 \in F$ such that $\overline{y_i} \in C_0$ and $y_i \in C_1$, add a clause $C_0 \cup C_1 \setminus \{y_i, \overline{y_i}\}$ (if this clause contains a pair of complementary literals, ignore it); then, remove all clauses containing y_i or $\overline{y_i}$. The resulting formula still encodes PAR_n, but has a smaller number of clauses than F (we remove $1 + t$ clauses and add at most t clauses).

In the case analysis below, by l_i we denote a literal that corresponds to a deterministic variable x_i or its negation $\overline{x_i}$.

1. *F contains a 3^+-literal l_i.* Assigning $l_i \leftarrow 1$ eliminates at least three clauses from F.
2. *F contains a 1-literal l_i.* Let $l_i \in C \in F$ be a clause containing l_i. C cannot contain other deterministic variables: if $l_i, l_j \in C$ (for $i \neq j \in [n]$), consider $x \in \{0,1\}^n$ such that $\mathrm{PAR}_n(x) = 1$ and $l_i = l_j = 1$ (such x exists since $n > 3$), and its extension $y \in \{0,1\}^s$ such that $F(x, y) = 1$; then, F does not contain a critical clause w.r.t. (x, y, i). Clearly, C cannot be a unit clause, hence it must contain a non-deterministic variable y_j. Consider $x \in \{0,1\}^n$, such that $\mathrm{PAR}_n(x) = 1$ and $l_i = 1$, and its extension $y \in \{0,1\}^s$ such that $F(x, y) = 1$. If $y_j = 1$, then F does not contain a critical clause w.r.t. (x, y, i). Thus, for every $(x, y) \in \{0,1\}^{n+s}$ such that $F(x, y) = 1$ and $l_i = 1$, it holds that $y_j = 0$. This observation allows us to proceed as follows: first assign $l_i \leftarrow 1$, then assign $y_j \leftarrow 0$. The former assignment satisfies the clause C, the latter one satisfies all the clauses containing $\overline{y_j}$. Thus, at least three clauses are removed.
3. *For all $i \in [n]$, x_i is a $(2, 2)$-literal.* If there is no clause in F containing at least two deterministic variables, then F contains at least $4n$ clauses and there is nothing to prove. Let $l_i, l_j \in C_1 \in F$, where $i \neq j$, be a clause containing two deterministic variables and let $l_i \in C_2 \in F$ and $l_j \in C_3 \in F$ be the two clauses containing other occurrences of l_i and l_j ($C_1 \neq C_2$ and $C_1 \neq C_3$, but it can be the case that $C_2 = C_3$).

   Assume that C_2 contains another deterministic variable: $l_k \in C_2$, where $k \neq i, j$. Consider $x \in \{0,1\}^n$, such that $\mathrm{PAR}_n(x) = 1$ and $l_i = l_j = l_k = 1$ (such x exists since $n > 3$), and its extension $y \in \{0,1\}^s$ such that $F(x, y) = 1$. Then, F does not contain a critical clause w.r.t. (x, y, i): C_1 is satisfied by l_j, C_2 is satisfied by l_k. For the same reason, C_2 cannot contain the literal l_j. Similarly, C_3 cannot contain other deterministic variables and the literal l_i. (At the same time, it is not excluded that $\overline{l_j} \in C_2$ or $\overline{l_i} \in C_3$.) Hence, $C_2 \neq C_3$. Note that each of C_2 and C_3 must contain at least one non-deterministic variable: otherwise, it would be possible to falsify F by assigning l_i and l_j.

a. *At least one of C_2 and C_3 contains a single non-deterministic variable.* Assume that it is C_2:

$$\{l_i, y_1\} \subseteq C_2 \subseteq \{l_i, \overline{l_j}, y_1\}\,.$$

Assign $l_j \leftarrow 1$. This eliminates two clauses: C_1 and C_3 are satisfied. Also, under this substitution, $C_2 = \{l_i, y_1\}$ and l_i is a 1-literal. We claim that in any satisfying assignment of the resulting formula F', $l_i = \overline{y_1}$. Indeed, if (x, y) satisfies F' and $l_i = y_1$, then $l_i = y_1 = 1$ (otherwise C_2 is falsified). But then there is no critical clause in F' w.r.t. (x, y, i). Since in every satisfying assignment $l_i = \overline{y_1}$, we can replace every occurrence of y_1 ($\overline{y_1}$) by $\overline{l_i}$ (y_1, respectively). This, in particular, satisfies the clause C_2.

b. *Both C_2 and C_3 contain at least two non-deterministic variables:*

$$\{l_i,\ l_j\} \subseteq C_1, \quad \{l_i,\ y_1,\ y_2\} \subseteq C_2, \quad \{l_j,\ y_3,\ y_4\} \subseteq C_3\,.$$

Here, y_1 and y_2 are different variables, y_3 and y_4 are also different, though it is not excluded that some of y_1 and y_2 coincide with some of y_3 and y_4. Let $Y \subseteq \{y_1, \ldots, y_s\}$ be non-deterministic variables appearing in C_2 or C_3.

Recall that for every $(x, y) \in \{0,1\}^{n+s}$ such that $F(x, y) = 1$ and $l_i = l_j = 1$, it holds that $y = 0$ for all $y \in Y$. This means that if a variable $y \in Y$ appears in both C_2 and C_3, then it has the same sign in both clauses. Consider two subcases.

i. $Y = \{y_1, y_2\}$:

$$\{l_i,\ l_j\} \subseteq C_1, \quad \{l_i,\ y_1,\ y_2\} \subseteq C_2, \quad \{l_j,\ y_1,\ y_2\} \subseteq C_3\,.$$

Assume that $\overline{y_1} \notin C_1$. Assign $l_i \leftarrow 1$, $l_j \leftarrow 1$. Then, assigning $y_1 \leftarrow 0$ eliminates at least two clauses. Let us show that there remains a clause that contains $\overline{y_2}$. Consider $x \in \mathrm{PAR}_n^{-1}(1)$, such that $l_i = l_j = 1$, and its extension $y \in \{0,1\}^s$, such $F(x, y) = 1$. We know that y_1 and y_2 must be equal to 0. However, flipping the value of y_2 results in a satisfying assignment. Thus, it remains to analyze the following case:

$$\{l_i,\ l_j, \overline{y_1}, \overline{y_2}\} \subseteq C_1, \quad \{l_i,\ y_1,\ y_2\} \subseteq C_2, \quad \{l_j,\ y_1,\ y_2\} \subseteq C_3\,.$$

Assume that $\overline{l_j} \notin C_2$ and $\overline{l_i} \notin C_1$. Assign $l_i \leftarrow 1$, then assign $y_1 \leftarrow 0$ and $y_2 \leftarrow 0$. Under this assignment, $C_3 = \{l_j\}$ (recall that C_3 cannot contain other deterministic variables, see Case 3). This would mean that $l_j = 1$ in every satisfying assignment of the resulting CNF formula which cannot be the case for a CNF encoding of parity. Thus, we may assume that either $\overline{l_j} \in C_2$ or $\overline{l_i} \in C_1$. Without loss of generality, assume that $\overline{l_j} \in C_2$.

Let us show that for every $(x, y) \in \{0,1\}^{n+s}$, such that $F(x, y) = 1$ and $l_i = 1$, it holds that $l_j \neq y_1$ and $l_j \neq y_2$. Indeed, if there is $(x, y) \in \{0,1\}^{n+s}$ such that $F(x, y) = 1$ and $l_i = l_j = 1$, then y_1 and y_2 must be equal to 0. If there is $(x, y) \in \{0,1\}^{n+s}$, such that $F(x, y) = 1, l_i = 1, l_j = 0$, then y_1 and y_2 must be equal to 0, otherwise F does not contain a critical clause w.r.t. (x, y, i). Thus, assigning $l_i \leftarrow 1$ eliminates two clauses (C_1 and C_2). We then replace y_1 and y_2 with $\overline{l_j}$ and delete the clause C_3.

ii. $|Y| \geq 3, \{y_1, y_2, y_3\} \subseteq Y$:

$$\{l_i,\ l_j\} \subseteq C_1, \quad \{l_i,\ y_1,\ y_2\} \subseteq C_2, \quad \{l_j,\ y_1,\ y_3\} \subseteq C_3\,.$$

Assigning $l_i \leftarrow 1, l_j \leftarrow 1$ eliminates C_1, C_2, C_3. Assigning $y_1 \leftarrow 0$ eliminates at least one more clause (y_1 appears positively at least two times, but it may appear in C_1). There must be a clause with $\overline{y_2}$ (otherwise we could assign $y_2 \leftarrow 1$). Assigning $y_2 \leftarrow 0$ eliminates at least one more clause. Similarly, assigning $y_3 \leftarrow 1$ eliminates another clause. In total, we eliminate at least six clauses. ◀

References

1 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

2 Judy Goldsmith, Matthew A. Levy, and Martin Mundhenk. Limited nondeterminism. *SIGACT News*, 27(2):20–29, 1996. doi:10.1145/235767.235769.

3 Shuichi Hirahara. A duality between depth-three formulas and approximation by depth-two. *Electron. Colloquium Comput. Complex.*, page 92, 2017. URL: https://eccc.weizmann.ac.il/report/2017/092.

4 Stasys Jukna. *Boolean Function Complexity – Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

5 Petr Kucera, Petr Savický, and Vojtech Vorel. A lower bound on CNF encodings of the at-most-one constraint. *Theor. Comput. Sci.*, 762:51–73, 2019. doi:10.1016/j.tcs.2018.09.003.

6 William J. Masek. Some NP-complete set covering problems. Unpublished Manuscript, 1979.

7 Hiroki Morizumi. Lower bounds for the size of nondeterministic circuits. In Dachuan Xu, Donglei Du, and Ding-Zhu Du, editors, *Computing and Combinatorics – 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, volume 9198 of *Lecture Notes in Computer Science*, pages 289–296. Springer, 2015. doi:10.1007/978-3-319-21398-9_23.

8 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chic. J. Theor. Comput. Sci.*, 1999, 1999. URL: http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html.

9 Steven David Prestwich. SAT problems with chains of dependent variables. *Discret. Appl. Math.*, 130(2):329–350, 2003. doi:10.1016/S0166-218X(02)00410-9.

10 Steven David Prestwich. CNF encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 75–97. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-75.

11 Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming – CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005. doi:10.1007/11564751_73.

12 G. S. Tsejtin. On the complexity of derivation in propositional calculus. Semin. Math., V. A. Steklov Math. Inst., Leningrad 8, 115-125 (1970); translation from Zap. Nauchn. Semin. Leningr. Otd. Mat. Inst. Steklova 8, 234-259 (1968)., 1968.

13 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977. doi:10.1007/3-540-08353-7_135.

On the Number of Quantifiers as a Complexity Measure

Ronald Fagin
IBM Research, Almaden, CA, USA

Jonathan Lenchner[1]
IBM T.J. Watson Research Center, Cambridge, MA, USA

Nikhil Vyas
MIT EECS, Cambridge, MA, USA

Ryan Williams
MIT CSAIL and EECS, Cambridge, MA, USA

Abstract

In 1981, Neil Immerman described a two-player game, which he called the "separability game" [15], that captures the number of quantifiers needed to describe a property in first-order logic. Immerman's paper laid the groundwork for studying the number of quantifiers needed to express properties in first-order logic, but the game seemed to be too complicated to study, and the arguments of the paper almost exclusively used quantifier rank as a lower bound on the total number of quantifiers. However, last year Fagin, Lenchner, Regan and Vyas [10] rediscovered the game, provided some tools for analyzing them, and showed how to utilize them to characterize the number of quantifiers needed to express linear orders of different sizes. In this paper, we push forward in the study of number of quantifiers as a bona fide complexity measure by establishing several new results. First we carefully distinguish minimum number of quantifiers from the more usual descriptive complexity measures, minimum quantifier rank and minimum number of variables. Then, for each positive integer k, we give an explicit example of a property of finite structures (in particular, of finite graphs) that can be expressed with a sentence of quantifier rank k, but where the same property needs $2^{\Omega(k^2)}$ quantifiers to be expressed. We next give the precise number of quantifiers needed to distinguish two rooted trees of different depths. Finally, we give a new upper bound on the number of quantifiers needed to express s-t connectivity, improving the previous known bound by a constant factor.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases number of quantifiers, multi-structural games, complexity measure, s-t connectivity, trees, rooted trees

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.48

Related Version *Extended Version*: `https://arxiv.org/abs/2207.00104` [11]

1 Introduction

In 1981 Neil Immerman described a two-player combinatorial game, which he called the "separability game" [15], that captures the number of quantifiers needed to describe a property in first-order logic (henceforth FOL). In that paper Immerman remarked,

> "Little is known about how to play the separability game. We leave it here as a jumping off point for further research. We urge others to study it, hoping that the separability game may become a viable tool for ascertaining some of the lower bounds which are 'well believed' but have so far escaped proof."

[1] Corresponding author

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 48; pp. 48:1–48:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Immerman's paper laid the groundwork for studying the number of quantifiers needed to express properties in FOL, but alas, the game seemed too complicated to study and the paper used the surrogate measure of quantifier rank, which provides a lower bound on the number of quantifiers, to make its arguments. One of the reasons for the difficulty of directly analyzing the number of quantifiers is that the separability game is played on a pair $(\mathcal{A}, \mathcal{B})$ of *sets* of structures, rather than on a pair of structures as in a conventional Ehrenfeucht-Fraïssé game. However, last year Fagin, Lenchner, Regan and Vyas [10] rediscovered the games, provided some tools for analyzing them, and showed how to utilize them to characterize the number of quantifiers needed to express linear orders of different sizes. In this paper, we push forward in the study of number of quantifiers as a bona fide complexity measure by establishing several new results, using these rediscovered games as an important, though not exclusive, tool. Although Immerman called his game the "separability game," we keep to the more evocative "multi-structural game," as coined in [10].

Given a property P definable in FOL, let $\mathit{Quants}(P)$ denote the minimum number of quantifiers over all FO sentences that express P. **This paper exclusively considers expressibility in FOL.** $\mathit{Quants}(P)$ is related to two more widely studied descriptive complexity measures, the minimum quantifier rank needed to express P, and the minimum number of variables needed to express P. The quantifier rank of an FO sentence σ is typically denoted by $qr(\sigma)$. We shall denote the *minimum* quantifier rank over *all* FO sentences describing the property P by $\mathit{Rank}(P)$, and denote the minimum number of variables needed to describe P by $Vars(P)$. When referring to a specific sentence σ, we shall denote the analogs of $\mathit{Quants}()$ and $Vars()$ by $\mathit{quants}(\sigma)$ and $\mathit{vars}(\sigma)$. (That is, $\mathit{quants}(\sigma), vars(\sigma)$ and $qr(\sigma)$ refer to the number of quantifiers, variables and quantifier rank of the particular sentence σ.) On the other hand, $\mathit{Quants}(P), Vars(P)$ and $\mathit{Rank}(P)$ refer to the minimum values of these quantities among all expressions describing P. Possibly there is one sentence establishing $\mathit{Quants}(P)$, another establishing $Vars(P)$, and a third establishing $\mathit{Rank}(P)$. We investigate the *extremal* behavior of $\mathit{Quants}(P)$, via studying concrete properties P for which $\mathit{Quants}(P)$ behaves differently from the other measures.

First of all, for every property P, since every variable in a sentence describing P is bound to a quantifier, and quantifiers can only be bound to a single variable, it must be that $Vars(P) \leq \mathit{Quants}(P)$. The following simple proposition observes that $Vars(P)$ is also upper bounded by $\mathit{Rank}(P)$.

▶ **Proposition 1.** *For every property P: $Vars(P) \leq \mathit{Rank}(P)$.*

A proof is in [11, Prop. 1]. Since clearly $\mathit{Rank}(P) \leq \mathit{Quants}(P)$, we have:

$$Vars(P) \leq \mathit{Rank}(P) \leq \mathit{Quants}(P). \tag{1}$$

Furthermore, it follows from Immerman [16, Prop. 6.15] that $\mathit{Quants}(P)$ and $\mathit{Rank}(P)$ can both be arbitrarily larger than $Vars(P)$. When the property P is *s-t* connectivity up to path length k, Immerman shows that $Vars(P) \leq 3$, yet $\mathit{Rank}(P) \geq \log_2(k)$.

Summary of Results

From equation (1), we see that the number of quantifiers needed to express a property is lower-bounded by the minimum quantifier rank and number of variables. How much larger can $\mathit{Quants}(P)$ be, compared to the other two measures? It is known (see [8]) that there *exists* a fixed vocabulary V and an infinite sequence $P_1, P_2, \ldots$ of properties such that P_k is a property of finite structures with vocabulary V such that $\mathit{Rank}(P_k) \leq k$, yet $\mathit{Quants}(P_k)$ is not an elementary function of k. However, the existence of such P_k are proved via counting arguments. We provide an *explicitly computable* sequence of properties $\{P_k\}$ with a high growth rate in terms of the number of quantifiers required. (By "explicitly computable", we mean that there is an algorithm A such that, given a positive integer k, the algorithm A prints a FO sentence σ_k with quantifier rank k defining the property P_k, in time polynomial in the length of σ_k.)

▶ **Theorem** (Theorem 4, Section 2). *There is an explicitly computable sequence of properties $\{P_k\}$ such that for all k we have $Rank(P_k) \leq k$, yet $Quants(P_k) \geq 2^{\Omega(k^2)}$.*

Next, we give an example of a setting in which one can completely nail down the number of quantifiers that are necessary and sufficient for expressing a property. Building on Fagin *et al.* [10], which gives results on the number of quantifiers needed to distinguish linear orders of different sizes, we study the number of quantifiers needed to distinguish rooted trees of different depths.

Let $t(r)$ be the maximum d such that there is a formula with r quantifiers that can distinguish rooted trees of depth d (or larger) from rooted trees of depth less than d. Reasoning about the relevant multi-structural games, we can completely characterize $t(r)$, as follows.

▶ **Theorem** (Theorem 14, Section 3). *For all $r \geq 1$ we have*

$$t(2r) = \frac{7 \cdot 4^r}{18} + \frac{4r}{3} - \frac{8}{9}, \qquad t(2r+1) = \frac{8 \cdot 4^r}{9} + \frac{4r}{3} - \frac{8}{9}.$$

It follows from the above theorem that we can distinguish (rooted) trees of depth at most d from trees of depth greater than d using only $\Theta(\log d)$ quantifiers, and we can in fact pin down the *exact* depth that can be distinguished with r quantifiers. This illustrates the power of multi-structural games, and gives hope that more complex problems may admit an exact number-of-quantifiers characterization.

Next, we consider the question of how many quantifiers are needed to express that two nodes s and t are connected by a path of length at most n, in directed (or undirected) graphs. In our notation, we wish to determine $Quants(P)$ where P is the property of s-t connectivity via a path of length at most n. Considering the significance of s-t connectivity in both descriptive complexity and computational complexity, we believe this is a basic question that deserves a clean answer. It follows from the work of Stockmeyer and Meyer that s-t connectivity up to path length n can be expressed with $3\log_2(n) + O(1)$ quantifiers. As mentioned earlier, s-t connectivity is well-known to require quantifier rank at least $\log_2(n) - O(1)$. We manage to reduce the number of quantifiers necessary for s-t connectivity.

▶ **Theorem** (Theorem 15, Section 4). *The number of quantifiers needed to express s-t connectivity is at most $3\log_3(n) + O(1) \approx 1.893\log_2(n) + O(1)$.*

The remainder of this manuscript proceeds as follows. In the next subsection we describe multi-structural games and compare them to Ehrenfeucht-Fraïssé games. In the subsection that follows we review related work in complexity. We then prove the theorems mentioned above. In Section 2 we prove Theorem 4. In Section 3 we prove Theorem 14. In Section 4 we prove Theorem 15. In Section 5, we give final comments and suggestions for future research.

1.1 Multi-Structural Games

The standard Ehrenfeucht-Fraïssé game (henceforth E-F game) is played by "Spoiler" and "Duplicator" on a pair (A, B) of structures over the same FO vocabulary V, for a specified number r of *rounds*. If V contains constant symbols $\lambda_1, ..., \lambda_k$, then designated ("constant") elements c_i of A, and c'_i of B, must be associated with each λ_i. In each round, Spoiler chooses an element from A or from B, and Duplicator replies by choosing an element from the other structure. In this way, they determine sequences of elements $a_1, \ldots, a_r, c_1, \ldots, c_k$ of A and $b_1, \ldots, b_r, c'_1, \ldots, c'_k$ of B, which in turn define substructures A' of A and B' of B. Duplicator wins if the function given by $f(a_i) = b_i$ for $i = 1, \ldots, r$, and $f(c_j) = c'_j$ for $j = 1, \ldots, k$, is an isomorphism of A' and B'. Otherwise, Spoiler wins.

The equivalence theorem for E-F games [9, 12] characterizes the minimum *quantifier rank* of a sentence ϕ over V that is true for A but false for B. The quantifier rank $qr(\phi)$ is defined as zero for a quantifier-free sentence ϕ, and inductively:

$$\begin{aligned} qr(\neg\phi) &= qr(\phi), \\ qr(\phi \vee \psi) &= qr(\phi \wedge \psi) = \max\{qr(\phi), qr(\psi)\}, \\ qr(\forall x\phi) &= qr(\exists x\phi) = qr(\phi) + 1. \end{aligned}$$

▶ **Theorem 2** ([9, 12], Equivalence Theorem for E-F Games)**.** *Spoiler wins the r-round E-F game on (A, B) if and only if there is a sentence ϕ of quantifier rank at most r such that $A \models \phi$ while $B \models \neg\phi$.*

In this paper we make use of a variant of E-F games, which have come to be called *multi-structural games* [10]. Multi-structural games (henceforth M-S games) make Duplicator more powerful and can be used to characterize the *number* of quantifiers, rather than the quantifier *rank*. In an M-S game there are again two players, Spoiler and Duplicator, and there is a fixed number r of rounds. Instead of being played on a pair (A, B) of structures with the same vocabulary (as in an E-F game), the M-S game is played on a pair $(\mathcal{A}, \mathcal{B})$ of *sets of structures*, all with the same vocabulary. For k with $0 \leq k \leq r$, by a *labeled structure* after k rounds, we mean a structure along with a labeling of which elements were selected from it in each of the first k rounds. Let $\mathcal{A}_0 = \mathcal{A}$ and $\mathcal{B}_0 = \mathcal{B}$. Thus, $\mathcal{A}_0$ represents the labeled structures from $\mathcal{A}$ after 0 rounds, and similarly for $\mathcal{B}_0$ – in other words nothing is yet labelled except for constants. If $1 \leq k < r$, let $\mathcal{A}_k$ be the labeled structures originating from $\mathcal{A}$ after k rounds, and similarly for $\mathcal{B}_k$. In round $k+1$, Spoiler either chooses an element from each member of $\mathcal{A}_k$, thereby creating $\mathcal{A}_{k+1}$, or chooses an element from each member of $\mathcal{B}_k$, thereby creating $\mathcal{B}_{k+1}$. Duplicator responds as follows. Suppose that Spoiler chose an element from each member of $\mathcal{A}_k$, thereby creating $\mathcal{A}_{k+1}$. Duplicator can then make multiple copies of each labeled structure of $\mathcal{B}_k$, and choose an element from each copy, thereby creating $\mathcal{B}_{k+1}$. Similarly, if Spoiler chose an element from each member of $\mathcal{B}_k$, thereby creating $\mathcal{B}_{k+1}$, Duplicator can then make multiple copies of each labeled structure of $\mathcal{A}_k$, and choose an element from each copy, thereby creating $\mathcal{A}_{k+1}$. Duplicator wins if there is some labeled structure A in $\mathcal{A}_r$ and some labeled structure B in $\mathcal{B}_r$ where the labelings give a partial isomorphism. Otherwise, Spoiler wins.

In discussing M-S games we sometimes think of the play of the game by a given player, in a given round, as taking place on one of two "sides", the $\mathcal{A}$ side or the $\mathcal{B}$ side, corresponding to where the given player plays from on that round.

Note that on each of Duplicator's moves, Duplicator can make "every possible choice", via the multiple copies. Making every possible choice creates what we call the *oblivious strategy.* Indeed, Duplicator has a winning strategy if and only if the oblivious strategy is a winning strategy.

The following equivalence theorem, proved in [15, 10], is the analog of Theorem 2 for E-F games.

▶ **Theorem 3** ([15, 10], Equivalence Theorem for Multi-Structural Games)**.** *Spoiler wins the r-round M-S game on $(\mathcal{A}, \mathcal{B})$ if and only if there is a sentence ϕ with at most r quantifiers such that $A \models \phi$ for every $A \in \mathcal{A}$ while $B \models \neg\phi$ for every $B \in \mathcal{B}$.*

In [10] the authors provide a simple example of a property P of a directed graph that requires 3 quantifiers but which can be expressed with a sentence of quantifier rank 2. P is the property of having a vertex with both an in-edge and an out-edge. P can be expressed via the sentence $\sigma = \exists x(\exists y E(x,y) \wedge \exists y E(y,x))$, where $E(,)$ denotes the directed edge relation. In [10] it is shown that while Spoiler wins a 2-round E-F game on the two graphs A and B in Figure 1, Duplicator wins the analogous 2-round M-S game starting with these two graphs.

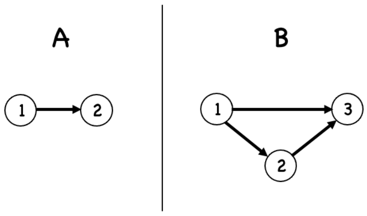


Figure 1 The graph B, on the right, contains a vertex with both an in-edge and an out-edge, while the graph A, on the left, does not.

Hence, by Theorem 3, the property P is not expressible with just 2 quantifiers.

1.2 Related Work in Complexity

Trees are a much studied data structure in complexity theory and logic. It is well known that it is impossible, in FOL, to express that a graph with no further relations is a tree [18, Proposition 3.20]. We note, however, that given a partial ordering on the nodes of a graph, it is easy to express in FOL the property that the partial ordering gives rise to a tree. The relevant sentence expresses that there is a root (i.e., a greatest element) from which all other nodes descend, and if a node x has nodes y and z as distinct ancestors then one of y and z must have the other as its own ancestor. Hence the needed sentence is the conjunction of the following two sentences:

$$\exists x \forall y (y \neq x \rightarrow y < x),$$
$$\forall x \forall y \forall z ((x < y \wedge x < z \wedge y \neq z) \rightarrow (y < z \vee z < y)).$$

There are also interesting models of computation and logics based on trees. See, for example, the literature on Finite Tree Automata [7] and Computational Tree Logic [6].

We now discuss s-t connectivity. In this paragraph only, n denotes the number of nodes in the graph and k the number of edges in a shortest path from s to t. The s-t connectivity problem has been studied extensively in both logic [1, 16] and complexity theory. Most complexity studies of this problem have focused on space and time complexity. Directed s-t connectivity is known to be NL-complete (see for example Theorem 16.2 in [19]), while undirected s-t connectivity is known to be in L [20]. Savitch [22] proved that s-t connectivity can be solved in $O(\log^2(n))$ space and $n^{\log_2(n)(1+O(1))}$ time. Recent work of Kush and Rossman [17] has shown that the *randomized* AC^0 formula complexity of s-t connectivity is at most size $n^{0.49\log_2(k)+O(1)}$, a slight improvement. Barnes, Buss, Ruzzo and Schieber [2] gave an algorithm running in both sublinear space and polynomial time for s-t connectivity. Gopalan, Lipton, and Meka [13] presented randomized algorithms for solving s-t connectivity with non-trivial time-space tradeoffs. The s-t connectivity problem has also been studied from the perspective of circuit and formula depth. For the weaker model of AC^0 formulas an $n^{\Omega(\log(k))}$ size lower bound is known to hold unconditionally [4, 5, 21].

There is also a natural and well-known correspondence with the number of quantifiers in FOL and circuit complexity, in particular with the circuit class AC^0 (constant-depth circuits comprised of NOT gates along with unbounded fan-in OR and AND gates). For example, Barrington, Immerman, and Straubing [3] proved that uniform$_{\mathsf{FO}}$-$\mathsf{AC}^0 = \mathsf{FO}[<, \mathrm{BIT}]$, thus characterizing the problems solvable in uniform AC^0 by those expressible in FOL with ordering and a BIT relation.

More generally it is known that uniform$_{\mathsf{FO}}$-$\mathsf{AC}[t(n)] = \mathsf{FO}[<, \mathrm{BIT}][t(n)]$ ([16], Theorem 5.22), i.e., FO formulas over ordering and BIT relations, defined via constant-sized blocks that are "iterated" for $O(t(n))$ times, are equivalent in expressibility with AC circuits of depth $O(t(n))$. (See [11, Appendix C] for a more detailed statement.) Generally speaking, the number of quantifiers of FOL sentences (with a regular form) roughly corresponds to the *depth* of a (highly uniform) AC^0 circuit deciding the truth or falsity of the given sentence. Thus the number of quantifiers can be seen as a proxy for "uniform circuit depth".

2 Difference in Magnitude: Quantifier Rank vs. Number of Quantifiers

Let V be a vocabulary with at least one relation symbol with arity at least 2. It is known [8] that the number of inequivalent sentences in vocabulary V with quantifier rank k is not an elementary function of k (that is, grows faster than any tower of exponents). Since the number of sentences in vocabulary V with k quantifiers is at most only double exponential in k (e.g., a function that grows like $2^{2^{p(k)}}$ for some polynomial $p(k)$ – see [11, Appendix A] for a proof), it follows by a counting argument that for each positive integer k, there is a property P of finite structures with vocabulary V that can be expressed by a sentence of quantifier rank k, but where the number of quantifiers needed to express P is not an elementary function of k. However, to our knowledge, up to now no *explicit* examples have been given of a property P where the quantifier rank of a sentence to express P is k, but where the number of quantifiers needed to express the property P is at least exponential in k. In the proof of the following theorem, we give such an explicit example.

Let $f_V(k)$ be the number of structures with k nodes up to isomorphism in vocabulary V (such as the number of non-isomorphic graphs with k nodes). Note that in the case of graphs (a single binary relation symbol), $f_V(k)$ is asymptotic to $(2^{k^2})/k!$ [14], and Stirling's formula implies that $f_V(k) = 2^{\Omega(k^2)}$). We have the following theorem.

▶ **Theorem 4.** *Assume that the vocabulary V contains at least one relation symbol with arity at least 2. There is an algorithm such that given a positive integer k, the algorithm produces a FO sentence σ of quantifier rank k where the minimum number of quantifiers needed to express σ in FOL is $kf_V(k-1)$, which grows like $2^{\Omega(k^2)}$, and where the algorithm runs in time polynomial in the length of σ.*

Proof. For simplicity, let us assume that the vocabulary V consists of a single binary relation symbol, so that we are dealing with graphs. It is straightforward to modify the proof to deal with an arbitrary vocabulary with at least one relation symbol of arity at least 2. Let us write f for f_V. Let $C_1, \ldots, C_{f(k-1)}$ be the $f(k-1)$ distinct graphs up to isomorphism with $k-1$ nodes. For each j with $1 \leq j \leq f(k-1)$, derive the graph D_j that is obtained from C_j by adding one new node with a single edge to every node in C_j. Thus, D_j has k nodes. D_j uniquely determines C_j, since C_j is obtained from D_j by removing a node a that has a single edge to every remaining node; even if there were two such nodes a, the result would be the same. Therefore, there are $f(k-1)$ distinct graphs D_j. We now give our sentence σ. Let σ_j be the sentence $\exists x_1 \cdots \exists x_k \tau_j(x_1, \ldots, x_k)$, which expresses that there is a graph with a subgraph isomorphic to D_j. Then the sentence σ is the conjunction of the sentences σ_j for $1 \leq j \leq f(k-1)$. Since the sentence σ is of length $2^{\Omega(k^2)}$, it is not hard to verify that this sentence can be generated by an algorithm running in polynomial time in the length of the sentence (there is enough time to do all of the isomorphism tests by a naive algorithm).

The sentence σ has quantifier rank k. As written, this sentence has $kf(k-1)$ quantifiers. Let A be the disjoint union of $D_1, \ldots, D_{f(k-1)}$. If p is a point in A, define B_p to be the result of deleting the point p from A. Let $\mathcal{A}$ consist only of A, and let $\mathcal{B}$ consist of the graphs B_p for each p in A. If p is in the connected component D_j of A, then B_p does not have a subgraph isomorphic to D_j. Hence, no member of $\mathcal{B}$ satisfies σ. Since the single member A of $\mathcal{A}$ satisfies σ, and since no member of $\mathcal{B}$ satisfies σ, we can make use of M-S games played on $\mathcal{A}$ and $\mathcal{B}$ to find the number of quantifiers needed to express σ.

Assume that we have labeled copies of A and the various B_p's after i rounds of an M-S game played on $\mathcal{A}$ and $\mathcal{B}$. The labelling tells us which points have been selected in each of the first i rounds. Let us say that a labeled copy of A and a labeled copy of B_p are *in harmony* after i rounds if the following holds. For each m with $1 \leq m \leq i$, if a is the point labeled m in A, and b is the point labeled m in B_p, then $a = b$. In particular, if the labeled copies of A and B_p are in harmony, then there is a partial isomorphism between the labeled copies of A and B_p.

Let Duplicator have the following strategy. Assume first that in round i, Spoiler selects in $\mathcal{A}$, and selects a point a from a labeled member A of $\mathcal{A}$. Then Duplicator (by making extra copies of labeled graphs in $\mathcal{B}$ as needed) does the following for each labeled B_p in $\mathcal{B}$. If $a \neq p$, and if the labeled copies of A and B_p before round i are in harmony, then Duplicator selects a in B_p, which maintains the harmony. If $a = p$, or if the labeled A and B_p before round i are not in harmony, then Duplicator makes an arbitrary move in B_p.

Assume now that in round i, Spoiler selects in $\mathcal{B}$. When Spoiler selects the point b from a labeled copy of B_p, then for each labeled A from $\mathcal{A}$, if the labeled copy of A is in harmony with the labeled copy of B_p before round i, then Duplicator selects b in A, and thereby maintains the harmony. We shall show shortly (Property * below) that in every round, each labeled member of $\mathcal{A}$ is in harmony with a labeled member of $\mathcal{B}$, so in the case we are now considering where Spoiler selects in $\mathcal{B}$, Duplicator does select a point in round i in each labeled member of $\mathcal{A}$.

We prove the following by induction on rounds:

Property *: If A is a labeled graph in $\mathcal{A}$ and if point p in A was not selected in the first i rounds, then there is a labeled copy of B_p that is in harmony with A after i rounds.

Property * holds after 0 rounds (with no points selected). Assume that Property * holds after i rounds; we shall show that it holds after $i+1$ rounds. There are two cases, depending on whether Spoiler moves in $\mathcal{A}$ or in $\mathcal{B}$ in round $i+1$. Assume first that Spoiler moves in $\mathcal{A}$ in round $i+1$. Assume that point p was not selected in A after $i+1$ rounds. By inductive assumption, there are labeled versions of A and B_p that are in harmony after i rounds. So by Duplicator's strategy, labeled versions of of A and B_p are in harmony after $i+1$ rounds. Now assume that Spoiler moves in $\mathcal{B}$ in round i+1. For each labeled graph A in $\mathcal{A}$, if a labeled B_p is in harmony with the labeled A after i rounds, then by Duplicator's strategy, the harmony continues between the labeled A and B_p after $i+1$ rounds. So Property * continues to hold after $i+1$ rounds. This completes the proof of Property *.

After $(kf(k-1))-1$ rounds, pick an arbitrary labeled graph A in $\mathcal{A}$. Since at most $(kf(k-1))-1$ points have been selected after $(kf(k-1))-1$ rounds, and since A contains $kf(k-1)$ points (because it is the disjoint union of $f(k-1)$ graphs each with k points), there is some point p that was not selected in A in the first $(kf(k))-1$ rounds. Therefore, by Property *, a labeled version of A and of B_p are in harmony after $(kf(k))-1$ rounds, and hence there is a partial isomorphism between the labeled A and B_p. So Duplicator wins the $(kf(k-1))-1$ round M-S game! Therefore, by Theorem 3, the number of quantifiers needed to express σ is more than $(kf(k-1))-1$. Since σ has $kf(k-1)$ quantifiers it follows that the minimum number of quantifiers need to express σ is exactly $kf(k-1)$. ◀

3 Rooted Trees

Our aim in this section is to establish the minimum number of quantifiers needed to distinguish rooted trees of depth at least k from those of depth less than k using first-order formulas, given a partial ordering on the vertices induced by the structure of the rooted tree. Figure 2 gives an example of a tree where we designate x as the root node. We define the depth of such a tree to be the maximum number of nodes in a path from the root to a leaf, where all segments in the path are directed from parent to child. Although it is more customary to denote the depth of a tree in terms of the number of *edges* along such a path, we keep to the above definition because we will often run into the special case of linear orders, which we view as trees in the natural way, and linear orders are characterized by their size (number of nodes) and we would like the size of a liner order to correspond to the depth of the associated tree. Let us denote the tree rooted at x by T_x. We make the arbitrary choice that the node x is the *largest* element in the induced partial order, so that for two nodes α, β of T_x, we have $\alpha > \beta$ if and only if there is a path $(x_1, ..., x_n)$ in T_x with $\alpha = x_1$ and $\beta = x_n$ such that x_i is a parent of x_{i+1} for $1 \leq i \leq n-1$. Thus, for example, in Figure 2, $x > q$ and $z > s$, etc.

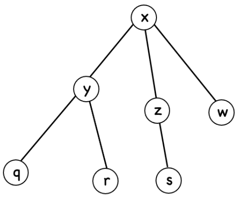


Figure 2 A rooted tree with designated root node x and depth 3.

The problem of distinguishing the depth of a rooted tree via a first-order formula with a minimum number of quantifiers is similar to the analogous problem for linear orders of different sizes, since a rooted tree has depth k or greater if and only if it has a leaf node, above which there is linear order of size at least $k-1$.

Our strategy will be to characterize a tree of depth d recursively as a graph containing a vertex v which has a subtree of depth k that includes v and everything below it, and a linear order of length $d-k$ comprising the vertices above v, where k is chosen to minimize the total number of quantifiers. We then show that this is the minimum quantifier way to characterize a tree of each given depth.

The following result is classic and key to establishing a number of fundamental inexpressibility results in FOL [18]. It is typically obtained by appeal to Theorem 2.

▶ **Theorem 5** ([18], Theorem 3.6). *Let $f(r) = 2^r - 1$. In an r-round E-F game played on two linear orders of different sizes, Duplicator wins if and only if the size of the smaller linear order is at least $f(r)$.*

Analogs of Theorem 5 are proven for M-S games in [10]. The following definition and theorems are from that paper.

▶ **Definition 6** ([10]). *Define the function $g : \mathbb{N} \to \mathbb{N}$ such that $g(r)$ is the maximum number k such that there is a formula with r quantifiers that can distinguish linear orders of size k or larger from linear orders of size less than k.*

▶ **Theorem 7** ([10]). *The function g takes on the following values: $g(1) = 1, g(2) = 2, g(3) = 4, g(4) = 10$, and for $r > 4$,*

$$g(r) = \begin{cases} 2g(r-1) & \text{if } r \text{ is even,} \\ 2g(r-1)+1 & \text{if } r \text{ is odd.} \end{cases}$$

▶ **Theorem 8** ([10]). *In an r-round M-S game played on two linear orders of different sizes Duplicator has a winning strategy if and only if the size of the smaller linear order is at least $g(r)$.*

For given positive integers r and k, we want to know if there exist sentences with r quantifiers that distinguish rooted trees of depth k or larger from rooted trees of depth smaller than k. For $k = r$, one such sentence is

$$\exists x_1 \cdots \exists x_r \bigwedge_{1 \leq i < r} (x_i < x_{i+1}), \tag{2}$$

which distinguishes rooted trees of depth r or larger from rooted trees of depth less than r. Here, if T_x is a rooted tree of depth exactly r then x_1 would be a deepest child. Since there are only finitely many inequivalent formulas in up to r variables that include the relations $<$ and $=$ and at most r quantifiers, there is some maximum such k, which we shall designate by $t(r)$. With $\mathbb{N} = \{1, 2, ...\}$, we restate this definition of t formally as follows. Note that no meaningful sentence about trees can be constructed with a single quantifier, so the definition begins at $r = 2$.

▶ **Definition 9.** *Define the function $t : \{2, 3, ...\} \to \mathbb{N}$ such that $t(r)$ is the maximum number z such that there is a formula with r quantifiers that can distinguish rooted trees of depth z or larger from rooted trees of depth less than z.*

By (2) above, $t(r) \geq r$ for $r \geq 2$. For an M-S game of r rounds on rooted trees of sizes $t(r)$ or larger on one side, and $t(r) - 1$ or smaller on the other side, by the Equivalence Theorem, Spoiler will have a winning strategy.

Since linear orders are perfectly good rooted trees, we have the following:

▶ **Observation 10.** *For all r we have $t(r) \leq g(r)$.*

Simple arguments establishing that $t(2) = 2$, and $t(3) = 4$, stemming from Observation 10, are given in [11, Sec. 3.1]. An analysis establishing $t(4) = 8$ is given in [11, Sec. 3.2]. Except for one paragraph, we shall not need that analysis, nor the particular result, though the reader may have an easier time understanding the rather intricate inductive argument that follows by first reading the analysis of this case.

In the proof of Theorem 7 [10], the authors provide explicit sentences that distinguish linear orders of size $g(r)$ or greater from those of size less than $g(r)$. From the proof of their Theorem 1.6, it can be seen that the distinguishing sentences Φ_r, for $r > 4$ take the form:

$$\Phi_r = \begin{cases} \exists x_1 \forall x_2 \cdots \forall x_{r-1} \exists x_r \phi_r & \text{for } r \text{ odd,} \\ \forall x_1 \exists x_2 \cdots \forall x_{r-1} \exists x_r \phi_r & \text{for } r \text{ even,} \end{cases}$$

where ϕ_r is quantifier-free. For odd r, the formula Φ_r says that there exists a point x_1, with a least linear order of size at least $\lfloor \frac{r}{2} \rfloor$ to both sides of x_1. For even r, the formula Φ_r says that for all x_1, there exists a linear order of at least size $\frac{r}{2}$ to one side or the other of x_1.

Let us denote by $t_\forall(r)$ the maximum number k such that rooted trees of depth k and above can be distinguished from rooted trees of depth less than k using prenex formulas with r quantifiers beginning with a universal quantifier. Equivalently, $t_\forall(r) = k$ is the largest depth of a rooted tree such that Spoiler has a winning strategy on r-round M-S games played on rooted trees of depth k or greater versus those of depth less than k when his first move is constrained to be on the tree of lesser depth. Analogously, when considering linear orders, let $g_\forall(r)$ and $g_\exists(r)$ denote, respectively, the maximum number k such that linear orders of size k and above can be distinguished from linear orders of size less than k using prenex formulas with r quantifiers beginning, respectively, with a universal or existential quantifier.

▶ **Lemma 11.** *For $r > 1$, one has $t_\forall(r) = t(r-1) + 1$.*

A proof of this very simple lemma is given in [11, Proof of Lemma 18].

▶ **Lemma 12.** *For $r \geq 2$, the following hold:*

$$g_\exists(2r) = 2g_\forall(2r-1) + 1, \tag{3}$$

$$g_\forall(2r+1) = 2g_\exists(2r). \tag{4}$$

Further, there are expressions establishing the $g_\exists$ relations having prenex signatures $\exists\forall\cdots\exists\forall\exists\exists$ with $r-1$ iterations of the $\exists\forall$ pair and then a final $\exists\exists$, while there are expressions establishing the $g_\forall$ relations having prenex signatures $\forall\exists\cdots\forall\exists\exists$ with r iterations of the $\forall\exists$ pair and then a final $\exists$.

A proof of this lemma is given in [11, Proof of Lemma 19].

▶ **Theorem 13.** *For $r \geq 2$, the following holds:*

$$t(r) = g_\forall(r-1) + t_\forall(r-1) + 1 = g_\forall(r-1) + t(r-2) + 2. \tag{5}$$

If r is odd, then $g_\forall(r-1) = g(r-1)$ so for r odd we have:

$$t(r) = g(r-1) + t_\forall(r-1) + 1 = g(r-1) + t(r-2) + 2. \tag{6}$$

A proof of this theorem is given in [11, Proof of Theorem 20].

We can combine the previous results to prove the following explicit expression for $t()$ [11, Proof of Theorem 21].

▶ **Theorem 14.** *For all $r \geq 1$ we have*

$$t(2r) = \frac{7 \cdot 4^r}{18} + \frac{4r}{3} - \frac{8}{9}, \qquad t(2r+1) = \frac{8 \cdot 4^r}{9} + \frac{4r}{3} - \frac{8}{9}.$$

A table comparing the values of f, g and t for $2 \leq r \leq 10$ is given in [11, Section 3.4].

4 s-t Connectivity

In this section we explore the number of quantifiers needed to express either directed or undirected s-t connectivity (henceforth STCON) in FOL with the binary edge relation E, as a function of the number n of edges in a shortest path between the distinguished nodes s and

t. STCON, also known as *reachability* between labelled nodes s and t, refers to the property of graphs that labelled nodes s and t are connected. STCON(n) denotes the property that s and t are connected by a path of length at most n edges.

In [11, Appendix B], we show how to describe STCON(n) using $2\log_2(n)+O(1)$ quantifiers. The following theorem generalizes that construction, to improve the number of quantifiers to $3\log_3(n)+O(1)$. A similar argument shows that $K\log_K(n)+O(1)$ quantifiers can be used for any positive integer K, although this quantity is minimized for $K=3$.

▶ **Theorem 15.** *STCON(n) can be expressed with* $3\log_3(n)+O(1)$ *quantifiers.*

Proof. We shall use Υ_i to denote sentences with i quantifiers, and τ_j to denote quantifier-free expressions, where the subscript j denotes the path length characterized by τ_j.

We start with the following simple expression stating that s and t are connected and $d(s,t)\leq 3$, where $d(s,t)$ denotes the length of the shortest path from s to t:

$$\Upsilon_2 = \exists x_1 \exists x_2 (\tau_3 \vee \tau_2 \vee \tau_1), \tag{7}$$

where:

$$\tau_3 \quad = \quad E(s,x_1) \wedge E(x_1,x_2) \wedge E(x_2,t), \tag{8}$$
$$\tau_2 \quad = \quad E(s,x_1) \wedge E(x_1,t), \tag{9}$$
$$\tau_1 \quad = \quad E(s,t). \tag{10}$$

We now iteratively add three quantifiers at each stage and slot two nodes between each of the previously established nodes, as in Figure 3.

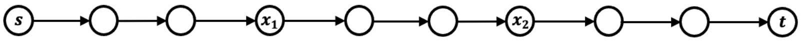


Figure 3 An illustration of slotting two nodes between each of the pre-established nodes s, x_1, x_2 and t in order to express a distance 9, $s-t$ path, using 5 quantifiers as in expressions (11) and (12).

We express that there is a path of length at most 9 from s to t, using 5 quantifiers as follows:

$$\Upsilon_5 = \exists x_1 \exists x_2 \forall x_3 \exists x_4 \exists x_5 (\tau_9 \vee \tau_8 \vee \cdots \vee \tau_1). \tag{11}$$

In this case, we just show τ_9. The simplifications required to get from τ_8 down to τ_4 are analogous to those for getting from τ_3 down to τ_1, but where we apply (8) – (10) separately to each of (12) – (14).

$$\tau_9 = \quad ((x_3 = s) \rightarrow E(s,x_4) \wedge E(x_4,x_5) \wedge E(x_5,x_1)) \;\; \wedge \tag{12}$$
$$((x_3 = t) \rightarrow E(x_1,x_4) \wedge E(x_4,x_5) \wedge E(x_5,x_2)) \;\; \wedge \tag{13}$$
$$((x_3 \neq s \wedge x_3 \neq t) \rightarrow E(x_2,x_4) \wedge E(x_4,x_5) \wedge E(x_5,t)). \tag{14}$$

Using 8 quantifiers, we can slot two new nodes between each node established in the prior step, as depicted in Figure 4. The associated logical expression is

$$\Upsilon_8 = \exists x_1 \exists x_2 \forall x_3 \exists x_4 \exists x_5 \forall x_6 \exists x_7 \exists x_8 (\tau_{27} \vee \tau_{26} \vee \cdots \vee \tau_1), \text{ and} \tag{15}$$

$$\tau_{27} = ((x_3 = s \wedge x_6 = s) \rightarrow E(s,x_7) \wedge E(x_7,x_8) \wedge E(x_8,x_4)) \;\; \wedge \tag{16}$$
$$((x_3 = s \wedge x_6 = t) \rightarrow E(x_4,x_7) \wedge E(x_7,x_8) \wedge E(x_8,x_5)) \;\; \wedge \tag{17}$$
$$((x_3 = s \wedge (x_6 \neq s \wedge x_6 \neq t)) \rightarrow E(x_5,x_7) \wedge E(x_7,x_8) \wedge E(x_8,x_1)) \;\; \wedge \tag{18}$$
$$\ldots \tag{19}$$
$$(((x_3 \neq s \wedge x_3 \neq t) \wedge (x_6 \neq s \wedge x_6 \neq t)) \rightarrow E(x_5,x_7) \wedge E(x_7,x_8) \wedge E(x_8,t)). \tag{20}$$

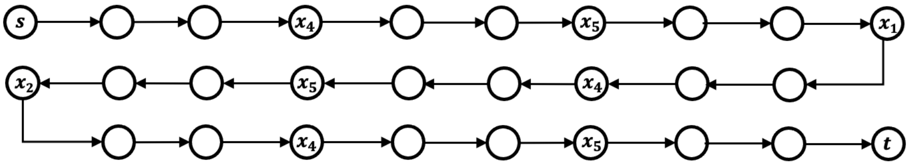


Figure 4 Slotting two nodes between each of the pre-established nodes s, x_1, x_2, x_4, x_5, and t in order to express a distance 27, $s-t$ path, using 8 quantifiers as in expressions (15) and (16)-(20).

The expression τ_{27} will have the two "pivot points" around the universally quantified variables x_3 and x_6 and so have $3^2 = 9$ antecedent conditions corresponding to the possible ways the universally quantified variables x_3 and x_6 can each take the values s, t or neither s nor t. The right hand side of each equality condition describes how to fill in the edges in Figure 4 with two new vertices (utilizing the two newest existentially quantified variables, x_7 and x_8) and three new edges.

In this way we obtain sentences with $3n-1$ quantifiers that can express STCON instances of path length up to 3^n. Thus, when n is a power of 3, we can express STCON instances of length n with $3\log_3(n) - 1 \approx 1.893\log_2(n) - 1$ quantifiers, and when n is *not* a power of 3, with $\lfloor 3\log_3(n) + 2 \rfloor$ quantifiers. The theorem therefore follows. ◀

A Remark on Lower Bounds on Quantifier Rank and hence on Number of Quantifiers

Lower bounds on the number of quantifiers for s-t connectivity follow readily from the literature. The well-known proof that connectivity is not expressible in FOL ([16, Prop. 6.15] or [18, Corollary 3.19]) can be used to establish that s-t connectivity with path length n is not expressible as a formula of quantifier rank $\log_2(n) - c$ for some constant c.

▶ **Theorem 16** (Immerman, Proposition 6.15 [16]). *There exists a constant c such that s-t connectivity to path length n is not expressible as a formula of quantifier rank $\log_2(n) - c$.*

Since the quantifier rank is a lower bound on the number of quantifiers, the previous theorem immediately implies a lower bound on the number of quantifiers as well. While we have shown that STCON(n) can be expressed with $3\log_3(n) + O(1)$ quantifiers, we note that the minimum quantifier *rank* of STCON(n) is well-known to be lower.

▶ **Theorem 17** ([18]). *s-t connectivity to path length n can be expressed with a formula of quantifier rank $\log_2(n) + O(1)$.*

5 Final Comments and Future Directions

Although progress on M-S games did not come until 40 years after their initial discovery in [15], the results of this paper show that these games are quite amenable to analysis, and the more detailed information they give about the requisite quantifier structure has the potential to yield many new and interesting insights.

Theorem 4 tells us that the number of quantifiers can be more than exponentially larger than the quantifier rank. This shows that the number of quantifiers is a more refined measure than the quantifier rank, and gives an interesting and natural measure of the complexity of a FO formula. It would be interesting to find explicit examples where the quantifier rank is k, but where the required number of quantifiers grows even faster than in our example in the proof of Theorem 4. Ideally, we would even like to find explicit examples where the required number of quantifiers is non-elementary in k.

We have extended the results on the number of quantifiers needed to distinguish linear orders of different sizes [10] to distinguish rooted trees of different depths. Can this line of attack be carried further to incorporate other structures, say to other structures with induced partial orderings such as finite lattices?

The most immediate question arising from our work is whether one can improve the known upper or lower bounds on the number of quantifiers needed to express s-t connectivity. In particular, what is the smallest constant $c \geq 1$ such that s-t connectivity (up to path length n) is expressible using $c \log_2(n)$ quantifiers? Our Theorem 15 shows that $\mathit{Quants}(\text{STCON}(n))$ is at most $3\log_3(n) + O(1) \approx 1.893 \log_2(n) + O(1)$. The well-known lower bound of $\mathit{Rank}(\text{STCON}(n)) \geq \log_2(n) - O(1)$ (cited as Theorem 16) yields the only lower bound we know on $\mathit{Quants}(\text{STCON}(n))$, but we also know the upper bound $\mathit{Rank}(\text{STCON}(n)) \leq \log_2(n) + O(1)$ (cited as Theorem 17). As these upper and lower bounds for the quantifier rank of $\text{STCON}(n)$ essentially match, in order to improve the lower bound on $\mathit{Quants}(\text{STCON}(n))$ further (by a multiplicative constant), we *cannot* rely on a rank lower bound: we will have to resort to other methods, such as M-S games.

Another question is whether we can find other problems with even larger quantifier number lower bounds than logarithmic ones. Let us stress that substantially larger lower bounds on the number of quantifiers would have major implications for circuit complexity lower bounds. For example, by the standard way of expressing uniform circuit complexity classes in FOL [16], a property (over the $<$ relation) that requires $\log^{\alpha(n)}(n)$ quantifiers, where $\alpha(n)$ is an unbounded function of n, would imply a lower bound for uniform$_{\mathsf{FO}}$-NC. See [11, Appendix C] for an exact statement.

Another interesting direction to push this research is to extend the notion of multi-structural games to 2nd-order logic, FOL with counting, or to fixed point logic.

References

1 Miklós Ajtai and Ronald Fagin. Reachability is harder for directed than for undirected finite graphs. *J. Symbolic Logic*, 55(1):113–150, March 1990.

2 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

3 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. In *Proceedings: Third Annual Structure in Complexity Theory Conference, Georgetown University, Washington, D. C., USA, June 14-17, 1988*, pages 47–59. IEEE Computer Society, 1988. doi:10.1109/SCT.1988.5262.

4 Paul Beame, Russell Impagliazzo, and Toniann Pitassi. Improved depth lower bounds for small distance connectivity. *Comput. Complex.*, 7(4):325–345, 1998. doi:10.1007/s000370050014.

5 Xi Chen, Igor Carboni Oliveira, Rocco A. Servedio, and Li-Yang Tan. Near-optimal small-depth lower bounds for small distance connectivity. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 612–625. ACM, 2016. doi:10.1145/2897518.2897534.

6 Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on the Logic of Programs, Lecture Notes in Computer Science*, volume 131, pages 52–71, 1981.

7 Hubert Comon, Max Dauchet, Rémi Gilleroz, Florent Jacquemard, Denis Luguiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, 1999. URL: http://www.eecs.harvard.edu/~shieber/Projects/Transducers/Papers/comon-tata.pdf.

8 A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Model theory makes formulas large. In L. Arge, C. Cachin, T. Jurdziński, and A. Tarlecki, editors, *ICALP07*, volume 4596, pages 913–924. Springer, 2007.

9 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

10 Ronald Fagin, Jonathan Lenchner, Kenneth W. Regan, and Nikhil Vyas. Multi-structural games and number of quantifiers. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2021.

11 Ronald Fagin, Jonathan Lenchner, Nikhil Vyas, and Ryan Williams. On the number of quantifiers as a complexity measure, 2022. `arXiv:2207.00104`.

12 Roland Fraïssé. Sur quelques classifications des systèmes de relations. *Université d'Alger, Publications Scientifiques, Série A*, 1:35–182, 1954.

13 Parikshit Gopalan, Richard J. Lipton, and Aranyak Mehta. Randomized time-space tradeoffs for directed graph connectivity. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 208–216. Springer, 2003. `doi:10.1007/978-3-540-24597-1_18`.

14 Frank Harary. Note on Carnap's relational asymptotic relative frequencies. *J. Symb. Log.*, 23(3):257–260, 1958.

15 Neil Immerman. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981.

16 Neil Immerman. *Descriptive complexity*. Graduate Texts in Computer Science. Springer, 1999. `doi:10.1007/978-1-4612-0539-5`.

17 Deepanshu Kush and Benjamin Rossman. Tree-depth and the formula complexity of subgraph isomorphism. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 31–42. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00012`.

18 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2012. URL: `https://www.springer.com/gp/book/9783540212027`.

19 Christos Papdimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Reading, MA USA, 1995.

20 Omer Reingold. Undirected st-connectivity in log-space. *Proceedings of the 37th annual ACM symposium on the theory of computing*, pages 376–385, 2005.

21 Benjamin Rossman. Formulas vs. circuits for small distance connectivity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 203–212. ACM, 2014. `doi:10.1145/2591796.2591828`.

22 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. `doi:10.1016/S0022-0000(70)80006-X`.

Non-Determinism in Lindenmayer Systems and Global Transformations

Alexandre Fernandez ✉
Univ Paris Est Creteil, LACL, 94000, Creteil, France

Luidnel Maignan ✉
Univ Paris Est Creteil, LACL, 94000, Creteil, France
Université Paris-Saclay, Inria, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France

Antoine Spicher ✉
Univ Paris Est Creteil, LACL, 94000, Creteil, France

Abstract

Global transformations provide a categorical framework for capturing synchronous rewriting systems, generalizing cellular automata to dynamical systems over dynamic spaces. Originally developed for addressing deterministic dynamical systems, the presented work raises the question of non-determinism. While a usual approach is to develop a general non-deterministic setting where deterministic systems can be retrieved as a specific case, we show here that by choosing the right parametrization, global transformations can already be used to handle non-determinism. Context-free Lindenmayer systems, already shown to be captured by global transformation in the deterministic case, are used to illustrate the approach. From this concrete example, the formal obstructions are exhibited, leading to a solution involving a 2-categorical monad and its associated Kleisli construction.

2012 ACM Subject Classification Theory of computation → Parallel computing models; Theory of computation → Concurrency; Mathematics of computing → Discrete mathematics; Computing methodologies → Modeling and simulation

Keywords and phrases Global Transformations, Non-deterministic Dynamical Systems, Lindenmayer Systems, Category Theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.49

1 Introduction

Global transformations (GT) has been introduced in [12] as a precise formal description of dynamical systems defined over a space which is also dynamic while being still synchronous. This synchronicity property makes GT apart from the main stream of the literature on graph transformations and graph rewriting systems. They however share with this literature the fact that they are very generically defined using category theory in order not to be tied to a specific kind of space. For instance, GT can be used with directed or undirected graphs, labeled or not, hypergraphs, abstract cell complexes.

In order to present the framework to a public not familiar with category theory, the well known deterministic Lindenmayer systems have been presented in terms of GT in [4]. In this paper, we return to Lindenmayer systems, not as a pedagogical exercise but to explore non-determinism in GT in the simplest possible concrete setting.

A usual approach for studying non-determinism from a deterministic object consists in generalizing the deterministic object into a non-deterministic one and then to show that the original deterministic case is a degenerate case of the new non-deterministic setting. We rather seek to demonstrate that non-determinism is already encompassed as a particular case of the current definition of GT. This is in complete analogy with dynamical systems. Indeed, the general definition of dynamical systems is in terms of sets of states and evolution functions. Non-deterministic dynamical systems are *particular* dynamical systems where

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 49; pp. 49:1–49:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the sets happen to be powersets of an underlying set of real states. This quest emerged from the necessity to talk about non-deterministic, probabilistic and quantum systems, with the intuition that it should not be very different to what happens for dynamical systems. It is not obvious at first that this is possible, because GT are mainly about the notion of (dynamic) locality, but non-deterministic, probabilistic and quantum systems all exhibit somewhat non-local features of correlation and entanglement.

At this point, it is not possible to say much without properly defining the terms that we use. So we dig into the formal definitions of the main objects in Section 2. This will allow us to show that a naive approach of this quest, based on powersets, does not work as explored in Section 3. In Section 4, we circumvent the obstruction in the most natural way and go to an intuitive solution that does not look as a dynamical system, because the transformation does not go from a set to itself but from a set to a bigger set. Section 5 comes back on the relation between dynamical system and non-deterministic system in terms of monad and Kleisli category. This well-known categorical point of view on dynamical systems leads us to consider a 2-monad and its Kleisli 2-category, linking together the previous attempts with this setting as a coherent whole, and providing a positive answer to our quest.

2 Preliminaries

The following notations are used along the paper. Formal language operations are written as follows. For a given alphabet Σ, Σ^* is the set of finite words on Σ, and ε is the empty word. The length of a word $u \in \Sigma^*$ is written $|u|$ and its ith letter, for $0 \leq i < |u|$, is written u_i. The concatenation of two words $u, v \in \Sigma^*$ is written $u \cdot v$. Concatenation of sets $U, V \subseteq \Sigma^*$ is written $U \cdot V$ and is defined by $\{u \cdot v \mid u \in U, v \in V\}$. For a given set U, the powerset of U is written $\mathrm{P}(U)$. The cartesian product of a family of sets $\{U_i\}_{i\in I}$ is written $\prod_{i\in I} U_i$, and the projection on component i for an element x of that product is written $x(i)$.

The reader is assumed familiar with basic notions of category theory. The colimit of a given diagram D is written $\mathrm{Colim}(D)$. The notation F/x stands for the comma category F over x where x is an object of some category and F is a functor into that category. The first projection is then written $\mathrm{Proj}[F/x]$. The restriction of a category $\mathbf{C}$ to the full subcategory with objects $S \subset \mathbf{C}$ is written $\mathbf{C} \upharpoonright S$. The restriction of a functor F to a subcategory $\mathbf{C}$ of its domain is written $F \upharpoonright \mathbf{C}$ as well.

2.1 Global Transformations

A global transform is a *synchronous* rewriting rule system. This is made possible by considering inclusions between rules in order to make explicit how overlapping applications of rules should be handle, similarly to the notion of amalgamation in classical graph transformation [2, 1]. Asking the coherence of the rule inclusions means exactly to ask them to form categories and functors, leading to the following definitions.

▶ **Definition 1** (rule systems and global transformations). *A* rule system *T on a category $\mathbf{C}$ is a tuple $\langle \mathbf{\Gamma}_T, \mathrm{L}_T, \mathrm{R}_T\rangle$ where $\mathbf{\Gamma}_T$ is a category whose objects and morphism are called* rules *and* rule inclusions*, $\mathrm{L}_T : \mathbf{\Gamma}_T \to \mathbf{C}$ is a full embedding functor called the* l.h.s. functor*, and $\mathrm{R}_T : \mathbf{\Gamma}_T \to \mathbf{C}$ is a functor called the* r.h.s. functor*. A rule system is a* global transformation *when the functor:*

$$T(-) = \mathrm{Colim}(\mathrm{R}_T \circ \mathrm{Proj}[\mathrm{L}_T/-]) \tag{1}$$

abusively also denoted T, is well-defined. The subscript T is omitted when this does not lead to any confusion.

This definition is a simplified version of alternative definitions found in [12, 5] but is enough for the present study. This definition makes GT T into a (pointwise) left Kan extensions of R_T along L_T, *i.e.*, a pair $\langle T, \eta : \mathrm{R}_T \Longrightarrow T \circ \mathrm{L}_T\rangle$ such that any other pair $\langle K, \rho : \mathrm{R}_T \Longrightarrow K \circ \mathrm{L}_T\rangle$ factorizes through η by a unique $\lambda : T \Longrightarrow K$ as in the following diagram. The natural transformation η is the identity, *i.e.*, $T \circ \mathrm{L}_T = \mathrm{R}_T$.

$$\begin{tikzcd}
\mathbf{C} \arrow[rrd, "T"] \arrow[rrd, bend left, "K"] \arrow[rd, phantom, "\lambda\Uparrow"] & & \\
\mathbf{\Gamma} \arrow[u, "\mathrm{L}_T"] \arrow[rr, "\mathrm{R}_T"'] \arrow[r, phantom, "\rho\Uparrow\ \ \eta\Uparrow"] & & \mathbf{C}
\end{tikzcd}$$

For more information on GT, one can consult [12, 4, 5, 7] but the specific case considered below might be enough to exemplify the relation between synchronous rewriting systems and left Kan extensions as explored by GT.

2.2 Non-Deterministic Lindenmayer Systems

Lindenmayer systems are a variant of formal grammars for specifying languages through a mechanism of parallel string rewriting [13, 10]. The present study focuses on Lindenmayer systems without context, where a word u on an alphabet Σ is synchronously rewritten by mapping each individual letter to a word (deterministic case) or set of words (non-deterministic case).

▶ **Definition 2.** *A* non-deterministic Lindenmayer system *on an alphabet Σ is given by a function $\delta : \Sigma \to \mathrm{P}(\Sigma^*)$ and produces the function on words $\Delta : \Sigma^* \to \mathrm{P}(\Sigma^*)$:*

$$\Delta(u) = \{v_0 \cdot \ldots \cdot v_{|u|-1} \mid (v_0, \ldots, v_{|u|-1}) \in \delta(u_0) \times \ldots \times \delta(u_{|u|-1})\} \tag{2}$$

and the dynamical system $\overline{\Delta} : \mathrm{P}(\Sigma^*) \to \mathrm{P}(\Sigma^*)$:

$$\overline{\Delta}(U) = \bigcup_{u \in U} \Delta(u). \tag{3}$$

▶ **Example 3.** Consider the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ with function δ defined by $\delta(\mathtt{a}) = \{\mathtt{a}, \mathtt{ab}\}$ and $\delta(\mathtt{b}) = \{\varepsilon, \mathtt{b}\}$. In this system, each a may potentially produce a new b on its right, and each b remains or vanishes. The behavior on the word ab is given by $\Delta(\mathtt{ab}) = \{\mathtt{a}\cdot\varepsilon, \mathtt{ab}\cdot\varepsilon, \mathtt{a}\cdot\mathtt{b}, \mathtt{ab}\cdot\mathtt{b}\} = \{\mathtt{a}, \mathtt{ab}, \mathtt{abb}\}$. Notice that ab is produced in two different ways.

In [4], deterministic Lindenmayer systems (without and with context) are presented as GT. This encoding relies on the category $\mathbf{W}$ of finite words that also plays a crucial role in this study. Let us fix the symbol Σ for the alphabet.

▶ **Definition 4.** *Let $\mathbf{W}$ be the category having Σ^* as set of objects, and*

$$\mathbf{W}(u, v) := \{\, p \in \{0, \ldots, |v| - |u|\} \mid u_i = v_{p+i}\ \forall i \in \{0, \ldots, |u| - 1\}\,\}$$

as set of arrows from any $u \in \Sigma^*$ to any $v \in \Sigma^*$. We write $p : u \to v$ for $p \in \mathbf{W}(u, v)$. The composite $q \circ p : u \to w$ of any two arrows $q : v \to w$ and $p : u \to v$ is given by $q + p$, 0 being the identity arrow of any $u \in \Sigma^*$.

Category $\mathbf{W}$ records the many places words appear in each other which is the relevant notion of *locality* for Lindenmayer systems. Indeed, the concatenations involved in Equation (2) correspond to colimits in $\mathbf{W}$, as stated by the following theorem, and is the basic ingredient of the expression of deterministic Lindenmayer systems as GT.

▶ **Theorem 5** (from [4]). *For any two words u, v, the word $u \cdot v$ is the colimit in $\mathbf{W}$ of the diagram:*

$$\begin{tikzcd}
u & & v & & & u \cdot v & \\
& \varepsilon \arrow[ul, "|u|"] \arrow[ur, "0"'] & & \text{with components} \quad u \arrow[urr, "0"] & & & v \arrow[ul, "|u|"'] \; . \\
& & & & & \varepsilon \arrow[ull, "|u|"] \arrow[ur, "0"'] \arrow[uu, "|u|"'] &
\end{tikzcd}$$

▶ **Example 6.** Let us illustrate the construction of [4] with a simple example. Consider the deterministic Lindenmayer system defined on $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ by $\delta(\mathtt{a}) = \{\mathtt{ab}\}$ and $\delta(b) = \{\mathtt{a}\}$. The associated GT $T = \langle \mathbf{\Gamma}, \mathrm{L}, \mathrm{R} \rangle$ is completely determined by the following diagrams presenting respectively the category of rules $\mathbf{\Gamma}$ as a full subcategory of $\mathbf{W}$ with inclusion functor L, and the image of $\mathbf{\Gamma}$ by R:

$$\begin{tikzcd}
\mathtt{a} & \varepsilon \arrow[l, bend right, "1"'] \arrow[l, bend left, "0"] \arrow[r, bend left, "1"] \arrow[r, bend right, "0"'] & \mathtt{b} & {} \arrow[r, mapsto, "\mathrm{R}"] & {} & \mathtt{ab} & \varepsilon \arrow[l, bend right, "2"'] \arrow[l, bend left, "0"] \arrow[r, bend left, "1"] \arrow[r, bend right, "0"'] & \mathtt{a}
\end{tikzcd}$$

The computation of $T(ab)$ as defined by Equation (1) is pictured by the following diagrams:

$$\begin{tikzcd}
& \mathtt{ab} & & & & & & \mathtt{ab} \cdot \mathtt{a} & \\
\mathtt{a} \arrow[ur, "0"] & & \mathtt{b} \arrow[ul, "1"'] & \mathtt{ab} & & \mathtt{a} & \mathtt{ab} \arrow[ur, "0"] & & \mathtt{a} \arrow[ul, "2"'] \\
& \varepsilon \arrow[ul, "1"] \arrow[ur, "0"'] \arrow[uu, "1"'] & & & \varepsilon \arrow[ul, "2"] \arrow[ur, "0"'] & & & \varepsilon \arrow[ul, "2"] \arrow[ur, "0"'] \arrow[uu, "2"'] &
\end{tikzcd}$$

On the left, the diagram illustrates the information provided by the comma category $\mathrm{L}/\mathtt{ab}$. It corresponds to the pattern matching of the rules l.h.s. in the input word $\mathtt{ab}$. On the middle, the diagram $\mathrm{R} \circ \mathrm{Proj}[\mathrm{L}/\mathtt{ab}]$ is represented. On the right, the application of Theorem 5 for computing the colimit requested by $T(\mathtt{ab})$ constructs the expected result $\mathtt{aba}$.

3 The Challenge of Powersets

Let us recall the goal and make it more precise in light of the formal definitions. We already know from [4] that deterministic Lindenmayer systems are GT, and now we want to establish that non-deterministic Lindenmayer systems are also GT, without any extension of the framework. This means that we want to provide a rule system (Definition 1) based on δ such that $\overline{\Delta}$ (Definition 2) is obtained by the colimit formula of Equation (1). This implies in particular that we need to design the appropriate category, say $\mathbf{PW}$, with a calibrated notion of arrows to capture what we can informally call *non-deterministic locality.* The first idea that comes to mind is to take $\mathrm{P}(\Sigma^*)$ as set of objects for $\mathbf{PW}$ so that an object represents a set of possibilities. It remains to define arrows of $\mathbf{PW}$. However, it will cause difficulties as we are now going to see.

To make this more concrete, let us take the simple example of $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $\delta(\mathtt{a}) = \{\mathtt{a}, \mathtt{b}\}$ and $\delta(\mathtt{b}) = \{\varepsilon\}$. First, notice that $\Delta(\varepsilon) = \{\varepsilon\}$. Second, on the input $\mathtt{aa}$, it produces the behavior $\Delta(\mathtt{aa}) = \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}\}$ corresponding to the four possibilities combining the possible evolutions of each $\mathtt{a}$. Using the lessons learned in the deterministic case as illustrated in Example 6, we make the educated guess that the involved diagram should be of the form:

$$\begin{tikzcd}
\{\mathtt{a},\mathtt{b}\} & & \{\mathtt{a},\mathtt{b}\} \\
& \{\varepsilon\} \arrow[ul, "\mathrm{end}"] \arrow[ur, "\mathrm{beg}"'] &
\end{tikzcd}$$

to produce the colimit

$$\begin{tikzcd}
& \{\mathtt{aa},\mathtt{ab},\mathtt{ba},\mathtt{bb}\} & \\
\{\mathtt{a},\mathtt{b}\} \arrow[ur, "\eta_1"] & & \{\mathtt{a},\mathtt{b}\} \arrow[ul, "\eta_2"'] \\
& \{\varepsilon\} \arrow[ul, "\mathrm{end}"] \arrow[ur, "\mathrm{beg}"'] \arrow[uu, "\eta_3"'] &
\end{tikzcd}$$.

The diagram contains $\{\varepsilon\}$ for the matched ε between the two $\mathtt{a}$ in $\mathtt{aa}$, and two times $\{\mathtt{a},\mathtt{b}\}$ for each occurrence of $\mathtt{a}$. The arrows in the diagram indicate that the empty words at the end of the words in the left object need to correspond to the empty words at the beginning of the words in the right object as in Theorem 5. The expected colimit results in the concatenation $\{\mathtt{aa},\mathtt{ab},\mathtt{ba},\mathtt{bb}\}$. Based on these assumptions, we know that:

- the arrow "end" of $\mathbf{PW}$ should be based on arrows $1 : \varepsilon \to \mathtt{a}$ and $1 : \varepsilon \to \mathtt{b}$ of $\mathbf{W}$,
- the arrow "beg" of $\mathbf{PW}$ should be based on arrows $0 : \varepsilon \to \mathtt{a}$ and $0 : \varepsilon \to \mathtt{b}$ of $\mathbf{W}$,
- the arrow η_1 should be based on $0 : \mathtt{a} \to \mathtt{aa}$, $0 : \mathtt{a} \to \mathtt{ab}$, $0 : \mathtt{b} \to \mathtt{ba}$, and $0 : \mathtt{b} \to \mathtt{bb}$,
- the arrow η_2 should be based on $1 : \mathtt{a} \to \mathtt{aa}$, $1 : \mathtt{b} \to \mathtt{ab}$, $1 : \mathtt{a} \to \mathtt{ba}$, and $1 : \mathtt{b} \to \mathtt{bb}$,
- the arrow η_3 should be based on $1 : \varepsilon \to \mathtt{aa}$, $1 : \varepsilon \to \mathtt{ab}$, $1 : \varepsilon \to \mathtt{ba}$, and $1 : \varepsilon \to \mathtt{bb}$.

A natural choice for designing the arrows of $\mathbf{PW}$ is then to gather of these arrows of $\mathbf{W}$ into sets of arrows, leading to the following definition.

▶ **Definition 7.** *Let $\mathbf{PW}$ be the category having $\mathrm{P}(\Sigma^*)$ as set of objects, and*

$$\mathbf{PW}(U, V) := \mathrm{P}(\{\, p : u \to v \in \mathbf{W} \mid u \in U, v \in V \,\})$$

as set of arrows from any $U \in \mathrm{P}(\Sigma^*)$ to any $V \in \mathrm{P}(\Sigma^*)$. As usual, we write $P : U \to V$ for $P \in \mathbf{PW}(U, V)$. The composite $Q \circ P : U \to W$ of any two arrows $P : U \to V$ and $Q : V \to W$ is given by $\{p + q : u \to w \mid p : u \to v \in P, q : v \to w \in Q\}$, $\{0 : u \to u \mid u \in U\}$ being the identity arrow of any $U \in \Sigma^*$.

Notice that for any $P \in \mathbf{PW}(U, V)$, P does not contain integers but arrows of $\mathbf{W}$ with their domain and codomain. To avoid any confusion, we always write $p : u \to v \in P$ for elements of P.

▶ **Example 8.** With this category, the previous example works as expected if we take our previous list of constraints to define end, beg, η_1, η_2, and η_3 as sets of arrows: end $= \{1 : \varepsilon \to \mathtt{a}, 1 : \varepsilon \to \mathtt{b}\}$, beg $= \{0 : \varepsilon \to \mathtt{a}, 0 : \varepsilon \to \mathtt{b}\}$, $\eta_1 = \{0 : \mathtt{a} \to \mathtt{aa}, 0 : \mathtt{a} \to \mathtt{ab}, 0 : \mathtt{b} \to \mathtt{ba}, 0 : \mathtt{b} \to \mathtt{bb}\}$, and so on so forth. To see this, consider another cocone to some object $U \in \mathbf{PW}$ with components $\rho_1 : \{\mathtt{a},\mathtt{b}\} \to U$, $\rho_2 : \{\mathtt{a},\mathtt{b}\} \to U$, $\rho_3 : \{\varepsilon\} \to U$. We aim at showing that there is a unique arrow $\mu : \{\mathtt{aa},\mathtt{ab},\mathtt{ba},\mathtt{bb}\} \to U$ such that $\rho_i = \mu \circ \eta_i$ for $i \in \{1, 2, 3\}$.

First notice that ρ_1, ρ_2, and ρ_3 are bijectively related. Indeed, by definition of the composition in $\mathbf{PW}$ and by commutativity of the cocone, each $f_1 : l_1 \to u \in \rho_1$ compose with the unique appropriate arrow $1 : \varepsilon \to l_1$ of "end" to give a bijectively corresponding arrow in $f_1 + 1 : \varepsilon \to u \in \rho_3$. Surjectivity holds by the definition of the composition. Injectivity stands on the fact that an occurrence of $\mathtt{a}$ and an occurrence of $\mathtt{b}$ in a given word (here u) cannot be at the same position. Therefore, given $f_1 + 1 : \varepsilon \to u \in \rho_3$, there is a unique letter l_1 such that $f_1 : l_1 \to u \in \rho_1$. Similarly, there is also such a bijection mapping each $f_2 : l_2 \to u \in \rho_2$ to $f_2 : \varepsilon \to u \in \rho_3$. Consequently, the mediating μ has to contain exactly arrows with domain the concatenation of source letters l_i of two bijectively related arrows $f_1 : l_1 \to u \in \rho_1$ and $f_2 : l_2 \to u \in \rho_2$, and with codomain u. Formally μ has to be $\{f_1 : l_1 \cdot l_2 \to u \mid f_1 : l_1 \to u \in \rho_1$ and corresponding $f_1 + 1 : l_2 \to u \in \rho_2\}$.

The fact that this mediating commutes appropriately comes directly from the fact that $\mu \circ \eta_1 = \{f_1 : l_1 \to u \mid 0 : l_1 \to l_1 \cdot l_2 \in \eta_1, f_1 : l_1 \cdot l_2 \to u \in \mu\}$ which turns to be ρ_1. Commutations for ρ_2 and ρ_3 hold as well. Finally, μ is unique by construction.

While the previous example works, the proof uses explicitly the properties specific to the example. In the general case, the construction fails as shown by the following example.

▶ **Example 9.** We consider a different non-deterministic Lindenmayer system where $\Sigma = \{\mathtt{a}\}$ and $\delta(\mathtt{a}) = \{\mathtt{a}, \mathtt{aa}\}$. This produces the behavior $\Delta(\mathtt{aa}) = \{\mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}\}$. Notice in particular that the concatenations $\mathtt{a} \cdot \mathtt{aa}$ and $\mathtt{aa} \cdot \mathtt{a}$ give the same word $\mathtt{aaa}$ in $\Delta(\mathtt{aa})$. Following the same construction as presented in Example 8, we consider the following colimit:

$$\begin{tikzcd}
 & \{\mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}\} & \\
\{\mathtt{a}, \mathtt{aa}\} \arrow[ur, "\eta_1"] & & \{\mathtt{a}, \mathtt{aa}\} \arrow[ul, "\eta_2"'] \\
 & \{\varepsilon\} \arrow[uu, "\eta_3"'] \arrow[ul, "\mathrm{end}"] \arrow[ur, "\mathrm{beg}"'] &
\end{tikzcd}$$

where

- $\eta_1 = \{0 : \mathtt{a} \to \mathtt{aa}, 0 : \mathtt{a} \to \mathtt{aaa}, 0 : \mathtt{aa} \to \mathtt{aaa}, 0 : \mathtt{aa} \to \mathtt{aaaa}\}$,
- $\eta_2 = \{1 : \mathtt{a} \to \mathtt{aa}, 1 : \mathtt{aa} \to \mathtt{aaa}, 2 : \mathtt{a} \to \mathtt{aaa}, 2 : \mathtt{aa} \to \mathtt{aaaa}\}$, and
- $\eta_3 = \{1 : \varepsilon \to \mathtt{aa}, 1 : \varepsilon \to \mathtt{aaa}, 2 : \varepsilon \to \mathtt{aaa}, 2 : \varepsilon \to \mathtt{aaaa}\}$.

To see that this cocone is *not* a colimit, consider the cocone to $\{\mathtt{aaa}\}$ having components $\rho_1 = \{0 : \mathtt{a} \to \mathtt{aaa}\}$, $\rho_2 = \{1 : \mathtt{aa} \to \mathtt{aaa}\}$, and $\rho_3 = \{1 : \varepsilon \to \mathtt{aaa}\}$. The only possible mediating is $\mu = \{0 : \mathtt{aaa}, \mathtt{aaa}\}$ but it fails to respect the required commutation property. Indeed, $\mu \circ \eta_1 = \{0 : \mathtt{a} \to \mathtt{aaa}, 0 : \mathtt{aa} \to \mathtt{aaa}\}$ which definitively differs from ρ_1.

So the first and simplest intuitive idea does not work and we have not designed the appropriate category. In particular, the construction fails since it is not able to distinguish the different ways for generating a given output (case $\mathtt{aaa}$ in Example 9). We then deduce that an appropriate category, if it exists, needs to keep track of this information. Moreover, notice that in Equation (1), the arrows of the category are not only used for constructing the result as a colimit, but also to decompose an input U into a coma category L_T/U and produce the diagram by the formula $\mathrm{R} \circ \mathrm{Proj}[\mathrm{L}_T/U]$. So the previous discussion has only addressed one side of the problem.

4 From Sets to Indexed Families

There are plenty of other possible definitions for designing arrows of **PW** and the research space to get the right one is pretty large. However, having considered several attempts of definitions, we come to the following working hypothesis.

▶ **Conjecture 10.** *There is no category with set of objects* $\mathrm{P}(\Sigma^*)$ *and an appropriate choice of arrows* $\mathrm{beg}_U, \mathrm{end}_U : \{\varepsilon\} \to U$ *producing concatenation as a colimit.*

Irrespectively of the validity of that conjecture, we choose to circumvent directly the problem that occurred in the previous example and to jump to other aspects of the program. As previously evoked, the obstruction was that $\mathtt{a} \cdot \mathtt{aa}$ and $\mathtt{aa} \cdot \mathtt{a}$ merged into a single word $\mathtt{aaa}$, so that the mediating arrow could not specify whether it needed this word as a result of the first concatenation or of the second one. To keep track of that information, we simply allow for a word to appear many times and we take families of words instead of sets of words. Taking this path of least resistance, we obtain the following category where an arrow from a source family of words to a target family of words, picks a unique word from the source for each word in the target. This additional "unique source word" property is respected by all the arrows considered up to now. Taking the view that an arrow $p : u \to v$ in $\mathbf{W}$ selects a subword of v, this constraint says that an arrow to a family of words similarly selects a subword for each member of that family.

▶ **Definition 11.** *For any $\mathbf{C} \in \mathbf{Cat}$, $\mathbb{O}\mathbf{C} \in \mathbf{Cat}$ has pairs $(I : \mathrm{Set}, U \in \mathbf{C}^I)$ as objects and*

$$\mathbb{O}\mathbf{C}((I,U),(J,V)) := \{\, (P : J \to I, P' : \prod_{j \in J} \mathbf{C}(U_{P(j)}, V_j))\,\}$$

as set of arrows from any $(I,U) \in \mathbb{O}\mathbf{C}$ to any $(J,V) \in \mathbb{O}\mathbf{C}$. As usual, we write $(P,P') : (I,U) \to (J,V)$ for $(P,P') \in \mathbb{O}\mathbf{C}((I,U),(J,V))$. The composite $(Q,Q') \circ (P,P') : (I,U) \to (K,W)$ of any two arrows $(P,P') : (I,U) \to (J,V)$ and $(Q,Q') : (J,V) \to (K,W)$ is given by the pair $(R : K \to I, R' : \prod_{k \in K} \mathbf{C}(U_{R(k)}, W_k))$ where :

$$R(k) = P(Q(k)) \text{ and } R'(k) = Q'(k) \circ P'(Q(k)) : U_{P(Q(k))} \to V_{Q(k)} \to W_k \in \mathbf{C}.$$

The identity arrow of any (I,U) is (P,P') where $P(i) = i$ and $P'(i) = \mathrm{id}_{P(i)} : P(i) \to P(i)$.

One may have recognised in this definition the construction for non-determinism of [11]. It happens to be the free cartesian completion of $\mathbf{C}$, the dual of $\mathrm{Fam}(\mathbf{C})$ construction for free coproduct completion [9, 3, 14], i.e. $\mathbb{O}\mathbf{C} = \mathrm{Fam}(\mathbf{C}^{\mathrm{op}})^{\mathrm{op}}$.

It is not hard to see that each object of this category has many isomorphic objects of bijective index set, so that the particular index set used is irrelevant. The real information contained in an equivalence class of isomorphic objects is the number of times each word occurs. Here, we allow this cardinality to be arbitrary. The issue of cardinality is a detail at this point, and we do not bother commenting on this issue before the conclusion. In the meantime, one can freely add the word *finite* anywhere one feels it is needed.

At this time, some notations are required to handle families and some relevant elements of $\mathbb{O}\mathbf{W}$. Given an arbitrary set U, we write $\overline{U}$ for the corresponding family containing each elements of U exactly once and given by the pair (U, id_U). Also, for any $(I,U) \in \mathbb{O}\mathbf{W}$, we consider the appropriate arrows $\mathrm{beg}_{(I,U)}, \mathrm{end}_{(I,U)} : \overline{\{\varepsilon\}} \to (I,U)$ identifying the occurrences of the empty words respectively at the beginning and at the end of the words of the family (I,U), and which are given by $\mathrm{beg}_{(I,U)} = ([i \mapsto \varepsilon], [i \mapsto (0 : \varepsilon \to U_i)])$ and $\mathrm{end}_{(I,U)} = ([i \mapsto \varepsilon], [i \mapsto (|U_i| : \varepsilon \to U_i)])$. We make use here of the notation $[x \mapsto f(x)]$ to specify succinctly an anonymous function; domains and codomains can always be retrieved from the context.

With the category $\mathbb{O}\mathbf{W}$ and these beginning and ending arrows, we obtain concatenation as wanted and in a very similar way to concatenation in $\mathbf{W}$ with Theorem 5.

▶ **Proposition 12.** *For any two families $(I,U),(J,V) \in \mathbb{O}\mathbf{W}$, a colimit of the diagram*

$$\begin{tikzcd}
(I,U) & & (J,V) \\
& \overline{\{\varepsilon\}} \arrow[ul, "\mathrm{end}_{(I,U)}"] \arrow[ur, "\mathrm{beg}_{(I,U)}"'] &
\end{tikzcd}$$

is given by the cocone

$$\begin{tikzcd}
& (I \times J, (i,j) \mapsto U_i \cdot V_j) & \\
(I,U) \arrow[ur, "\eta_1"'] & & (J,V) \arrow[ul, "\eta_2"] \\
& \overline{\{\varepsilon\}} \arrow[ul, "\mathrm{end}_{(I,U)}"] \arrow[ur, "\mathrm{beg}_{(I,U)}"'] \arrow[uu, "\eta_3"'] &
\end{tikzcd}$$

where

- $\eta_1 = ([(i,j) \mapsto i], [(i,j) \mapsto (0 : U_i \to U_i \cdot V_j)])$,
- $\eta_2 = ([(i,j) \mapsto j], [(i,j) \mapsto (|U_i| : V_j \to U_i \cdot V_j)])$, *and*
- $\eta_3 = ([(i,j) \mapsto \varepsilon], [(i,j) \mapsto (|U_i| : \varepsilon \to U_i \cdot V_j)])$.

Proof. Indeed, consider another cocone to some (K, W) with components, $(P, P') : (I, U) \to (K, W)$, $(Q, Q') : (J, V) \to (K, W)$, and $(R, R') : \overline{\{\varepsilon\}} \to (K, W)$. The mediating arrow is given by $([k \mapsto (P(k), Q(k))], [k \mapsto P'(k) : U_{P(k)} \cdot V_{Q(k)} \to W_k])$. Notice that this mediating follows the exact same definition as the one exhibited in Example 8. ◀

It is now time to summarize what we have just achieved. We took a diagram shape that allowed to obtain *deterministic* Lindenmayer systems as GT by encoding concatenation as colimit. Then, we changed the objects and arrows in this diagram to obtain what we can informally call *non-deterministic concatenation*. But the diagram shape itself arises from the *deterministic* decomposition of a single input word (see [4]). In other words, the pattern matching is not considered in $\mathbb{O}\mathbf{W}$ but in $\mathbf{W}$. So for now, we only have the following.

▶ **Definition 13.** *For any non-deterministic Lindenmayer system* $(\Sigma, \delta : \Sigma \to \mathrm{P}(\Sigma^*))$*, we write* $\mathbf{D} : \mathbf{W} \to \mathbb{O}\mathbf{W}$ *for the functor mapping words* $u \in \mathbf{W}$ *to* $\mathbf{D}(u) = (I, V)$ *where*

$$I := \delta(u_0) \times \ldots \times \delta(u_{|u|-1}) \text{ and } V_{(v_0,\ldots,v_{|u|-1})} := v_0 \cdot \ldots \cdot v_{|u|-1}$$

and arrows $p : u' \to u$ *to* $\mathbf{D}(p) = (P, P')$ *where*

$$P((v_0, \ldots, v_{|u|-1})) := (v_p, \ldots, v_{p+|u'|-1}) \text{ and}$$

$$P'((v_0, \ldots, v_{|u|-1})) := |v_0| + \ldots + |v_{p-1}| : v_p \cdot \ldots \cdot v_{p+|u'|-1} \to v_0 \cdot \ldots \cdot v_{|u|-1}.$$

The functor $\mathbf{D}$ is a categorical counterpart of $\Delta : \Sigma^* \to \mathrm{P}(\Sigma^*)$ of Definition 2 but with families making sure that we keep the multiple instances of each word. Indeed, for $\mathbf{D}(u) = (I, U)$, each index $(v_0, \ldots, v_{|u|-1}) \in I$ corresponds to a choice of a word v_i among the possibilities provided by $\delta(u_i)$, for each letter u_i of u. For such a choice, the associated resulting word $V_{(v_0,\ldots,v_{|u|-1})}$ is simply given by the concatenation of the v_i. The definition of $\mathbf{D}(p)$ expresses the monotony of Δ. The monotony can be illustrated as follows. Consider $u = \alpha_1 \cdot u' \cdot \alpha_2$ with $|\alpha_1| = p$. Taking $v' \in \Delta(u')$, $\gamma_1 \in \Delta(\alpha_1)$, and $\gamma_2 \in \Delta(\alpha_2)$, we have $v = \gamma_1 \cdot v' \cdot \gamma_2 \in \Delta(u)$. So we have an arrow $|\gamma_1| : v' \to v$. As a family of arrows, $\mathbf{D}(p)$ gathers all of these arrows. In the formula of Definition 13, we have $\alpha_1 = u_0 \ldots u_{p-1}$, $u' = u_p \ldots u_{p+|u'|-1}$, $\gamma_1 = v_0 \cdot \ldots \cdot v_{p-1}$, and $v' = v_p \cdot \ldots \cdot v_{p+|u'|-1}$.

Exactly as Δ is generated from its sole behavior on letters given by δ as stated by Definition 2, we will see that the functor $\mathbf{D}$ is generated from its restriction to the letters and ε. We start by defining the categorical counterpart $\mathbf{d}$ of δ.

▶ **Definition 14.** *For any non-deterministic Lindenmayer system* $(\Sigma, \delta : \Sigma \to \mathrm{P}\Sigma^*)$*, we write* $\mathbf{d} : \mathbf{W} \restriction \Sigma \cup \{\varepsilon\} \to \mathbb{O}\mathbf{W}$ *for the functor from the full subcategory* $\mathbf{W} \restriction \Sigma \cup \{\varepsilon\}$ *of* $\mathbf{W}$ *to* $\mathbb{O}\mathbf{W}$ *defined as* $\mathbf{d} = \mathbf{D} \restriction (\mathbf{W} \restriction \Sigma \cup \{\varepsilon\})$*.*

The functor $\mathbf{d}$ is entirely characterized in terms of arrows beg and end.

▶ **Lemma 15.** *For any* $a \in \Sigma$*, we have* $\mathbf{d}(0 : \varepsilon \to a) = \mathrm{beg}_{\mathbf{d}(a)}$ *and* $\mathbf{d}(1 : \varepsilon \to a) = \mathrm{end}_{\mathbf{d}(a)}$*.*

Proof. Consider $0 : \varepsilon \to a$. By Defs 14 and 13, $\mathbf{d}(\varepsilon) = \mathbf{D}(\varepsilon) = (\{\varepsilon\}, [\varepsilon \mapsto \varepsilon]) = \overline{\{\varepsilon\}}$, and $\mathbf{d}(a) = \mathbf{D}(a) = (\delta(a), [i \mapsto i])$. By the same definitions, $\mathbf{d}(0 : \varepsilon \to a) = \mathbf{D}(0 : \varepsilon \to a) = (P, P')$ with $P(i) = \varepsilon$ and $P'(i) = |\varepsilon| : \varepsilon \to i$ where i ranges over $\delta(a)$. Clearly, $\mathbf{d}(0 : \varepsilon \to a) = \mathrm{beg}_{(\delta(a),[i \mapsto i])} = \mathrm{beg}_{\mathbf{d}(a)}$ as expected. We get $\mathbf{d}(1 : \varepsilon \to a) = \mathrm{end}_{\mathbf{d}(a)}$ similarly. ◀

We can now establish that $\mathbf{D}$ is obtained as an extension of $\mathbf{d}$ thereby providing a categorical counterpart of Definition 2.

▶ **Proposition 16.** *$\mathbf{D}$ is a pointwise left Kan extension of $\mathbf{d}$ along the inclusion functor $\iota : \mathbf{W} \restriction \Sigma \cup \{\varepsilon\} \to \mathbf{W}$ as in the following diagram where η is the identity.*

$$\begin{tikzcd}
\mathbf{W} \arrow[drr, "\mathbf{D}"] & & \\
\mathbf{W} \restriction \Sigma \cup \{\varepsilon\} \arrow[u, hook, "\iota"] \arrow[rr, "\mathbf{d}"'] & {} \arrow[u, phantom, "\eta\Uparrow"] & \mathbb{O}\{\mathbf{W}\}
\end{tikzcd}$$

Proof. Using the explicit definition of pointwise left Kan extensions in terms of colimit, we are left to show that $\mathbf{D}(-) = \mathrm{Colim}(\mathbf{d} \circ \mathrm{Proj}[\iota/-])$. In [4], it is already proved that the diagram $\mathrm{Proj}[\iota/u]$ has the following zigzag shape:

$$\begin{tikzcd}
& u_0 & & u_1 & & \cdots & & u_{|u|-1} & \\
\varepsilon \arrow[ur, "0"'] & & \varepsilon \arrow[ul, "1"] \arrow[ur, "0"'] & & \varepsilon \arrow[ul, "1"] \arrow[ur, "0"'] & & \varepsilon \arrow[ul, "1"] \arrow[ur, "0"'] & & \varepsilon \arrow[ul, "1"]
\end{tikzcd}$$

Using Lemma 15, the diagram $\mathbf{d} \circ \mathrm{Proj}[\iota/u]$ is:

$$\begin{tikzcd}
& (\delta(u_0), [i \mapsto i]) & & (\delta(u_1), [i \mapsto i]) & & \cdots & & (\delta(u_{|u|-1}), [i \mapsto i]) & \\
\overline{\{\varepsilon\}} \arrow[ur, "\mathrm{beg}"'] & & \overline{\{\varepsilon\}} \arrow[ul, "\mathrm{end}"] \arrow[ur, "\mathrm{beg}"'] & & \overline{\{\varepsilon\}} \arrow[ul, "\mathrm{end}"] \arrow[ur, "\mathrm{beg}"'] & & \overline{\{\varepsilon\}} \arrow[ul, "\mathrm{end}"] \arrow[ur, "\mathrm{beg}"'] & & \overline{\{\varepsilon\}} \arrow[ul, "\mathrm{end}"]
\end{tikzcd} .$$

Iteratively using Prop. 12 on this finite sequence, the colimit of this diagram is clearly the non-deterministic concatenation of the $(\delta(u_k), [i \mapsto i])$, $0 \leq k < |u|$, which is also the definition of $\mathbf{D}(u)$ as given in Def. 13. To prove that η is the identity, it is enough to consider the particular case of the previous reasoning with $|u| \leq 1$ that shows that $(\mathbf{D} \circ \iota)(u) = \mathbf{d}(u)$.

Given some $p : u' \to u$, for proving that $\mathbf{D}(p) = \mathrm{Colim}(\mathbf{d} \circ \mathrm{Proj}[\iota/p])$ we remark that $\mathbf{D}(p)$ has to be a mediating arrow. By unicity of the mediating arrow, it remains to show that $\mathbf{D}(p)$ obeys the requested commutations of mediating arrows, which is straightforward. ◀

So far, for a non-deterministic Lindenmayer system (Σ, δ), we have $\mathbf{d}$ as a categorical counterpart of δ, which gives rise by left Kan extension to $\mathbf{D}$, the categorical counterpart of Δ. However, we still do not have a dynamical system, since the domain and codomain of $\mathbf{D} : \mathbf{W} \to \mathbb{O}\mathbf{W}$ are not strictly the same. In other words, we now want a left Kan extension counterpart of $\overline{\Delta} : \mathrm{P}(\Sigma^*) \to \mathrm{P}(\Sigma^*)$ of Definition 2, say $\overline{\mathbf{D}} : \mathbb{O}\mathbf{W} \to \mathbb{O}\mathbf{W}$. Clearly, we already know the expected definition of $\overline{\mathbf{D}}$ since we want to apply independently $\mathbf{D}$ on each element of a family (I, U) and to flatten the results altogether.

▶ **Definition 17.** *Let $\overline{\mathbf{D}} : \mathbb{O}\mathbf{W} \to \mathbb{O}\mathbf{W}$ be the functor defined as*

$$\overline{\mathbf{D}}((I,U)) = \left(\bigcup_{i \in I} (\{i\} \times J_i) \, , \, [(i,j) \mapsto (V_i)_j] \right) \ \textit{where } (J_i, V_i) = \mathbf{D}(U_i),$$

and $\overline{\mathbf{D}}((P,P') : (I',U') \to (I,U)) = (Q,Q')$ such that, for each $(i,j) \in \bigcup_{i \in I} (\{i\} \times J_i)$:

$$Q((i,j)) = (P(i), R_i(j)) \ \textit{and } Q'((i,j)) = R'_i(j) \ \textit{where } (R_i, R'_i) = \mathbf{D}(P'(i)).$$

Obtaining $\overline{\mathbf{D}}$ as a Kan extension consists in embedding $\mathbf{W}$ into $\mathbb{O}\mathbf{W}$, then extending along this embedding. The notation $\overline{U}$ that we have introduced earlier can be turned into a *singleton functor* for defining this embedding.

▶ **Definition 18.** *For any category $\mathbf{C}$, the singleton functor $\mathrm{sing}_{\mathbf{C}} : \mathbf{C} \to \mathbb{O}\mathbf{C}$ is defined as*

$$\mathrm{sing}_{\mathbf{C}}(x) = \overline{\{x\}} \text{ and } \mathrm{sing}_{\mathbf{C}}(f : x \to y) = ([y \to x], [y \mapsto (f : x \to y)]).$$

Unfortunately, the program stops here since $\overline{\mathbf{D}}$ fails to be the extension of $\mathbf{d}$ along $\mathrm{sing}_{\mathbf{W}} \circ \iota : \mathbf{W} \restriction \Sigma \cup \{\varepsilon\} \to \mathbf{W} \hookrightarrow \mathbb{O}\mathbf{W}$. In fact, the arrows of $\mathbb{O}\mathbf{W}$ are not well-suited for decomposing a family of words in the appropriate way for the expected concatenation. For instance, consider the family in inputs $\overline{\{\mathtt{aa}, \mathtt{bb}\}}$. The comma category $\mathrm{sing}_{\mathbf{W}} \circ \iota/\overline{\{\mathtt{aa}, \mathtt{bb}\}}$ fails to identify the occurrences of $\mathtt{a}$ in this family. Indeed, an arrow from $\overline{\{\mathtt{a}\}}$ to $\overline{\{\mathtt{aa}, \mathtt{bb}\}}$ requires to identify an occurrence of $\mathtt{a}$ in $\mathtt{bb}$ but there is none. So there is no arrow between those two families and the diagram $\mathbf{d} \circ \mathrm{Proj}[\mathrm{sing}_{\mathbf{W}} \circ \iota/\overline{\{\mathtt{a}, \mathtt{b}\}}]$ of Equation (1) contains only ε's and does not exhibit the expected zigzag shape.

5 The Kleisli 2-Category of the 2-Monad of Families

As explained in Section 1, we propose to solve the last issue by placing oneself in a 2-categorical context. Since this solution can seem more elaborate than necessary, let us make precise why this transition to 2-categories is conceptually natural with respect to our goal.

Let us develop the relation between dynamical systems and their non-deterministic counterparts. At a general level of description, dynamical systems can be defined once we have a collection of objects to model the states, and a way to specify endo-functions on these objects to model the dynamics. For instance, in the category of sets and functions, the states are modeled as a set and the dynamics as a function; the usual case of (deterministic) dynamical systems is captured. But in the category of topological spaces and continuous functions, states are modeled as a topological space and the dynamics by a continuous function, allowing to handle the so-called topological dynamical systems. Similarly, in the category of sets and relations, states are modeled as a set and the dynamics by a relation. This last case is particularly interesting for our concern since it is the place to deal with non-deterministic dynamical systems. Formally this latter category is equivalently described as the *Kleisli category* of the so-called *powerset monad*. This is based on the fact that $R \subseteq X \times Y$ is equivalently a function $f : X \to \mathrm{P}(Y)$, that singletons allow any set X to be seen as included in $\mathrm{P}(X)$, and that unions allow any sets of sets in $\mathrm{P}(\mathrm{P}(X))$ to be simplified in a simple set of $\mathrm{P}(X)$. The two lessons we learn here are that (1) dynamical systems are parametrized by the nature of the objects and the arrows they rely on, and that (2) the parametrization for the non-determinitic counterpart is based on the powerset monad.

We now proceed to apply the same scheme for the GT. The difference with dynamical systems is that GT are not defined with two layers (an object for the states and an arrow for the dynamics) but with three layers: categories, functors and natural transformations as it can be seen in the pointwise left Kan extension diagram of Section 2.1. So they are parametrized by a 2-category. For instance, the simple GT as defined in Definition 1 are parametrized by **Cat**, the 2-category of categories. Following the second lesson on non-deterministic dynamical systems, for the particular case of non-deterministic GT, we propose this 2-category parameter to be set to the Kleisli 2-category induced by the 2-monad of families.

We already have all the ingredients of a 2-monad on **Cat** as we now proceed to show. Firstly, the construction $\mathbb{O}\mathbf{C}$ of Definition 11 can be extended to act on functors and natural transformations and yields a 2-functor $\mathbb{O} : \mathbf{Cat} \to \mathbf{Cat}$.

▶ **Definition 19.** *For any functor $F : \mathbf{C} \to \mathbf{C}'$, the functor $\mathbb{O}F : \mathbb{O}\mathbf{C} \to \mathbb{O}\mathbf{C}'$ is defined as $\mathbb{O}F((I, U)) = (I, F \circ U)$, and $\mathbb{O}F((P, P') : (I, U) \to (J, V)) = (P, F \circ P')$. For any natural transformation $\alpha : F \Longrightarrow G : \mathbf{C} \to \mathbf{C}'$, the natural transformation $\mathbb{O}\alpha : \mathbb{O}F \Longrightarrow \mathbb{O}G : \mathbb{O}\mathbf{C} \to \mathbb{O}\mathbf{C}'$ has components $(\mathbb{O}\alpha)_{(I,U)} = ([i \mapsto i], [i \mapsto \alpha_{U_i}]) : (I, F \circ U) \to (I, G \circ U)$.*

To make the 2-functor $\mathbb{O}$ into a 2-monad, we need to consider the obvious pairs of operations, the first one, $\mathrm{sing}_{\mathbf{C}} : \mathbf{C} \to \mathbb{O}\mathbf{C}$, lifting an object of a category $\mathbf{C}$ to a singleton family of $\mathbb{O}\mathbf{C}$, and the second one, $\mu_{\mathbf{C}} : \mathbb{O}\mathbb{O}\mathbf{C} \to \mathbb{O}\mathbf{C}$, flattening a family of families of objects into a simple family of objects. Notice that this last construction has already been encountered in Definition 17 of $\overline{\mathbf{D}}$, whose role of flattening has also been underlined above. In particular, the function $\overline{\mathbf{D}}$ is in fact obtained as $\mu_{\mathbf{W}} \circ \mathbb{O}\mathbf{D}$ from Definition 13. The two operations form indeed a 2-monad.

▶ **Proposition 20.** *Operations* $\mathrm{sing}\_$ *and* $\mu\_$ *make* $\mathbb{O} : \mathbf{Cat} \to \mathbf{Cat}$ *into a 2-monad, i.e., all instances of the following diagrams weakly commute.*

$$\begin{tikzcd}
\mathbb{O}\mathbb{O}\mathbb{O}\mathbf{C} \arrow[r, "\mathbb{O}\mu_{\mathbf{C}}"] \arrow[d, "\mu_{\mathbb{O}\mathbf{C}}"'] & \mathbb{O}\mathbb{O}\mathbf{C} \arrow[d, "\mu_{\mathbf{C}}"] \\
\mathbb{O}\mathbb{O}\mathbf{C} \arrow[r, "\mu_{\mathbf{C}}"'] \arrow[ur, phantom, "\overset{\sim}{\Nearrow}"] & \mathbb{O}\mathbf{C}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{O}\mathbb{O}\mathbf{C} \arrow[dr, "\mu_{\mathbf{C}}"'] & \mathbb{O}\mathbf{C} \arrow[l, "\mathrm{sing}_{\mathbb{O}\mathbf{C}}"'] \arrow[r, "\mathbb{O}(\mathrm{sing}_{\mathbf{C}})"] \arrow[d, "\mathrm{id}_{\mathbb{O}\mathbf{C}}"] \arrow[dl, phantom, "\overset{\sim}{\Longrightarrow}", near start] \arrow[dr, phantom, "\overset{\sim}{\Longleftarrow}", near start] & \mathbb{O}\mathbb{O}\mathbf{C} \arrow[dl, "\mu_{\mathbf{C}}"] \\
& \mathbb{O}\mathbf{C} &
\end{tikzcd}$$

Proof. For the first square, and given an object $(I, U) \in \mathbb{O}\mathbb{O}\mathbb{O}\mathbf{C}$, the top-right path leads to index set of the form $(i, (j, k))$ for the object in $\mathbb{O}\mathbf{C}$ while the left-bottom path leads to the form $((i, j), k)$, hence the weak commutation. For the triangles on the right, an object $(I, U) \in \mathbb{O}\mathbf{C}$ sees each index $i \in I$ transformed into $((I, U), i) \in \{(I, U)\} \times I$ by the left path and into $(i, U_i) \in \{(i, U_i) \mid i \in I\}$ by the right path. ◀

In order to ease the reading of elements of the Kleisli weak 2-category, let us introduce some notations. We write $\widetilde{F} : \mathbf{C} \mathrel{-\!\!\circ\!\!\rightarrow} \mathbf{D}$ to represent a functor $F : \mathbf{C} \to \mathbb{O}\mathbf{D}$ of the Kleisli weak 2-category. A 2-arrow $\widetilde{\eta} : \widetilde{F} \mathrel{=\!\!\circ\!\!\Rightarrow} \widetilde{G}$ stands simply for a natural transformation $\eta : F \Longrightarrow G$. The composition of arrows in the Kleisli weak 2-category, written $\widetilde{G} \circ \widetilde{F} : \mathbf{C} \mathrel{-\!\!\circ\!\!\rightarrow} \mathbf{D} \mathrel{-\!\!\circ\!\!\rightarrow} \mathbf{E}$, is the functor $\mu_{\mathbf{E}} \circ \mathbb{O}G \circ F : \mathbf{C} \to \mathbb{O}\mathbf{D} \to \mathbb{O}\mathbb{O}\mathbf{E} \to \mathbb{O}\mathbf{E}$.

We finally reach our initial goal as we are now able to show that the diagram of Prop. 16 is in fact a summary of a GT in the Kleisli weak 2-category induced by the 2-monad $\mathbb{O}$. More accurately, considering the GT diagram of the rule system $\langle \mathbf{W} \restriction \Sigma \cup \{\varepsilon\}, \widetilde{\mathrm{sing}_{\mathbf{W}} \circ \iota}, \widetilde{\mathbf{d}} \rangle$ is completely equivalent to considering the diagram of Prop. 16, achieving the fact that the initial non-determinitic Lindenmayer system is indeed a GT. Moreover, this works for any non-deterministic rule system $\langle \mathbf{\Gamma}, \widetilde{\mathrm{sing}_{\mathbf{C}} \circ \mathrm{L}}, \widetilde{\mathrm{R}} \rangle$ on any category $\mathbf{C}$. Notice the particular form of the l.h.s. functor defined using $\mathrm{L} : \Gamma \to \mathbf{C}$ which is still required to be a full embedding.

▶ **Theorem 21.** *Let* $\widetilde{T} = \langle \mathbf{\Gamma}_T, \widetilde{\mathrm{sing}_{\mathbf{C}} \circ \mathrm{L}_T}, \widetilde{\mathrm{R}_T} \rangle$ *be a rule system in the Kleisli weak 2-category induced by the 2-monad* $\mathbb{O}$. $\widetilde{T}$ *is a GT iff* T *is the left Kan extension of* R_T *along* L_T *in the 2-category* $\mathbf{Cat}$.

Proof. The rule system being a GT, we have a pair $\langle \widetilde{T}, \widetilde{\eta} : \widetilde{\mathrm{R}_T} \Longrightarrow \widetilde{T} \circ \widetilde{\mathrm{sing}_{\mathbf{C}} \circ \mathrm{L}_T} \rangle$ which is a left Kan extension in the Kleisli 2-category and takes the following diagrammatic form for any other pair $\langle \widetilde{K}, \widetilde{\rho} : \widetilde{\mathrm{R}_T} \Longrightarrow \widetilde{K} \circ \widetilde{\mathrm{sing}_{\mathbf{C}} \circ \mathrm{L}_T} \rangle$:

$$\begin{tikzcd}[column sep=huge]
\mathbf{C} \arrow[rrd, bend left, "\widetilde{K}"] \arrow[rrd, "\widetilde{T}"'] & \widetilde{\lambda} \Uparrow & \\
\mathbf{\Gamma} \arrow[u, "\widetilde{\mathrm{sing}_{\mathbf{C}} \circ \mathrm{L}_T}"] \arrow[rr, "\widetilde{\mathrm{R}_T}"'] & \widetilde{\rho} \Uparrow \quad \widetilde{\eta} \Uparrow & \mathbf{C}
\end{tikzcd}$$

This diagram corresponds by definition to the diagram in **Cat** on the left below. By naturality of sing and Prop. 20, it (weakly) simplifies into the expected diagram, on the right below.

$$\begin{tikzcd}
\mathbb{O}\mathbf{C} \arrow[rrd, "\mathbb{O}T"] \arrow[rrd, bend left, "\mathbb{O}K"] \arrow[rrd, phantom, near start, "\Uparrow \mathbb{O}\lambda \Uparrow"] & & \\
\mathbf{C} \arrow[u, "\mathrm{sing}_{\mathbf{C}}"] \arrow[rr, phantom, "\rho \Uparrow \quad \eta \Uparrow"] & & \mathbb{O}\mathbb{O}\mathbf{C} \arrow[d, "\mu_{\mathbf{C}}"] \\
\mathbf{\Gamma} \arrow[u, "\mathrm{L}_T"] \arrow[rr, "\mathrm{R}_T"'] & & \mathbb{O}\mathbf{C}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{C} \arrow[rrd, "T"] \arrow[rrd, bend left, "K"] \arrow[rrd, phantom, near start, "\lambda \Uparrow"] & & \\
\mathbf{\Gamma} \arrow[u, "\mathrm{L}_T"] \arrow[rr, "\mathrm{R}_T"'] \arrow[rr, phantom, shift left=3, "\rho \Uparrow \quad \eta \Uparrow"] & & \mathbb{O}\mathbf{C}
\end{tikzcd}$$

◀

6 Final Discussion

This journey started with the general goal of going toward non-deterministic, probabilistic and quantum GT. Starting with the first concrete step of capturing non-deterministic Lindenmayer systems as GT, we guessed some constructions based on the deterministic case. The result of these guesses did not have the precise form of a GT in the 2-category of categories, functors and natural transformations. But we showed they correspond in fact to a GT in another (weak) 2-category. The latter is induced by Kleisli's construction on a particular (weak) 2-monad that we made explicit. This is to be expected since the same architecture happens for non-deterministic dynamical systems. Indeed, they are also dynamical systems in another category, with the latter being induced by a monad. All in all, we ended with a general solution for mixing non-determinism with locality as described in the global transformation framework.

Along the way, we mentioned a few technicalities on which we now come back. The first one is the Conjecture 10 on which we want to add a comment. If it is wrong, then families can be simplified into sets. But in this case, there should be a relation between the family-based solution just presented and this new set-based solution. But thinking of this relation as a functor from families to sets reinforce us in the belief that the conjecture is true.

The second technicality is about the size of the families considered, which is related to the size issues for the "2-category of categories". For most practical purpose, it is possible to restrict oneself to finite families. In this case, a small category $\mathbf{C}$ leads to a small category $\mathbb{O}\mathbf{C}$. In this case, the 2-functor $\mathbb{O}$ is indeed an endomorphism of the 2-category of *small* categories. Dropping the finiteness constraint though, one then considers a 2-functor $\mathbb{O}\mathbf{C}$ from *small* categories to *large* categories. This is however perfectly fine, since this describes a so-called *relative pseudomonad* with an associate Kleisli's construction, as defined in [8].

In [6], one can find a direct account of the 2-category described here in terms of 2-monad. In particular, the *open* functors and *open* natural transformations are introduced using presheaves and proved to form a weak 2-category. More precisely, an open functor F from a category $\mathbf{C}$ to a category $\mathbf{D}$ is the data of a presheaf on $\mathbf{C}$ together with a functor from the category of elements of that presheaf to $\mathbf{D}$. An interesting feature of this presheaf presentation is that it allows to speak directly about special properties arising from the association of locality and non-determinism. For instance, correlations and intrications correspond to obstructions of the presheaf to be a sheaf. The formal definition in [6] can be made easier to manipulate by the use of discrete fibrations instead of categories of elements through the so-called Grothendieck construction, and doing so presents this bicategory as a particular bicategory of spans. Moreover, this bicategory is a sub-bicategory of the bicategory of profunctors (*a.k.a.* distributors). Notice that the many presentations of this bicategory are strongly related to the many possible presentations one can have of the notion of "relation": powerset monad (as in this paper), spans, and characteristic functions of the relation.

References

1 Enrico Biermann, Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Gabriele Taentzer. Parallel independence of amalgamated graph transformations applied to model transformation. In *Graph transformations and model-driven engineering*, pages 121–140. Springer, 2010.

2 Paul Boehm, Harald-Reto Fonio, and Annegret Habel. Amalgamation of graph transformations: a synchronization mechanism. *Journal of Computer and System Sciences*, 34(2-3):377–408, 1987.

3 Francis Borceux and George Janelidze. *Galois theories*, volume 72. Cambridge University Press, 2001.

4 Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. Lindenmayer systems and global transformations. In Ian McQuillan and Shinnosuke Seki, editors, *Unconventional Computation and Natural Computation – 18th International Conference, UCNC 2019, Tokyo, Japan, June 3-7, 2019, Proceedings*, volume 11493 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2019. doi:10.1007/978-3-030-19311-9_7.

5 Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. Accretive computation of global transformations. In *International Conference on Relational and Algebraic Methods in Computer Science*, pages 159–175. Springer, 2021.

6 Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. The bicategory of open functors, 2021. arXiv:2102.08051.

7 Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. Cellular automata and kan extensions. *arXiv preprint*, 2021. arXiv:2102.12156.

8 Marcelo Fiore, Nicola Gambino, Martin Hyland, and Glynn Winskel. Relative pseudomonads, kleisli bicategories, and substitution monoidal structures. *Selecta Mathematica*, 24(3):2791–2830, 2018.

9 Hongde Hu and Walter Tholen. Limits in free coproduct completions. *Journal of Pure and Applied Algebra*, 105(3):277–291, 1995. doi:10.1016/0022-4049(94)00153-7.

10 Karel Klouda and Štěpán Starosta. Characterization of circular d0l-systems. *Theoretical Computer Science*, 790:131–137, 2019. doi:10.1016/j.tcs.2019.04.021.

11 Daniel Lehmann. *Categories for fixpoint semantics*. PhD thesis, University of Warwick, 1976.

12 Luidnel Maignan and Antoine Spicher. Global graph transformations. In Detlef Plump, editor, *Proceedings of the 6th International Workshop on Graph Computation Models co-located with the 8th International Conference on Graph Transformation (ICGT 2015) part of the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences, L'Aquila, Italy, July 20, 2015*, volume 1403 of *CEUR Workshop Proceedings*, pages 34–49. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1403/paper4.pdf.

13 Grzegorz Rozenberg and Arto Salomaa. *The mathematical theory of L systems*, volume 90. Academic press, 1980.

14 Karthik Yegnesh. The fundamental infinity-groupoid of a parametrized family. *arXiv preprint*, 2017. arXiv:1701.02250.

A Robust Class of Languages of 2-Nested Words

Séverine Fratani
LIS, Aix-Marseille Univ, CNRS, Marseille, France

Guillaume Maurras
LIS, Aix-Marseille Univ, CNRS, Marseille, France

Pierre-Alain Reynier
LIS, Aix-Marseille Univ, CNRS, Marseille, France

Abstract

Regular nested word languages (a.k.a. visibly pushdown languages) strictly extend regular word languages, while preserving their main closure and decidability properties. Previous works have shown that considering languages of 2-nested words, *i.e.* words enriched with two matchings (a.k.a. 2-visibly pushdown languages), is not as successful: the corresponding model of automata is not closed under determinization. In this work, inspired by homomorphic representations of indexed languages, we identify a subclass of 2-nested words, which we call 2-wave words. This class strictly extends the class of nested words, while preserving its main properties. More precisely, we prove closure under determinization of the corresponding automaton model, we provide a logical characterization of the recognized languages, and show that the corresponding graphs have bounded treewidth. As a consequence, we derive important closure and decidability properties. Last, we show that the word projections of the languages we define belong to the class of linear indexed languages.

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

Keywords and phrases Nested word, Determinization, Indexed languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.50

Funding Partly supported by the DeLTA ANR project (ANR-16-CE40-0007)

1 Introduction

The class of regular languages constitutes a cornerstone of theoretical computer science, thanks to its numerous closure and decidability properties. A long line of research studied extensions of this class, while preserving its robustness. Context free languages (CFL for short) constitute a very important class: they admit multiple presentations, by means of pushdown automata, context-free grammars and more, and have led to numerous applications. Unfortunately, CFL do not satisfy several important properties enjoyed by regular languages. More precisely, the corresponding automaton model, namely pushdown automata, does not admit determinization. In addition, the class of CFL is not closed under intersection nor complement, and universality, inclusion and equivalence are undecidable properties.

A simple way to patch this is by considering as input the word together with the inherent matching relation, resulting in what is known as a *nested word* [3]. Indeed, as soon as a word belongs to a CFL, one can identify a matching relation on (some of) the positions of the word, whatever the presentation of the CFL. For instance, if the CFL is given as a pushdown automaton, then this relation associates push/pop positions. Another way to define the matching relation is to use an alternative presentation of CFL given in [16], which refines the Chomsky-Schützenberger theorem. Following this work, one can show that a language L is a CFL iff there exists a regular language R, a Dyck language D_2 over two pairs of brackets, and two homomorphisms h (h is non-erasing), g such that $L = h(g^{-1}(D_2) \cap R)$. This alternative presentation also leads to a natural matching relation, induced by D_2. The model of *nested* *word automata* naturally extends finite-state automata by allowing to label edges of the matching relation with states (often called hierarchical). This model accepts the so-called class of *regular nested word languages*, allowing to recover most of the nice properties of regular languages. More precisely, nested word automata can always be determinized. The class of regular nested word languages is closed under all the boolean operations, admits an equivalent presentation by means of logic (monadic-second order logic with a binary predicate corresponding to the matching relation), and expected decidability properties (emptiness, universality, inclusion and equivalence). It is worth noticing that another way to present this class is by splitting the alphabet into call/return/internal symbols. This leads to so-called *visibly pushdown languages* [2], and the associated model of visibly pushdown automata.

Several works tried to extend the class of regular nested word languages while preserving its closure and decidability properties. In particular, a focus has been put on words with multiple matching relations. In [19], the authors consider multiple stacks with a semantical restriction of push/pop operations known as k phases. That way, they obtain decidability of the emptiness problem. However, their model of automata cannot be determinized. In [6], the authors consider visibly pushdown automata with multiple stacks, with an ordering on stacks, and prove the closure under complement of their model. Their proof does not rely on the determinization of the model: indeed, as shown in [19], this class cannot be determinized in general. This corrects a previous result published in [5], which states that the general class of 2-stack visibly pushdown automata is closed under complementation, which does not hold, as shown in [4]. The crux in their proof was the use of determinization of 2-stack visibly pushdown automata. In [4], Bollig studies 2-stack visibly pushdown automata in their unrestricted form. He proves the equivalence with the existential fragment of monadic second-order logic, but that quantifier alternation leads to an infinite hierarchy in this setting. As a corollary, the resulting class of languages is not closed under complementation. Another restriction, known as scope-bounded pushdown languages, has been introduced in [20, 21], for which the authors manage to prove that the automaton model can be determinized.

The previous survey of related works shows the difficulty in identifying a class of 2-nested words for which the corresponding automaton model can be determinized. As a consequence, the decidability results presented in these papers require ad-hoc involved proofs. Graphs of bounded treewidth constitute an alternative approach for obtaining decidability properties. Indeed, in [13], the authors show that most of the previous classes with good decidability properties actually correspond to graphs of bounded treewidth, for which MSO decidability follows from [7, 17]. Yet, determinization of nested word automata is the keystone of the nice properties of this model, and thus constitutes a highly desirable feature. In the present work, taking inspiration in indexed languages, we identify a class of 2-nested words for which automata can be determinized. Our class is incomparable with those of [19], [6] and [20, 21]. Intuitively, between two matched positions of the first matching, they bound the number of switches between matchings, while we do not. In addition, the proof of determinizability of [20, 21] is different from ours, as their proof is a kind of superviser that uses determinization of [3] as a subroutine, while ours generalizes the construction of [3] to two nestings.

Indexed languages [1] correspond to the level 2 of the infinite hierarchy of higher-order languages [14]. With numerous applications in computational linguistics, they have been much studied during the seventies and the eighties [15, 8, 9]. Homomorphic characterizations of CFL that we presented before have been extended to (linear) indexed languages in several works, including [22, 10]. One of them (see [10]) shows that L is an indexed language iff there exists a regular language R, two Dyck languages D_2 and D_k over two and k pairs of brackets respectively, and two homomorphisms h, g such that $L = h(g^{-1}(D_2) \cap R \cap D_k)$, with some additional conditions on g. This presentation allows to associate with a word $w \in L$ two matchings, induced by D_2 and D_k, which interact in a very particular way. Graphically speaking, this interaction yields kinds of *waves* (see Figure 1). When restricted to linear indexed languages, the length of these waves is upper bounded by 2, yielding the structure of 2-waves that will be of interest to us in this paper. All these notions will be formally presented in the paper. We also refer the reader to Section 7, in which we explore the relationship between our work and (linear) indexed languages.

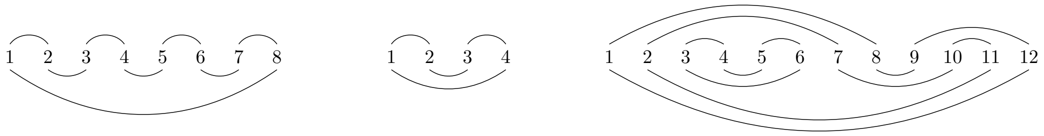


Figure 1 A 4-wave (left), a 2-wave (middle), and a combination of 2-waves (right).

In this paper, we consider 2-nested words whose matchings satisfy the structural restriction of 2-waves; we call them 2-wave words. The main result of this paper is to show that 2-nested word automata are closed under determinization on the class of 2-wave words. While determinization of nested word automata extends the well-known powerset construction with a reference state, our construction is more involved in order to be able to reconcile the labels of the different arches, and we give a detailed proof of correction. This allows us to prove the equivalence of automata with monadic-second order logic on 2-wave words, as well as with its existential fragment. This contrasts with results of [4] which show that on 2-nested words, quantifier alternation yields an infinite hierarchy. We also prove that the graphs associated with 2-wave words are MSO-definable, and have bounded treewidth. As a consequence, we obtain the following decidability results for the class of 2-wave words: satisfiability of MSO, emptiness of automata (in polynomial time), universality, inclusion and equivalence of automata (in exponential time).

In Section 2, we introduce the definitions of matchings and nested words, and provide a grammar for 2-wave words. In Section 3, we introduce the automaton model, and present in Section 4 the main result of the paper: the closure under determinization over 2-wave words. Applications to logic and decidability are presented in Sections 5 and 6. Last, a discussion on relations with (linear) indexed languages is given in Section 7.

2 Words and matchings

Words and relations. For any positive integer n, $[n] = \{1, \ldots, n\}$ is the set of all positive integers ranging from 1 to n. When referencing a position in a word, integers might be called *positions* or *index*. Σ denotes a finite alphabet. The empty word is denoted ϵ, and the set of finite words on Σ is denoted Σ^*. The length of $w \in \Sigma^*$ is denoted $|w|$. Given a non-empty word $w \in \Sigma^*$, its positions are numbered from 1 to $|w|$. We say that u is a *factor* of w if there exist two words x, y such that $w = xuy$. Given an interval $I \subseteq [|w|]$, we denote by $w_{|I}$ the factor of w corresponding to positions in I. Unless specified otherwise, words and automata introduced in this paper are defined on Σ.

▶ **Definition 1.** *A* matching relation *of length* $n \geq 0$ *is a binary relation* M *on* $[n]$ *such that:*
1. *if* $M(i, j)$ *then* $i < j$*, i.e.* M *is compatible with the natural order on integers*
2. *if* $M(i, j)$ *and* $M(k, l)$ *then* $\{i, j\} \cap \{k, l\} \neq \emptyset \implies i = k \wedge j = l$*, i.e. any integer is related at most once by the matching*
3. *if* $M(i, j)$ *and* $M(k, l)$ *then* $i < k < j \implies l < j$ *i.e. a matching is non-crossing.*

Since a matching is an injective functional relation, we will often use functional notations: $M(i) = j$ or $M^{-1}(j) = i$ rather than $M(i, j)$. If $M(i, j)$, we call i a *call* position, j a *return* position and if $k \in [n]$ is neither a call nor a return position we call it an *internal* position.

If I is a subset of $[n]$ we denote by I^c the subset of call positions of I, and by I^r the subset of return positions and I^{int} the subset of internal position. We say that I is *without pending arch* (wpa) if it has no pending call nor pending return, *i.e.* $M(I^c) \cup M^{-1}(I^r) \subseteq I$. Note that this definition holds when I is an interval, but even for an arbitrary subset of $[n]$. In particular, we say a pair (I_1, I_2) of intervals is without pending arch if $I_1 \cup I_2$ is. In this case, we may also say that (I_1, I_2) is a pair of *matched intervals.*

(2-)Nested words. We first recall the classical definition of nested words:

▶ **Definition 2.** *A* nested word *on* Σ *is a pair* $\omega = (w, M)$ *where* $w \in \Sigma^*$ *and* M *is a matching of length* $|w|$*. We write* $\mathsf{NW}(\Sigma)$ *the set of nested words on* Σ*, and* $\mathsf{NWL}(\Sigma)$ *the set of languages of nested words.*

Words equipped with two matchings are called 2-nested words:

▶ **Definition 3.** *A* 2-nested word *on* Σ *is a triple* $\omega = (w, M_1, M_2)$ *where* $w \in \Sigma^*$ *and* M_1*,* M_2 *are matchings of length* $|w|$*. We write* $\mathsf{2NW}(\Sigma)$ *for the set of 2-nested words on* Σ*, and* $\mathsf{2NWL}(\Sigma)$ *for the set of languages of 2-nested words.*

▶ **Example 4.** An example of a nested word (resp. two examples of 2-nested words) is depicted on the left (resp. on the middle and right) of Figure 2. For 2-nested words, the matching M_1 is depicted above, while matching M_2 is depicted below.

Given a 2-nested word $\omega = (w, M_1, M_2)$ and an interval $I \subseteq [|w|]$ which is wpa w.r.t. both M_1 and M_2, we denote by $\omega_{|I}$ the 2-nested word consisting of $w_{|I}$ and of the two matchings M_1' and M_2' obtained from M_1 and M_2 by restricting them to I, and then shifting them to the interval $[|I|]$. It is routine to verify that if ω is a 2-nested word, then so is $\omega_{|I}$.

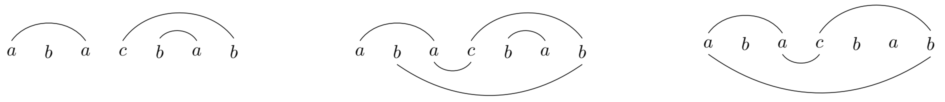


Figure 2 A nested word (left) and two 2-nested words (middle and right).

2-Waves and 2-wave words. In the sequel, we introduce the restriction of 2-nested words on which we will focus. Intuitively, wave structures are graphs obtained from the two matchings consisting of cycles alternating M_1-arches and M_2-arches whose shape evokes waves.

▶ **Definition 5.** *Let* n *be an integer, and* (M_1, M_2) *be a pair of matching relations of length* n*. A sequence of* 4 *integers* $1 \leq i_1 < i_2 < i_3 < i_4 \leq n$ *is a* 2-wave *if the following holds:*

- $M_1(i_1, i_2)$ *and* $M_1(i_3, i_4)$ *(top arches),*
- $M_2(i_2, i_3)$ *(bottom arch), and* $M_2(i_1, i_4)$ *(support arch)*

A pair (M_1, M_2) *is a* 2-wave structure *if any arch in* $M_1 \cup M_2$ *belongs to a 2-wave.*

▶ Remark 6. One could allow 2-wave structures to admit 1-waves, *i.e.* pairs of indices (i_1, i_2) with $i_1 < i_2$, $M_1(i_1, i_2)$ and $M_2(i_1, i_2)$. All our results would also hold for this generalization. However, in order to simplify the presentation of the paper, we do not consider them in this extended abstract.

▶ **Definition 7.** *A* 2*-wave word is a* 2*-nested word* $\omega = (w, M_1, M_2)$ *such that* (M_1, M_2) *is a* 2*-wave structure. We denote by* $\mathsf{WW}_2(\Sigma)$ *the set of* 2*-wave words over the alphabet* Σ.

▶ **Example 8.** Examples of waves are given on Figure 1. Let us consider the 2-nested words depicted on Figure 2. The one on the middle is not a 2-wave word (the upper arch $(5, 6)$ does not belong to a 2-wave), while the one on the right is.

Grammar. In order to proceed with structural induction, we present an inductive presentation of 2-wave words based on multiple context-free grammars (MCFG for short [18]). To this end, we turn to 2-visibly pushdown languages: the alphabet Σ is duplicated into five copies Σ^c_c, Σ^c_r, Σ^r_c, Σ^r_r and $\Sigma^{\text{int}}_{\text{int}}$, whose disjoint is denoted $\tilde{\Sigma}$. This way, the two matchings are encoded in the types of the symbols: the upper index is for the first and the lower index for the second matching relation. More formally, given $\omega \in \mathsf{WW}_2(\Sigma)$, we denote by $\tilde{\omega} \in \tilde{\Sigma}^*$ its visibly pushdown version.

In MCFG, non-terminals allow to express tuples of words. We present a grammar with two non-terminals W and H which represent respectively words (denoted $w \in \tilde{\Sigma}^*$), and pairs of words (denoted $(x, y) \in \tilde{\Sigma}^* \times \tilde{\Sigma}^*$). The grammar is defined by the following rules:

$$\begin{array}{lcl} \mathsf{W} \ni w & ::= & \epsilon \mid i \mid w_1 w_2 \mid x w y \\ \mathsf{H} \ni (x, y) & ::= & (\epsilon, \epsilon) \mid (x_1 x_2, y_2 y_1) \mid (w_1 x w_1', w_2 y w_2') \mid (axb, cyd) \end{array}$$

where $i \in \Sigma^{\text{int}}_{\text{int}}$ and $(a, b, c, d) \in \Sigma^c_c \times \Sigma^r_c \times \Sigma^c_r \times \Sigma^r_r$.

▶ **Lemma 9.** $L(\mathsf{W}) = \{\tilde{\omega} \in \tilde{\Sigma}^* \mid \omega \in \mathsf{WW}_2(\Sigma)\}$

Proof sketch. We give some hints on how to show the right to left implication. Let $\omega = (w, M_1, M_2) \in \mathsf{WW}_2(\Sigma)$. We show, by induction on $n \leq |w|$, the following properties:

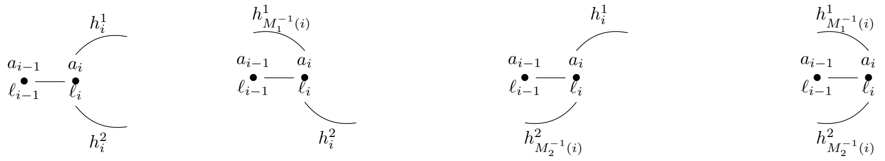
- Let $I \subseteq [|w|]$ such that $|I| = n$ and I is wpa, then $\tilde{\omega}_{|I} \in L(\mathsf{W})$.
- Let $I_1, I_2 \subseteq [|w|]$ such that $|I_1| + |I_2| = n$ and (I_1, I_2) is wpa, then $(\tilde{\omega}_{|I_1}, \tilde{\omega}_{|I_2}) \in L(\mathsf{H})$.

The proof decomposes the 2-wave, by distinguishing cases according to the structure of the two matchings. One can then verify that in all cases, one can produce the words using one of the rules of the grammar. ◀

3 2-Nested Word Automata

Nested word automata have been introduced in [3] as an extension of finite-state automata intended to recognize nested words. They label arches of the matching relation with so-called *hierarchical* states: if the position corresponds to a call (resp. a return), then the automaton "outputs" (resp. "receives") the hierarchical state used to label the arch, hence we place it after the input letter (resp. before). This corresponds to push/pop operations performed by a (visibly) pushdown automaton. The extension of this model to multiple matchings is natural, and has already been considered in [4, 6]: with two matchings, automata label edges of both matchings with hierarchical states.

We first introduce some notations. Let us assume that two matching relations M_1, M_2 of length n are given. Then, each index $i \in [n]$ can be labelled in 9 different ways depending on it's call, return and internal status with regard to matchings M_1 and M_2. We say that a position i is a *call-return* if it is a call w.r.t. M_1, and a return w.r.t. M_2. We extend this convention to other possible types of positions (*call-call*, *return-call*, *call-internal*. . .). Let $I \subseteq [n]$ and $x, y \in \{c, r, \text{int}\}$. Following our graphical representation of 2-nested words, in which M_1 is depicted above the word, and M_2 is depicted below, we denote by I^x_y the subset



$(\ell_{i-1}, a_i, h_i^1, h_i^2, \ell_i) \in \Delta_c^c$ $(\ell_{i-1}, h^1_{M_1^{-1}(i)}, a_i, h_i^2, \ell_i) \in \Delta_c^r$ $(\ell_{i-1}, h^2_{M_2^{-1}(i)}, a_i, h_i^1, \ell_i) \in \Delta_r^c$ $(\ell_{i-1}, h^1_{M_1^{-1}(i)}, h^2_{M_2^{-1}(i)}, a_i, \ell_i) \in \Delta_r^r$

Figure 3 Examples of transition steps.

of I with x status on M_1, and y status on M_2. For instance, given a position $i \in I$, we have $i \in I_r^c$ if i is a call w.r.t. M_1 and a return w.r.t. M_2. We also introduce the following shortcuts: for $x \in \{c, r\}$, $I^x = I_c^x \cup I_r^x \cup I_{\text{int}}^x$ and $I_x = I_x^c \cup I_x^r \cup I_x^{\text{int}}$. For instance, I^c (resp. I_c) denotes the set of positions in I which are a call w.r.t. M_1 (resp. M_2).

▶ **Definition 10.** *A* 2-nested word automaton *is a tuple $A = (Q, Q_0, Q_f, P, \Sigma, \Delta)$ where :*

- *Q is a finite set of states and $Q_0, Q_f \subseteq Q$ are respectively the initial and final states*
- *P is a set of hierarchical states*
- *$\Delta = (\Delta_y^x)_{x,y \in \{c,r,int\}}$ is a set of transitions : $\Delta_y^x \subseteq Q \times P^{\mathsf{in}_{x,y}} \times \Sigma \times P^{\mathsf{out}_{x,y}} \times Q$, where $\mathsf{in}_{x,y}$ (resp. $\mathsf{out}_{x,y}$) is the number of r (resp. c) in $\{x, y\}$. For instance, $\mathsf{in}_{c,r} = \mathsf{in}_{r,c} = 1$ and $\mathsf{out}_{c,c} = 2$.*

▶ Remark 11. In the previous definition, elements in $P^{\mathsf{in}_{x,y}}$ correspond to hierarchical states that label closing arches, which can be interpreted as popped symbols, while elements in $P^{\mathsf{out}_{x,y}}$ correspond to hierarchical states that label opening arches, which can be interpreted as pushed symbols. Elements of Q are called *linear* states, as they follow the edges of the linear order, in contrast to hierarchical states.

▶ **Definition 12** (Run/Language of a 2NWA). *Let $\omega = (a_1 \dots a_n, M_1, M_2) \in \mathsf{2NW}$ and A be a* 2NWA*. Let $\ell = (\ell_i)_{i \in [\![0,n]\!]}$ be a sequence of states, $h^1 = (h_i^1)_{i \in [n]^c}$ and $h^2 = (h_i^2)_{i \in [n]_c}$ be two sequences of elements of P. For all $i \in [n]$, we write $run_i^A(\omega, \ell, h^1, h^2)$ if one of the following cases holds: (the first four cases are illustrated on Figure 3)*

- **call-call:** *$i \in [n]_c^c$, and $(\ell_{i-1}, a_i, h_i^1, h_i^2, \ell_i) \in \Delta_c^c$*
- **return-call:** *$i \in [n]_c^r$, and $(\ell_{i-1}, h^1_{M_1^{-1}(i)}, a_i, h_i^2, \ell_i) \in \Delta_c^r$*
- **call-return:** *$i \in [n]_r^c$, and $(\ell_{i-1}, h^2_{M_2^{-1}(i)}, a_i, h_i^1, \ell_i) \in \Delta_r^c$*
- **return-return:** *$i \in [n]_r^r$, and $(\ell_{i-1}, h^1_{M_1^{-1}(i)}, h^2_{M_2^{-1}(i)}, a_i, \ell_i) \in \Delta_r^r$*
- **call-internal:** *$i \in [n]_{int}^c$, and $(\ell_{i-1}, a_i, h_i^1, \ell_i) \in \Delta_{int}^c$*
- **internal-call:** *$i \in [n]_c^{int}$, and $(\ell_{i-1}, a_i, h_i^2, \ell_i) \in \Delta_c^{int}$*
- **return-internal:** *$i \in [n]_{int}^r$, and $(\ell_{i-1}, h^1_{M_1^{-1}(i)}, a_i, \ell_i) \in \Delta_{int}^r$*
- **internal-return:** *$i \in [n]_r^{int}$, and $(\ell_{i-1}, h^2_{M_2^{-1}(i)}, a_i, \ell_i) \in \Delta_r^{int}$*
- **internal-internal:** *$i \in [n]_{int}^{int}$, and $(\ell_{i-1}, a_i, \ell_i) \in \Delta_{int}^{int}$*

If for all $i \in [n]$, $run_i^A(\omega, \ell, h^1, h^2)$ holds, the triple (ℓ, h^1, h^2) is said to be a run *of A* over *ω; it is an* accepting run *if $\ell_0 \in Q_0$ and $\ell_n \in Q_f$.*

We will write $q \xrightarrow[A]{\omega} q'$ when there exists a triple (ℓ, h^1, h^2) which is a run of A on ω, and whose first (resp. last) element of ℓ is equal to q (resp. q'). If we denote by n the length of ω, and if $I \subseteq [n]$ is an interval wpa, then we write $q \xrightarrow[A]{\omega, I} q'$ as a shortcut for $q \xrightarrow[A]{\omega_{|I}} q'$.

A 2-nested word is accepted by *A if it admits an accepting run. The set of all 2-nested words accepted by A is denoted $L(A)$, and a language of 2-nested words is called* regular *if it is accepted by a 2-nested word automaton. In the sequel, we will also be interested in restricting the language of a* 2NWA *to 2-wave words. Hence, we denote by $L_{\mathsf{WW}_2}(A)$ the set of 2-wave words accepted by A,* i.e. *$L(A) \cap \mathsf{WW}_2(\Sigma)$.*

▶ **Example 13.** Let $A_{ex} = (Q, Q_0, Q_f, P, \{a, b, c, d\}, \Delta)$, with $Q = \{q_a, q_b, q_c, q_d\}$, $Q_0 = \{q_a\}$, $Q_f = \{q_d\}$, $P = Q$ and Δ defined as follows (we only give non-empty transition sets):

$\Delta_c^c = \{(q_a, a, q_a, q_a, q_a)\}$
$\Delta_c^r = \{(q_x, q_a, b, q_b, q_b) \mid x \in \{a, b\}\}$
$\Delta_r^c = \{(q_x, q_b, c, q_c, q_c) \mid x \in \{b, c\}\}$
$\Delta_r^r = \{(q_x, q_c, q_a, d, q_d) \mid x \in \{c, d\}\}$

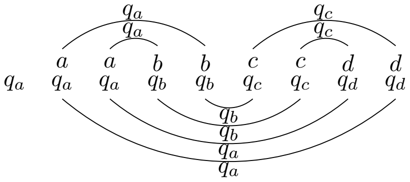

q_a
q_a
q_c
q_c
a a b b c c d d
q_a q_a q_a q_b q_b q_c q_c q_d q_d
q_b
q_b
q_a
q_a

Figure 4 A run of A_{ex} over ω_2.

We illustrate the semantics of 2NWA by giving on Figure 4 a graphical representation of a run of A_{ex} on the 2NW $\omega_2 = (a^2b^2c^2d^2, M_1, M_2)$, with M_1, M_2 depicted on Figure 4. We let the reader check that the projection of $L(A_{ex})$ on Σ^* is equal to $\{a^nb^nc^nd^n \mid n \geq 1\}$, hence not context-free.

▶ **Definition 14** (deterministic 2NWA). *A 2NWA is deterministic iff $Q_0 = \{q_0\}$ and for all $x, y \in \{c, r, int\}$, Δ_y^x induces a function $Q \times P^{\mathsf{in}_{x,y}} \times \Sigma \to P^{\mathsf{out}_{x,y}} \times Q$.*

▶ **Example 15.** The automaton A_{ex} considered in Example 13 is deterministic.

Normal form. To ease further constructions, we present a normal form for 2NWA that requires the hierarchical state of an arch to be equal to the target linear state of its call index. More formally, we say that a 2NWA is in weakly-hierarchical post form (post form for short) if $P = Q$ and for all $x, y \in \{c, r, \text{int}\}$, $\Delta_y^x \subseteq \bigcup_{q \in Q} Q \times Q^{\mathsf{in}_{x,y}} \times \Sigma \times \{q\}^{\mathsf{out}_{x,y}} \times \{q\}$. As a consequence, transitions of an automaton in post form can be simplified: $\Delta_y^x \subseteq Q \times Q^{\mathsf{in}_{x,y}} \times \Sigma \times Q$.

It is worth observing that in a 2NWA in post form, a run is completely characterized by the linear states. Hence, we can omit the hierarchical states in the formula run_i^A, and we can say that a sequence of (linear) states ℓ is a run of A on a 2-nested word ω.

▶ **Example 16.** The automaton A_{ex} considered in Example 13 is in post form.

▶ **Lemma 17.** *Given a* 2NWA $A = (Q, Q_0, Q_f, \Sigma, P, \Delta)$, *we can build a* 2NWA $A' = (Q', q_0', Q_f', \Sigma, Q', \Delta')$ *which is in weakly-hierarchical post form and such that $L(A) = L(A')$.*

Closure properties. Applying classical automata constructions to 2NWA, one can prove the following closure properties of regular 2NWL.

▶ **Proposition 18** (See also [4]). *Regular* 2NWL *are closed under union, intersection, and direct and reciprocal image by a non erasing alphabetic morphism.*

4 Determinization of 2NWA over 2-wave words

▶ **Theorem 19.** 2NWA *are determinizable on the subclass of 2-wave words,* i.e., *given a* 2NWA A, *we can build a deterministic* 2NWA A' *such that $L_{\mathsf{WW}_2}(A) = L_{\mathsf{WW}_2}(A')$.*

Thanks to Lemma 17, we start from a 2NWA in post normal form $A = (Q, Q_0, Q_f, \Delta)$. This normal form will allow to lighten the presentation, as less arguments are needed to write the transitions and the runs. We will describe the construction of a deterministic 2NWA A' (not in post normal form), which accepts the same language of 2-wave words.

Introduction to the construction. The determinization of finite-state automata, known as the powerset construction, registers all the states reachable by a run of A, which in the sequel are named *current* states. The determinization procedure of [3] for nested word automata requires the recording of two states. In addition to the current state, they also store the state labelling the call of the arch covering the current position. This state is named *reference* state. Intuitively, it stores where we come from, and will be used when closing the arch to reconcile the global run with what happened below this arch.

In our setting, as we consider a pair of matchings (M_1, M_2) instead of a single one, we need to store triples of states, composed of two reference states (one for M_1 and one for M_2), and one current state. Hence, states of A' will be sets of such triples of states.

The very particular shape of 2-wave words, and in particular the fact that the support arch and the top arches do end up at the same return-return position, ensure that we can gather the information collected along these two paths to compute, in a deterministic fashion, the set of possible current states.

However, when trying to address the setting of 2-wave words, we face another difficulty related to reference states. Indeed, the computations we will perform on each arch of the 2-wave word are somehow disconnected. In order to be sure that they can be reconciled, we will enrich the reference state of the second top arch of the 2-wave with the state of the bottom arch and that of the first top arch. This allows us to check whether the reference associated with the second top arch is *compatible* with that chosen for the first top arch.

Construction. We will define the deterministic 2NWA $A' = (Q', \{q'_0\}, Q'_f, P', \delta)$ in the following way. We first introduce the reference states for M_1 and M_2: [1]

$$\begin{array}{lll} \overline{\mathcal{R}}^1 & := Q & \text{the reference for positions at the surface of the first top arch} \\ \overline{\mathcal{R}}^2 & := Q^3 & \text{the reference for positions at the surface of the second top arch} \\ \underline{\mathcal{R}} & := Q & \text{the reference for } M_2 \end{array}$$

We denote by $\overline{\mathcal{R}}$ the union $\overline{\mathcal{R}}^1 \cup \overline{\mathcal{R}}^2$. This allows us to define:

$$\begin{array}{llll} Q' & := 2^{\overline{\mathcal{R}}^1 \times \underline{\mathcal{R}} \times Q} \cup 2^{\overline{\mathcal{R}}^2 \times \underline{\mathcal{R}} \times Q} & q'_0 & := \{(q_0, q_0, q_0);\ q_0 \in Q_0\} \\ Q'_f & := \{S \in Q';\ S \cap Q \times Q \times Q_f \neq \emptyset\} & P' & := (Q' \times \Sigma) \cup (Q' \times \Sigma)^2 \end{array}$$

The transition function δ for A' is defined as follows, by distinguishing cases according to the nature of the symbol. The construction is illustrated on Figure 5. In order to ease the writing, the arguments of the formula are not explicitly written.

- $\delta_{\text{int}}(S, \alpha) := \{(\overline{r}, \underline{r}, q');\ \exists q\ (\overline{r}, \underline{r}, q) \in S \wedge (q, \alpha, q') \in \Delta_{\text{int}}\}$
- $\delta_c^c(S_1, \alpha_1) := (S'_1, (S_1, \alpha_1), (S_1, \alpha_1))$ where

$$S'_1 := \{(q'_1, q'_1, q'_1);\ \exists \overline{r}_1, \underline{r}_0, q_1\ \phi_1\}$$
$$\phi_1 := (\overline{r}_1, \underline{r}_0, q_1) \in S_1 \wedge (q_1, \alpha_1, q'_1) \in \Delta_c^c.$$

[1] Observe that the reference state for the second top arch is a triple of states, as explained before.

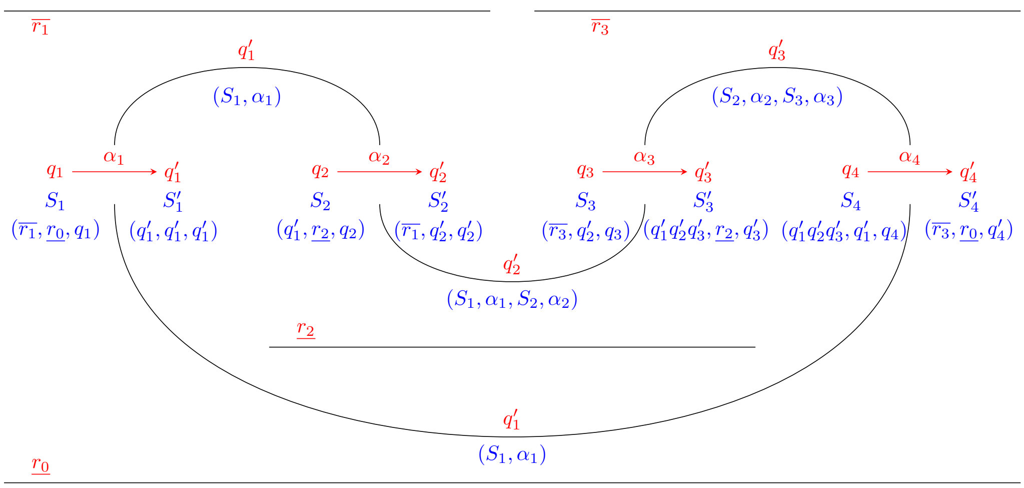

$\overline{r_1}$
q_1'
(S_1,α_1)
$\overline{r_3}$
q_3'
$(S_2,\alpha_2,S_3,\alpha_3)$
$q_1 \xrightarrow{\alpha_1} q_1'$
S_1
S_1'
$(\overline{r_1},\underline{r_0},q_1)$
(q_1',q_1',q_1')
$q_2 \xrightarrow{\alpha_2} q_2'$
S_2
S_2'
$(q_1',\underline{r_2},q_2)$
$(\overline{r_1},q_2',q_2')$
$q_3 \xrightarrow{\alpha_3} q_3'$
S_3
S_3'
$(\overline{r_3},q_2',q_3)$
$(q_1'q_2'q_3',\underline{r_2},q_3')$
$q_4 \xrightarrow{\alpha_4} q_4'$
S_4
S_4'
$(q_1'q_2'q_3',q_1',q_4)$
$(\overline{r_3},\underline{r_0},q_4')$
q_2'
$(S_1,\alpha_1,S_2,\alpha_2)$
$\underline{r_2}$
q_1'
(S_1,α_1)
$\underline{r_0}$

Figure 5 Illustration of the determinization. The (original) run in A is depicted in red, while the (new) one in A' is in blue. Elements of states of A' are illustrated by triples depicted below them.

- $\delta_c^r(S_2,(S_1,\alpha_1),\alpha_2) := (S_2',(S_1,\alpha_1,S_2,\alpha_2))$ where

$$S_2' := \{(\overline{r}_1,q_2',q_2');\ \exists \underline{r}_0,\underline{r}_2,q_1,q_1',q_2\ \phi_2\}$$
$$\phi_2 := \phi_1 \wedge (q_1',\underline{r}_2,q_2) \in S_2 \wedge (q_2,q_1',\alpha_2,q_2') \in \Delta_c^r$$

- $\delta_r^c(S_3,(S_1,\alpha_1,S_2,\alpha_2),\alpha_3) := (S_3',(S_2,\alpha_2,S_3,\alpha_3))$ where [2]

$$S_3' := \{(q_1'q_2'q_3',\underline{r}_2,q_3');\ \exists \overline{r}_1,\overline{r}_3,\underline{r}_0,q_1,q_2,q_3\ \phi_3\}$$
$$\phi_3 := \phi_2 \wedge \overline{r}_1 \preceq \overline{r}_3 \wedge (\overline{r}_3,q_2',q_3) \in S_3 \wedge (q_3,q_2',\alpha_3,q_3') \in \Delta_r^c$$

- $\delta_r^r(S_4,(S_2,\alpha_2,S_3,\alpha_3),(S_1,\alpha_1),\alpha_4) := S_4'$ where

$$S_4' := \{(\overline{r}_3,\underline{r}_0,q_4');\ \exists \overline{r}_1,\underline{r}_2,q_1,q_1',q_2,q_2',q_3,q_3',q_4\ \phi_4\}$$
$$\phi_4 := \phi_3 \wedge (q_1'q_2'q_3',q_1',q_4) \in S_4 \wedge (q_4,q_3',q_1',\alpha_4,q_4') \in \Delta_r^r$$

▶ Remark 20. Formulas $(\phi_j)_{j\in[4]}$ are not quantified at all, hence all their variables are free. Their objective is to link parameters extracted from the triplets of states of A'.

Proof of correctness. We fix a 2-wave word $\omega = (w, M_1, M_2)$, with $w = a_1 \dots a_n$. The following proposition states that on an interval without pending arch, runs of A' exactly capture possible runs of A, while keeping track of the reference states, as explained before.

▶ **Proposition 21.** *Let $[\![s,f]\!] \subseteq [n]$ be wpa, and $S,S' \in Q'$ such that $S \xrightarrow[A']{\omega,[\![s,f]\!]} S'$, then:*

$$\forall \overline{r} \in \overline{\mathcal{R}},\ \underline{r} \in \underline{\mathcal{R}},\ q' \in Q,\ \left((\overline{r},\underline{r},q') \in S' \iff \exists q \in Q\ (\overline{r},\underline{r},q) \in S \wedge q \xrightarrow[A]{\omega,[\![s,f]\!]} q'\right)$$

Before sketching the proof of this proposition, we fix some notations. Let $[\![s,f]\!] \subseteq [n]$ be an interval wpa, and $S,S' \in Q'$ such that $S \xrightarrow[A']{\omega,[\![s,f]\!]} S'$. Observe that as A' is deterministic,

[2] $\preceq$ denotes the prefix partial order on strings.

it has a unique run on the 2-wave word ω restricted to $[\![s,f]\!]$. We let $(L_i)_{i\in[\![s-1,f]\!]}$ denote the (linear) states of this run. In particular, we have $L_{s-1} = S$ and $L_f = S'$. As A' is deterministic, hierarchical states are completely determined from linear ones.

In order to prove Proposition 21, we separate the direct from the indirect case. For the indirect case, it is easy to show by induction that any run of A on ω (restricted to $[\![s,f]\!]$) will appear in the run of A' on ω (restricted to $[\![s,f]\!]$). This only requires to define carefully the corresponding reference states.

▶ **Lemma 22.** *Let $\ell = (\ell_i)_{i\in[\![s-1,f]\!]}$ be a run of A on ω (restricted to $[\![s,f]\!]$) such that $\ell_{s-1} = q$, $\ell_f = q'$ and $(\overline{r},\underline{r},q) \in S = L_{s-1}$. Then we can define reference mappings $\overline{R}$ and $\underline{R}$ from $[\![s-1,f]\!]$ to $\overline{\mathcal{R}}$ and $\underline{\mathcal{R}}$ respectively, such that $\overline{R}(f) = \overline{r}$, $\underline{R}(f) = \underline{r}$, and for all $i \in [\![s-1,f]\!]$, $(\overline{R}(i),\underline{R}(i),\ell_i) \in L_i$.*

The direct implication will be proven by induction on $[\![s,f]\!]$. More precisely, we rely on the grammar allowing to describe all 2-wave words given in Section 2. We use a slight abuse of notation here, as the grammar is based on the visibly pushdown presentation, while we work here with 2-nested words, but we believe the correspondence is without ambiguity. The three first cases of the grammar (internal, empty word, and concatenation of 2-wave words) are easy. The last case decomposes the 2-wave word ω as $\omega = x\omega' y$, with (x,y) being a pair of "matched" words, *i.e.* given by the non-terminal H. Intuitively, the grammar gives a decomposition of the wpa interval associated with ω into a wpa interval corresponding to ω', and a pair of matched intervals corresponding to (x,y). Hence, this requires to study pairs of matched intervals, *i.e.* a pair of intervals which is without pending arch. To this end, we introduce a notation for a "partial" run on a pair of matched intervals:

▶ **Definition 23.** *Let $(I_1 = [\![i_1,j_1]\!], I_2 = [\![i_2,j_2]\!])$ be a pair of matched intervals w.r.t. ω. Let q_1,q_2,q_1',q_2' be four states. We let $J = I_1 \cup I_2 \cup \{i_1-1, i_2-1\}$. We write $q_1,q_2 \xrightarrow[A]{\omega,I_1,I_2} q_1',q_2'$ if there exists $(\ell_i)_{i\in J}$ such that $\forall i \in J$, $run_i^A(\omega,\ell)$ and $\forall k \in [2]$, $q_k = \ell_{i_k-1}$ and $q_k' = \ell_{j_k}$*

We are now ready to state the following lemma:

▶ **Lemma 24.** *Let (i_1,i_2,i_3,i_4) be a 2-wave such that $i_1 \in [\![s,f]\!]$ and $(\overline{r}_3,\underline{r}_0,q_4') \in L_{i_4}$. For any $\overline{r}_1,\underline{r}_2,q_1,q_1',q_2,q_2',q_3,q_3',q_4$ satisfying ψ_4 we have $q_1,q_3 \xrightarrow[A]{\omega,[\![i_1,i_2]\!],[\![i_3,i_4]\!]} q_2',q_4'$.*

Lemma 24 confirms the intuition that a 2-wave is an encapsulation in its sort: any q_1, q_2' and q_3 yielded by $(\overline{r}_3,\underline{r}_0,q_4') \in L_{i_4}$ via the construction of L_{i_4} (e.g. via ψ_4) define with q_4' a run of A on ω on the subset of positions given by $[\![i_1,i_2]\!]$ and $[\![i_3,i_4]\!]$.

We explain how to conclude the proof of Proposition 21. Starting from the 2-wave word ω, we obtained a decomposition $\omega = x\omega' y$, with (x,y) being a pair of "matched" words produced by H. We can show in addition that extremal positions of x and y exactly correspond to a 2-wave, in the sense of the premises of Lemma 24. Combining it with the induction hypothesis (direct implication of Proposition 21) applied on ω', we can exhibit the expected run in A.

Proof of Theorem 19. First, observe that by construction, A' is deterministic and complete. Let $\omega \in \mathsf{WW}_2(\Sigma)$. As A' is deterministic and complete, it has a unique run on ω starting from q_0'. Let $(L_i)_{i\in[\![0,n]\!]}$ be this run, with $L_0 = q_0'$. We proceed by equivalence:

$$\begin{aligned}\omega \in L(A') \iff L_n \in Q_f' \iff L_n \cap Q \times Q \times Q_f \neq \emptyset \iff \exists(\overline{r},\underline{r},q_f) \in L_n\ q_f \in Q_f \\ \iff \exists q_0, q_f \in Q_f\ (\overline{r},\underline{r},q_0) \in q_0' \text{ and } q_0 \xrightarrow[A]{\omega} q_f \iff \omega \in L(A).\end{aligned}$$

◀

Closure under complementation. It is proved in [19] that regular 2NWL are not closed under complementation, since they are not determinizable. The definition of 2NWL they use is sightly different from ours because the two matchings do not share positions, but the same negative result can be shown for our definition. For example, there is no deterministic automaton recognizing the language $\{(a^{n+m}b^m a^n b^m a^n, \{(i, i+2(n+m)) \mid 1 \leq i \leq n+m\}, \{(i, i+n+m) \mid 1 \leq i \leq n+m\}) \mid n, m \geq 0\}$. However, thanks to our determinization result for regular WW_2 languages, we get:

▶ **Proposition 25.** *Regular languages of* 2*-wave words are closed under complementation.*

5 Logical characterization

We show that languages of 2-wave words definable in monadic second order logic (MSO) are exactly regular languages of 2-wave words. Let us fix $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{V}_1$ is the set of first order variables (whose elements will be written using lower-cases) and $\mathcal{V}_2$ is the set of second order variables (whose elements will be written using upper-cases).

The monadic second-order logic of 2-nested words (MSO$(\Sigma, <, M_1, M_2)$) is given by

$$\phi := Q_a(x) \mid x < y \mid X(x) \mid M_i(x, y) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x\ \phi \mid \exists X\ \phi$$

where $a \in \Sigma$, $x, y \in \mathcal{V}_1$, $X \in \mathcal{V}_2$, $i \in \{1, 2\}$.

The semantics is defined over 2-nested words in a natural way. The first-order variables are interpreted over positions of the nested word, while set variables are interpreted over sets of positions; $Q_a(x)$ holds if the symbol at the position interpreted for x is a, $x < y$ holds if the position interpreted for x is lesser than the position interpreted for y, and $M_i(x, y)$ holds if the positions interpreted for x and y are related by a nesting edge of matching M_i.

Let us define some fragments of MSO$(\Sigma, <, M_1, M_2)$. The set FO$(\Sigma, <, M_1, M_2)$ of first order (FO) formulas is the set of all formulas in MSO$(<, M_1, M_2)$ that do not contain any second-order quantifier. Furthermore, the set EMSO$(\Sigma, <, M_1, M_2)$ of existential MSO (EMSO) formulas consists of all formulas of the form $\exists X_1 \ldots \exists X_n \phi$, with $\phi \in$ FO$(\Sigma, <, M_1, M_2)$. If $\mathcal{L}$ is a logic (MSO, EMSO or FO), and ϕ is a closed formula in $\mathcal{L}(\Sigma, <, M_1, M_2)$, the *language defined by* ϕ, denoted $L(\phi)$ is the set of all 2-nested words that are a model for ϕ, and $L_{\mathsf{WW}_2}(\phi)$ is the set of all 2-wave words that are a model for ϕ, that is, $L_{\mathsf{WW}_2}(\phi) = L(\phi) \cap \mathsf{WW}_2(\Sigma)$.

In [4], Bollig shows the equivalence between automata and EMSO for the whole class of 2-nested words. However, he shows that quantifier alternation yields an infinite hierarchy. In our setting, this hierarchy collapses, and we obtain the equivalence between MSO and EMSO. The following characterization extends that given in [3] for regular nested word languages. Its proof follows classical lines, and relies on Propositions 18 and 25.

▶ **Theorem 26.** *A language of* 2*-wave words is definable in EMSO iff it is definable in MSO iff it is regular.*

6 Decision problems

The analysis we did so far of 2NWA over 2-wave words allows to establish the following decidability results:

▶ **Theorem 27.** *Let A and B be two* 2NWA *over* Σ*. The following holds:*

- *Determining whether* $L_{\mathsf{WW}_2}(A) = \emptyset$ *can be decided in polynomial time.*
- *Determining whether* $L_{\mathsf{WW}_2}(A) = \mathsf{WW}_2(\Sigma)$ *can be decided in exponential time.*
- *Determining whether* $L_{\mathsf{WW}_2}(A) \subseteq L_{\mathsf{WW}_2}(B)$ *(resp.* $L_{\mathsf{WW}_2}(A) = L_{\mathsf{WW}_2}(B)$*) can both be decided in exponential time.*

Proof sketch. The first result follows from the grammar presented in Section 2 that produces all 2-wave words. More precisely, we maintain a set W of pairs of states (for non-terminal W) and a set H of quadruplets of states (for non-terminal H). Intuitively, $(p, q) \in W$ means that there exists a 2-wave word ω and a run $p \xrightarrow[A]{\omega} q$. We initialize them with identity relations and saturate them using the rules of the grammar, and corresponding transitions of A. The other results are direct consequences from closure under complement using the determinization procedure, with the observation that the latter has exponential complexity. ◀

On the treewidth of 2-wave words. Any 2NW $\omega = (w, M_1, M_2)$ of length n can be seen as a graph whose set of vertices is $[n]$ and set of edges is $M_1 \cup M_2 \cup \{(i, i+1)\}_{i \in [n-1]}$, then an alternative approach to decidability is by using Courcelle's Theorem on graphs of bounded treewidth. Indeed, using the grammar presented in Section 2 for 2-wave words (Lemma 9), we show:

▶ **Lemma 28.** *For all $\omega \in \mathsf{WW}_2(\Sigma)$, ω has treewidth at most* 11.

In addition, it is easy to verify that the class of 2-wave words can be expressed in MSO. As a consequence, we obtain using [7, 17]:

▶ **Proposition 29.** *Given a formula $\phi \in MSO(<, M_1, M_2)$, checking whether there exists $\omega \in \mathsf{WW}_2(\Sigma)$ that satisfies ϕ is decidable.*

7 Relation to indexed languages

Let $\mathcal{L}$ be a logic (FO, EMO, or MSO), we denote by $\exists\text{Match}\mathcal{L}(\Sigma, <, M)$ the class of all word languages $L = \{u \in \Sigma^* \mid \exists M (u, M) \in L_M\}$ such that L_M is definable in $\mathcal{L}(\Sigma, <, M)$. Similarly, we denote by $\exists\mathsf{WW}_2\mathcal{L}(\Sigma, <, M_1, M_2)$ the class of languages $L = \{u \in \Sigma^* \mid \exists (u, M_1, M_2) \in L_{\mathsf{WW}_2}(\phi)\}$ for $\phi \in \mathcal{L}(\Sigma, <, M_1, M_2)$.

It is proved in [12] that CFL $= \exists\text{Match FO}(\Sigma, <, M) = \exists\text{Match MSO}(\Sigma, <, M)$. A key of the proof is the fact that CF grammars can be put in Greibach double normal form. Then each production has the form $X \to aub$, and the derivation tree of a word can be encoded by matching letters delimiting each production.

Consider now indexed grammars that generate Indexed Languages (IL). They are CF grammars where each non-terminal symbol carries a pushdown stack (often denoted X^ω where X is the non terminal and ω the stack). Grammars contain *push productions* of the form $X \to Y^p$ saying that any X^ω can be rewritten $Y^{p\omega}$; *pop productions* of the form $X^p \to Y$ saying that any $X^{p\omega}$ can be rewritten Y^ω and *copy productions* of the form $X \to YZ$ saying that any X^ω can be rewritten $Y^\omega Z^\omega$.

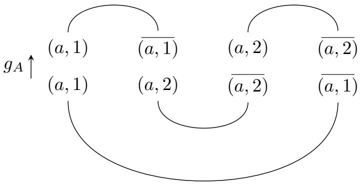
The derivation tree of a word can be encoded using two matching relations: one delimiting productions (corresponding to M_2), and one for the stack moves (corresponding to M_1). The nested structure thus obtained is a wave structure, where each wave follows a pushed symbol amongst the different copies. In particular, a 2-wave corresponds to a symbol which has been popped but has never been copied. Indexed grammars which never process copies are called *linear indexed grammars* and generate *linear indexed languages* (LIL). In such grammars, copy productions have the form $X \to X^\bullet Y$ or $X \to XY^\bullet$ where $\bullet$ marks the symbol on which the stack is transmitted.

The next Proposition establishes a formal connection between regular languages of 2-wave words and linear indexed languages:

▶ **Proposition 30.** *Languages in $\exists\mathsf{WW}_2 MSO(\Sigma, <, M_1, M_2)$ are linear indexed languages.*

Proof sketch. We use the homomorphic characterization of linear indexed languages given in [22]. We denote by $\mathcal{D}_A$ the Dyck language over alphabet A with inverse letters in $\bar{A} = \{\bar{a} \mid a \in A\}$. Then, a language L is linear indexed iff there exists an alphabet A, a regular language R and a morphism h such that $L = h(R \cap \mathcal{D}_{\Gamma_A} \cap g_A^{-1}(\mathcal{D}_{\Gamma_A}))$, where: $A_i = \{(a, i) \mid a \in A\}$, for $i = 1, 2$, $\Gamma_A = A_1 \cup A_2$ and g_A is the morphism defined by

$$g_A : \begin{array}{ll} (a,1) \mapsto (a,1) & \overline{(a,1)} \mapsto \overline{(a,2)} \\ (a,2) \mapsto \overline{(a,1)} & \overline{(a,2)} \mapsto (a,2) \end{array}$$



Remark that from a given a word u in $\mathcal{D}_{\Gamma_A} \cap g_A^{-1}(\mathcal{D}_{\Gamma_A})$, we can construct a 2NW (u, M_1, M_2) where arches in M_2 correspond to $\mathcal{D}_{\Gamma_A}$, and M_1 to $g_A^{-1}(\mathcal{D}_{\Gamma_A})$.

The proof is then similar to that of Chomsky-Schützenberger Theorem applied to pushdown automata: the regular language R encodes runs on linear states, $\mathcal{D}_{\Gamma_A}$ and $g_A^{-1}(\mathcal{D}_{\Gamma_A})$ ensure the consistency of hierarchical states and h projects the run on its word. ◀

However, we don't know if the equality holds, mainly because we don't know if linear indexed languages can be put in a Greibach double normal like form.

We can also ask whether the logical characterization of CFL extends to indexed languages. The appropriate structure seems to be 2NW whose edges form *waves* of unbounded length. Intuitively, a k-wave generalizes a 2-wave as follows: it consists of $2k$ indices in increasing order, with k top arches, $k-1$ bottom arches, and an additional support arch. An example of a 4-wave is depicted on Figure 1. This yields the notion of k-wave word (WW_k), and that of wave word (WW), which is simply the union of WW_k over k. As an example, the 2-nested word depicted on Figure 6 is in WW_4. So, the question is: does $\mathsf{IL} = \exists\mathsf{WWFO}(\Sigma, <, M_1, M_2) = \exists\mathsf{WWMSO}(\Sigma, <, M_1, M_2)$ hold? Proposition 31 provides an element of response, and we think that $\mathsf{IL} \neq \exists\mathsf{WWFO}(\Sigma, <, M_1, M_2)$.

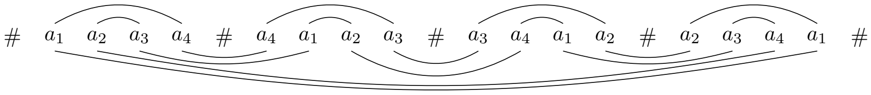


Figure 6 Example of a word in L and its associated wave structure.

▶ **Proposition 31.** *There exists a language which is not an indexed language but being* $\exists\mathsf{WW}$*EMSO*$(\Sigma, <, M_1, M_2)$*-definable.*

Proof sketch. Consider $\Sigma = A \cup \{\#\}$ for any alphabet A, and the set L of all words of the form $\#u_1\#u_2\# \dots u_n\#$, for $n \geq 1$, such that for all $i \in [1, n-1]$, $u_i \in A^n$, and if $u_i = a_1 \dots a_n$, then $u_{i+1} = a_n a_1 \dots a_{n-1}$. Using the Shrinking Lemma given in [11] for indexed languages, it can easily be proved that L is not an indexed language. In addition, one can write an EMSO-formula defining 2-wave words $(\#u_1\#u_2\# \dots u_n\#, M_1, M_2)$ such that (M_1, M_2) forms $n/2$ embedded n-waves (see an example on Figure 6). ◀

8 Conclusion

A natural perspective of this work consists in studying its extension to k-wave words. We believe that the determinization property should hold for this class too. This will require to improve our arguments, as the present proof seems intricate to be adapted to k-waves.

For unbounded waves, automata are not determinizable: Bollig proved in [4] that regular 2NWL are not closed under complementation, and are then not determinizable. His proof uses the encoding of grids in 2NWL, and can thus be adapted to (unbounded) wave words.

Another perspective consists in characterizing the subclass of indexed languages capturing languages $\exists\mathsf{WW}L$, when L is a regular WW_2 language, and determine if $\exists\mathsf{WW}L$ is still indexed when L is a regular WW_k language, for $k > 2$.

References

1 Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.

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 Benedikt Bollig. On the expressive power of 2-stack visibly pushdown automata. *Log. Methods Comput. Sci.*, 4(4), 2008. doi:10.2168/LMCS-4(4:16)2008.

5 Dario Carotenuto, Aniello Murano, and Adriano Peron. 2-visibly pushdown automata. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Developments in Language Theory, 11th International Conference, DLT 2007, Turku, Finland, July 3-6, 2007, Proceedings*, volume 4588 of *Lecture Notes in Computer Science*, pages 132–144. Springer, 2007. doi:10.1007/978-3-540-73208-2_15.

6 Dario Carotenuto, Aniello Murano, and Adriano Peron. Ordered multi-stack visibly pushdown automata. *Theor. Comput. Sci.*, 656:1–26, 2016. doi:10.1016/j.tcs.2016.08.012.

7 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 313–400. World Scientific, 1997.

8 W. Damm. The IO- and OI-hierarchies. *Theoret. Comput. Sci.*, 20(2):95–207, 1982.

9 J. Engelfriet. Iterated pushdown automata and complexity classes. In *Proceedings of the 14th Symposium on Theory of Computing*, pages 365–373. Association for Computing Machinery, 1983.

10 Séverine Fratani and El Makki Voundy. Epsilon-reducible context-free languages and characterizations of indexed languages. *Inf. Comput.*, 269, 2019. doi:10.1016/j.ic.2019.104444.

11 Robert H. Gilman. A shrinking lemma for indexed languages. *Theor. Comput. Sci.*, 163(1&2):277–281, 1996. doi:10.1016/0304-3975(96)00244-7.

12 Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for context-free languages. 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 205–216. Springer, 1994. doi:10.1007/BFb0022257.

13 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 283–294. ACM, 2011. doi:10.1145/1926385.1926419.

14 A. N. Maslov. The hierarchy of index languages of arbitrary level. *Dokl. Akad. Nauk SSSR*, 217:1013–1016, 1974.

15 A. N. Maslov. Multilevel pushdown automata. *Problemy Peredači Informacii*, 12(1):55–62, 1976.

16 A. Okhotin. Non-erasing variants of the chomsky-schützenberger theorem. In *Developments in Language Theory*, volume 7410 of *Lecture Notes in Comput. Sci.*, pages 121–129. Springer, 2012.

17 Detlef Seese. The structure of models of decidable monadic theories of graphs. *Ann. Pure Appl. Log.*, 53(2):169–195, 1991. doi:10.1016/0168-0072(91)90054-P.

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

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

20 Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. The language theory of bounded context-switching. In Alejandro López-Ortiz, editor, *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010. Proceedings*, volume 6034 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2010. doi:10.1007/978-3-642-12200-2_10.

21 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Scope-bounded pushdown languages. *Int. J. Found. Comput. Sci.*, 27(2):215–234, 2016. doi:10.1142/S0129054116400074.

22 D. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD thesis, University of Pennsylvania, 1988. Available as Technical Report MS-CIS-88-74.

Metric Dimension Parameterized by Feedback Vertex Set and Other Structural Parameters

Esther Galby ✉
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Liana Khazaliya ✉
Saint Petersburg State University, Saint Petersburg, Russia

Fionn Mc Inerney ✉
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Roohani Sharma ✉
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Prafullkumar Tale ✉
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

For a graph G, a subset $S \subseteq V(G)$ is called a *resolving set* if for any two vertices $u, v \in V(G)$, there exists a vertex $w \in S$ such that $d(w,u) \neq d(w,v)$. The Metric Dimension problem takes as input a graph G and a positive integer k, and asks whether there exists a resolving set of size at most k. This problem was introduced in the 1970s and is known to be NP-hard [GT 61 in Garey and Johnson's book]. In the realm of parameterized complexity, Hartung and Nichterlein [CCC 2013] proved that the problem is W[2]-hard when parameterized by the natural parameter k. They also observed that it is FPT when parameterized by the vertex cover number and asked about its complexity under *smaller* parameters, in particular the feedback vertex set number. We answer this question by proving that Metric Dimension is W[1]-hard when parameterized by the feedback vertex set number. This also improves the result of Bonnet and Purohit [IPEC 2019] which states that the problem is W[1]-hard parameterized by the treewidth. Regarding the parameterization by the vertex cover number, we prove that Metric Dimension does not admit a polynomial kernel under this parameterization unless NP $\subseteq$ coNP/*poly*. We observe that a similar result holds when the parameter is the distance to clique. On the positive side, we show that Metric Dimension is FPT when parameterized by either the distance to cluster or the distance to co-cluster, both of which are smaller parameters than the vertex cover number.

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

Keywords and phrases Metric Dimension, Parameterized Complexity, Feedback Vertex Set

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.51

Related Version *Full Version*: https://arxiv.org/abs/2206.15424

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

Acknowledgements The authors would like to thank Florent Foucaud for pointing us to Gutin et al. [21]. The article contains a result that subsumes our result conditionally refuting the polynomial kernel for Metric Dimension parameterized by the vertex cover number.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 51; pp. 51:1–51:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Problems dealing with distinguishing the vertices of a graph have attracted a lot of attention over the years, with the metric dimension problem being a classic one that has been vastly studied since its introduction in the 1970s by Slater [31], and independently by Harary and Melter [22]. Formally, given a graph G and an integer $k \geq 1$, the Metric Dimension problem asks whether there exists a subset $S \subseteq V(G)$ of vertices of G of size at most k such that, for any two vertices $u, v \in V(G)$, there exists a vertex $w \in S$ such that $d(w, u) \neq d(w, v)$. If such a subset $S \subseteq V(G)$ exists, it is called a *resolving set*. The size of a smallest resolving set of a graph G is the *metric dimension* of G, and is denoted by $MD(G)$.

There are many variants and problems associated to the metric dimension, with *identifying codes* [27], *adaptive identifying codes* [4], and *locating dominating sets* [32] asking for the vertices to be distinguished by their neighborhoods in the subset chosen. Other variants of note are the *k-metric dimension*, where each pair of vertices must be resolved by k vertices in $S \subseteq V(G)$ instead of just one [15], and the *truncated metric dimension*, where the distance metric is the minimum of the distance in the graph and some integer k [34]. Along similar lines, in the *centroidal dimension* problem, each vertex must be distinguished by its relative distances to the vertices in $S \subseteq V(G)$ [17]. The metric dimension has also been considered in digraphs, with Bensmail et al. [6] providing a summary of the related work in this area. Interestingly, there are many game-theoretic variants of the metric dimension, such as *sequential metric dimension* [5], the *localization game* [9, 24], and the *centroidal localization game* [8]. The metric dimension and its variants have been studied for both their theoretical interest and their numerous applications such as in network verification [2], fault-detection in networks [36], pattern recognition and image processing [30], graph isomorphism testing [1], chemistry [11, 26], and genomics [35]. For more on these variants and others, see [28] for the latest survey.

Much of the related work around the metric dimension problem focuses on its computational complexity. Metric Dimension was first shown to be NP-complete in general graphs in [19]. Later, it was also shown to be NP-complete in split graphs, bipartite graphs, co-bipartite graphs, and line graphs of bipartite graphs in [14], in bounded-degree planar graphs [12], and interval and permutation graphs of diameter 2 [18]. On the positive side, there are linear-time algorithms for Metric Dimension in trees [31], cographs [14], and cactus block graphs [25], and a polynomial-time algorithm for outerplanar graphs [12].

Since the problem is NP-hard even for very restricted cases, it is natural to ask for ways to confront this hardness. In this direction, the parameterized complexity paradigm allows for a more refined analysis of the problem's complexity. In this setting, we associate each instance I with a parameter ℓ, and are interested in an algorithm with running time $f(\ell) \cdot |I|^{\mathcal{O}(1)}$ for some computable function f. Parameterized problems that admit such an algorithm are called fixed parameter tractable (FPT) with respect to the parameter under consideration. On the other hand, under standard complexity assumptions, parameterized problems that are hard for the complexity class W[1] or W[2] do not admit such fixed-parameter algorithms. A parameter may originate from the formulation of the problem itself (called natural parameters) or it can be a property of the input graph (called structural parameters).

Hartung and Nichterlein [23] proved that Metric Dimension is W[2]-hard when parameterized by the natural parameter, the solution size k, even when the input graph is bipartite and has maximum degree 3. This motivated the study of the parameterized complexity of the problem under structural parameterizations. It was observed in [23] that the problem admits a simple FPT algorithm when parameterized by the vertex cover number. It took a considerable amount of work and/or meta-results to prove that there are FPT algorithms parameterized by the max leaf number [13], the modular width or treelength plus the maximum degree [3], and the treedepth [20]. In [14], they gave an XP algorithm parameterized by the feedback edge set number. Only recently, it was shown that Metric Dimension is W[1]-hard parameterized by the treewidth [7], answering an open question mentioned in [3, 12, 13]. This result was improved upon since, with it being shown that Metric Dimension is even NP-hard in graphs of treewidth 24 [29]. For more on the metric dimension, see [33] for a recent survey.

Our contributions. In this paper, we continue the analysis of structural parameterizations of Metric Dimension. See the Hasse diagram in Figure 1 for a summary of known results and our contributions. As mentioned before, it is known that Metric Dimension is W[1]-hard parameterized by the treewidth [7]. There are two natural directions to improve this result. One direction was to show that Metric Dimension is para-NP-hard parameterized by the treewidth, which was proven in [29]. Another direction is to prove that Metric Dimension is W[1]-hard for a higher parameter than treewidth, *i.e.*, one for which the treewidth is upper bounded by a function of it. A parameter fitting this profile is the feedback vertex set number since the treewidth of a graph G is upper bounded by the feedback vertex set number of G plus one. Moreover, the complexity of Metric Dimension parameterized by the feedback vertex set number is left as an open problem in [23], the seminal paper on the parameterized complexity of Metric Dimension. We take this direction and answer this open question of [23] by proving that Metric Dimension is W[1]-hard parameterized by the feedback vertex set number (see Sec. 2). We then revisit the complexity of the problem when parameterized by the vertex cover number. Recall that the problem is known to admit an FPT algorithm, and hence, a *kernel*, under this parameterization. We prove that, however, Metric Dimension does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP}/poly$ when parameterized by the vertex cover number (see Sec. 3)[1]. On the positive side, we then show that Metric Dimension is FPT for the structural parameters the distance to cluster and the distance to co-cluster both of which are smaller parameters than the vertex cover number (see Sec. 4). Note that the FPT algorithm for the distance to cluster parameter implies an FPT algorithm for the distance to clique parameter. With a slight modification of the reduction in Sec. 3, we establish the problem does not admit a polynomial kernel, under the same assumption, when the parameter is the distance to clique.

In this extended abstract, we omit the standard terminology and some formal proofs (which are marked with $\star$) due to space constraints and present them in the full version on arXiv. Recall that any two vertices $u, v \in V(G)$ are *true twins* if $N[u] = N[v]$, and are *false twins* if $N(u) = N(v)$. A subset of vertices $S \subseteq V(G)$ *resolves* a pair of vertices $u, v \in V(G)$ if there exists a vertex $w \in S$ such that $d(w, u) \neq d(w, v)$. A vertex $u \in V(G)$ is *distinguished* by a subset of vertices $S \subseteq V(G)$ if, for any $v \in V(G) \setminus \{u\}$, there exists a vertex $w \in S$ such that $d(w, u) \neq d(w, v)$. We end this section with the following simple observation.

▶ **Observation 1.** *Let G be a graph. Then, for any (true or false) twins $u, v \in V(G)$ and any resolving set S of G, $S \cap \{u, v\} \neq \emptyset$.*

[1] After this paper was short-listed for the proceedings of MFCS 2022, Florent Foucaud informed us of the paper of Gutin et al. [21], which contains a slightly stronger result.

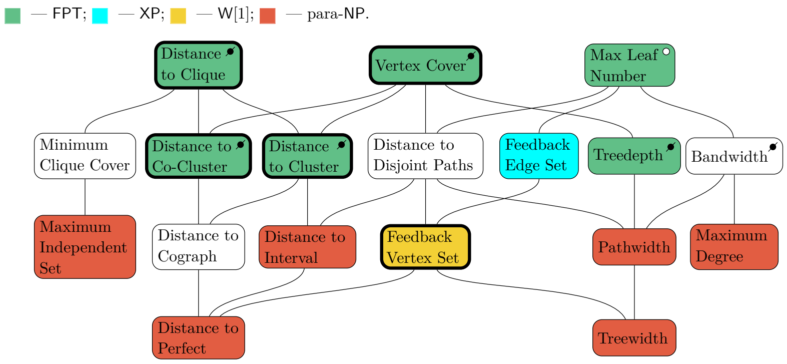


Figure 1 Hasse diagram of graph parameters and associated results for Metric Dimension. An edge indicates that the lower parameter is upper bounded by a function of the higher one. Colors correspond to the known hardness with respect to the highlighted parameter. The parameters for which the hardness remains an open question are not colored. The crossed bold circle in the upper-right corner means that Metric Dimension does not admit a polynomial kernel when parameterized by the marked parameter unless NP ⊆ coNP/*poly*; the white one if a polynomial kernel exists. The bold borders highlight parameters that are covered in this paper. Also see Footnote 1.

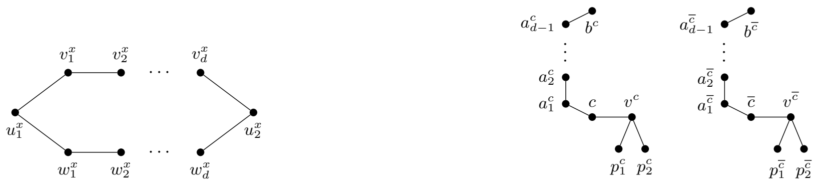
2 The Feedback Vertex Set Number

In this section, we prove that Metric Dimension is W[1]-hard parameterized by the feedback vertex set number. To prove this, we reduce from the NAE-Integer-3-Sat problem defined as follows. An instance of this problem consists of a set X of variables, a set C of clauses, and an integer d. Each variable takes a value in $\{1, \dots, d\}$, and clauses are of the form $(x \le a_x, y \le a_y, z \le a_z)$, where $a_x, a_y, a_z \in \{1, \dots, d\}$. A clause is satisfied if not all three inequalities are true and not all are false. The goal is to find an assignment of the variables that satisfies all given clauses. This problem was shown to be W[1]-hard parameterized by the number of variables [10].

▶ **Theorem 2.** Metric Dimension *is* W[1]*-hard parameterized by the feedback vertex set number.*

Proof. We reduce from NAE-Integer-3-Sat: given an instance (X, C, d) of this problem, we construct an instance (G, k) of Metric Dimension as follows. For each variable $x \in X$, we introduce a cycle G_x of length $2d + 2$ which has two distinguished *anchor vertices* u^x_1 and u^x_2 as depicted in Figure 2a; for convenience, we may also refer to u^x_1 as v^x_0 or w^x_0, and to u^x_2 as v^x_{d+1} or w^x_{d+1}. For each clause $c = (x \le a_x, y \le a_y, z \le a_z)$, we introduce the gadget G_c depicted in Figure 2b consisting of two vertex-disjoint copies H_c and $H_{\bar{c}}$ of the same graph. More precisely, for $\ell \in \{c, \bar{c}\}$, H_ℓ consists of a $K_{1,3}$ on the vertex set $\{\ell, v^\ell, p^\ell_1, p^\ell_2\}$, where v^ℓ has degree three, and a path P_{b^ℓ} of length d connects ℓ to b^ℓ. The subgraph of G_c induced by $\{\ell, v^\ell, p^\ell_1, p^\ell_2 \mid \ell \in \{c, \bar{c}\}\}$ is referred to as the *core of* G_c.

We further connect G_c to G_x, G_y, and G_z as follows. For every $t \in \{x, y, z\}$, we connect b^c to u^t_1 by a path $P^{t,c}_1$ of length $4d - a_t$, and v^c to u^t_2 by a path $P^{t,c}_2$ of length $4d + a_t - 1$. Furthermore, letting $w^{t,c}$ be the neighbor of v^c on $P^{t,c}_2$, we attach a copy $W^{t,c}$ of $K_{1,3}$ to $w^{t,c}$ by identifying $w^{t,c}$ with one of the leaves; we denote by $t^{t,c}_1$ and $t^{t,c}_2$ the two remaining leaves and refer to $W^{t,c}$ as a *pendant claw*. Similarly, for every $t \in \{x, y, z\}$, we connect $b^{\bar{c}}$ to u^t_2 by a path $P^{t,\bar{c}}_2$ of length $3d + a_t$, and $v^{\bar{c}}$ to u^t_1 by a path $P^{t,\bar{c}}_1$ of length $5d - a_t$. Furthermore, letting $w^{t,\bar{c}}$ be the neighbor of $v^{\bar{c}}$ on $P^{t,\bar{c}}_1$, we attach a copy $W^{t,\bar{c}}$ of $K_{1,3}$ to $w^{t,\bar{c}}$ by identifying $w^{t,\bar{c}}$ with one of the leaves; we denote by $t^{t,\bar{c}}_1$ and $t^{t,\bar{c}}_2$ the two remaining



(a) The variable gadget G_x.

(b) The clause gadget G_c is the disjoint union of H_c (left) and $H_{\bar{c}}$ (right).

Figure 2 The gadgets in the proof of Theorem 2.

leaves and refer to $W^{t,\bar{c}}$ as a *pendant claw*. Finally, we introduce a path $P = t_1pt_2$ which we connect to the clause gadgets as follows. For every clause $c \in C$ and $\ell \in \{c, \bar{c}\}$, we connect p to v^ℓ by a path P_ℓ of length $2d$. Furthermore, letting w^ℓ be the neighbor of p on P_ℓ, we attach a copy W^ℓ of $K_{1,3}$ to w^ℓ by identifying w^ℓ with one of the leaves; we denote by t_1^ℓ and t_2^ℓ the two remaining leaves and refer to W^ℓ as a *pendant claw*. This concludes the construction of G (see Figure 3).

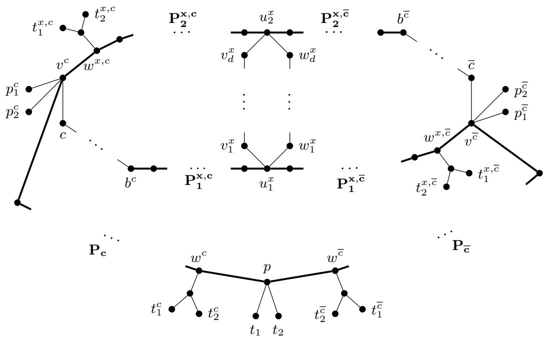


Figure 3 An illustration of the reduction in the proof of Theorem 2.

We set $k = |X| + 10|C| + 1$. Observe that the feedback vertex set number of G is at most $2|X| + 1$: indeed, removing $\{p\} \cup \{u_1^x, u_2^x \mid x \in X\}$ from G results in a graph without cycles. We next show that the instance (X, C, d) is satisfiable if and only if (G, k) is a Yes-instance for Metric Dimension. To this end, we first prove the following.

▷ Claim 3 (⋆). For any two distinct $s, t \in \{c, \bar{c} \mid c \in C\}$ and any two distinct variables $x, y \in X$, the following hold.

(i) The shortest path from H_s to H_t contains P_s and P_t as subpaths and has length $4d$.

(ii) $d(V(G_x), V(G_y)) \geq 6d$.

(iii) If x appears in the clause corresponding to s, then $d(V(G_x), V(H_s)) \geq 3d$.

(iv) If x does not appear in the clause corresponding to s, then any shortest path from G_x to H_s contains P_s as a subpath and has length at least $8d$.

▷ Claim 4 (⋆). For every clause $c = (x \leq a_x, y \leq a_y, z \leq a_z)$ and every $t \in \{x, y, z\}$, the following hold.

(i) For every $i \in \{0, \dots, d+1\}$, if $i \leq a_t$, then the shortest path from v_i^t to c contains $P_1^{t,c}$ as a subpath and has length $5d + i - a_t$. Otherwise, the shortest path from v_i^t to c contains $P_2^{t,c}$ as a subpath and has length $5d + 1 + a_t - i$.

(ii) For every $i \in \{0, \dots, d+1\}$, if $i \leq a_t - 1$, then the shortest path from v_i^t to v^c contains $P_1^{t,c}$ as a subpath and has length $5d + 1 + i - a_t$. Otherwise, the shortest path from v_i^t to v^c contains $P_2^{t,c}$ as a subpath and has length $5d + a_t - i$.

(iii) For every $i \in \{0, \ldots, d+1\}$, if $i \leq a_t - 2$, then the shortest path from v_i^t to $t_1^{t,c}$ contains $P_1^{t,c}$ as a subpath and has length $5d + 4 + i - a_t$. Otherwise, the shortest path from v_i^t to $t_1^{t,c}$ contains $P_2^{t,c}[u_2^t, w^{t,c}]$ as a subpath and has length $5d + 1 + a_t - i$.

▷ Claim 5 (⋆). For every clause $c = (x \leq a_x, y \leq a_y, z \leq a_z)$ and every $t \in \{x, y, z\}$, the following hold.

(i) For every $i \in \{0, \ldots, d+1\}$, if $i \leq a_t$, then the shortest path from v_i^t to $\overline{c}$ contains $P_1^{t,\overline{c}}$ as a subpath and has length $5d + 1 + i - a_t$. Otherwise, the shortest path from v_i^t to $\overline{c}$ contains $P_2^{t,\overline{c}}$ as a subpath and has length $5d + 1 + a_t - i$.

(ii) For every $i \in \{0, \ldots, d+1\}$, if $i \leq a_t + 1$, then the shortest path from v_i^t to $v^{\overline{c}}$ contains $P_1^{t,\overline{c}}$ as a subpath and has length $5d + i - a_t$. Otherwise, the shortest path from v_i^t to $v^{\overline{c}}$ contains $P_2^{t,\overline{c}}$ as a subpath and has length $5d + 2 + a_t - i$.

(iii) For every $i \in \{0, \ldots, d+1\}$, if $i \leq a_t + 2$, then the shortest path from v_i^t to $t_1^{t,\overline{c}}$ contains $P_1^{t,\overline{c}}[u_1^t, w^{t,\overline{c}}]$ as a subpath and has length $5d + 1 + i - a_t$. Otherwise, the shortest path from v_i^t to $t_1^{t,\overline{c}}$ contains $P_2^{t,\overline{c}}$ as a subpath and has length $5d + 5 + a_t - i$.

Assume first that (X, C, d) is satisfiable and let $\phi : X \to \{1, \ldots, d\}$ be an assignment of the variables satisfying every clause in C. We construct a resolving set S of G as follows. First, we add t_1 to S. For every variable $x \in X$, we add $v_{\phi(x)}^x$ to S. Finally, for every clause $c \in C$, we add $p_1^c, p_1^{\overline{c}}, t_1^c, t_1^{\overline{c}}$ to S and further add, for every variable t appearing in c, $t_1^{t,c}, t_1^{t,\overline{c}}$ to S. Note that $|S| = k$ and that every vertex of S is distinguished by itself. Let us show that S is indeed a resolving set of G. To this end, consider two distinct vertices $u, v \in V(G)$. We distinguish the following cases to show that there exists $w \in S$ such that $d(w, u) \neq d(w, v)$.

Case 1. *At least one of u and v belongs to a pendant claw.* W.l.o.g., assume first that $u \in V(W^\ell)$, where $\ell \in \{c, \overline{c} \mid c \in C\}$. If $v \in V(G) \setminus V(W^\ell)$, then $d(t_1^\ell, v) > 2 \geq d(t_1^\ell, u)$. Suppose therefore that $v \in V(W^\ell)$ as well. If $\{u, v\} \neq \{w^\ell, t_2^\ell\}$, then $d(t_1^\ell, u) \neq d(t_1^\ell, v)$. If $\{u, v\} = \{w^\ell, t_2^\ell\}$, then $d(t_1, w^\ell) = 2 < 4 = d(t_1, t_2^\ell)$. Second, assume that $u \in V(W^{t,\ell})$, where $\ell \in \{c, \overline{c}\}$ for some clause $c \in C$ and t is a variable appearing in clause c. If $v \in V(G) \setminus V(W^{t,\ell})$, then $d(t_1^{t,\ell}, v) > 2 \geq d(t_1^{t,\ell}, u)$. Suppose therefore that $v \in V(W^{t,\ell})$ as well. If $\{u, v\} \neq \{w^{t,\ell}, t_2^{t,\ell}\}$, then $d(t_1^{t,\ell}, u) \neq d(t_1^{t,\ell}, v)$. If $\{u, v\} = \{w^{t,\ell}, t_2^{t,\ell}\}$, then $d(p_1^\ell, w^{t,\ell}) = 2 < 4 = d(p_1^\ell, t_2^{t,\ell})$.

Case 2. *At least one of u and v belongs to the core of a clause gadget.* Assume, w.l.o.g., that $u \in \{\ell, v^\ell, p_1^\ell, p_2^\ell\}$, where $\ell \in \{c, \overline{c}\}$ for some clause $c = (x \leq a_x, y \leq a_y, z \leq a_z)$. If v is not a neighbor of v^ℓ, then $d(p_1^\ell, v) > 2 \geq d(p_1^\ell, u)$. If v is the neighbor of v^ℓ on the path P_ℓ, then $d(t_1, v) = d < d(t_1, u)$. Also, $v = w^{t,\ell}$ was covered by the previous case. So, consider $v \in \{\ell, v^\ell, p_1^\ell, p_2^\ell\}$. If $\{u, v\} \neq \{\ell, p_2^\ell\}$, then clearly $d(p_1^\ell, u) \neq d(p_1^\ell, v)$. Assume therefore that $\{u, v\} = \{\ell, p_2^\ell\}$. Since ϕ satisfies c, there exist $t, f \in \{x, y, z\}$ such that $\phi(t) \leq a_t$ and $\phi(f) > a_f$. Then, either $\phi(t) < a_t$, in which case, by Claim 4(i) and (ii),

$$d(v_{\phi(t)}^t, c) = 5d + \phi(t) - a_t < 5d + \phi(t) - a_t + 2 = d(v_{\phi(t)}^t, v^c) + 1 = d(v_{\phi(t)}^t, p_2^c),$$

or $\phi(t) = a_t$, in which case, by Claim 4(i) and (ii),

$$d(v_{\phi(t)}^t, c) = 5d < 5d + 1 = d(v_{\phi(t)}^t, v^c) + 1 = d(v_{\phi(t)}^t, p_2^c).$$

Similarly, either $\phi(f) = a_f + 1$, in which case, by Claim 5(i) and (ii),

$$d(v_{\phi(f)}^f, \overline{c}) = 5d < 5d + 2 = d(v_{\phi(f)}^f, v^{\overline{c}}) + 1 = d(v_{\phi(f)}^f, p_2^{\overline{c}}),$$

or $\phi(f) > a_f + 1$, in which case, by Claim 5(i) and (ii),

$$d(v_{\phi(f)}^f, \overline{c}) = 5d + 1 + a_f - \phi(f) < 5d + 3 + a_f - \phi(f) = d(v_{\phi(f)}^f, v^{\overline{c}}) + 1 = d(v_{\phi(f)}^f, p_2^{\overline{c}}).$$

In all cases, we conclude that there exists $w \in S$ such that $d(w, \ell) \neq d(w, p_2^\ell)$.

Case 3. *At least one of u and v belongs to a variable gadget.* Assume, w.l.o.g., that $u \in V(G_x)$ for some variable $x \in X$. By the previous cases, we may assume that v does not belong to the core of a clause gadget or a pendant claw. If $v \in V(G_y)$ for some variable $y \neq x$, then by Claim 3(ii),

$$d(v^x_{\phi(x)}, u) \leq d+1 < 6d \leq d(V(G_x), V(G_y)) \leq d(v^x_{\phi(x)}, v).$$

Now, suppose that $v \in V(G_x)$ as well. If $\{u, v\} = \{v^x_i, w^x_i\}$ for some $i \in [d]$, then

$$d(v^x_{\phi(x)}, v^x_i) = |\phi(x) - i| < d(v^x_{\phi(x)}, w^x_i) = \min\{\phi(x) + i, 2d + 2 - \phi(x) - i\}.$$

Suppose next that $u = v^x_i$ and $v = v^x_j$ for two distinct $i, j \in \{0, \ldots, d+1\}$, say $i < j$, w.l.o.g. Consider a clause $c = (x \leq a_x, y \leq a_y, z \leq a_z)$ containing x. If $j < a_x$, then by Claim 4(ii),

$$d(p^c_1, v^x_i) = d(v^c, v^x_i) + 1 = 5d + 2 + i - a_x < 5d + 2 + j - a_x = d(v^c, v^x_j) + 1 = d(p^c_1, v^x_j).$$

Now, suppose that $i < a_x \leq j$. Then, by Claim 4(ii),

$$d(p^c_1, v^x_i) - d(p^c_1, v^x_j) = 5d + 2 + i - a_x - (5d + a_x - j + 1) = i + j + 1 - 2a_x.$$

Thus, if $i + j + 1 - 2a_x \neq 0$, then $d(p^c_1, v^x_i) \neq d(p^c_1, v^x_j)$. Now, if $i + j + 1 - 2a_x = 0$, then either $j = a_x$ and $i = a_x - 1$, in which case, by Claim 4(iii),

$$d(t^{x,c}_1, v^x_i) = 5d + 2 > 5d + 1 = d(t^{x,c}_1, v^x_j),$$

or $j > a_x$ and $i < a_x - 1$, in which case, by Claim 4(iii),

$$d(t^{x,c}_1, v^x_j) = 5d + 1 + a_x - j = 5d + 2 + i - a_x < 5d + 4 + i - a_x = d(t^{x,c}_1, v^x_i).$$

Finally, if $a_x \leq i < j$, then by Claim 4(ii),

$$d(p^c_1, v^x_j) = 5d + 1 + a_x - j < 5d + 1 + a_x - i = d(p^c_1, v^x_i).$$

Since for any $t \in V(G) \setminus V(G_x)$ and $k \in [d]$, $d(t, v^x_k) = d(t, w^x_k)$, we conclude similarly if either $u = v^x_i$ and $v = w^x_j$, or $u = w^x_i$ and $v = w^x_j$ for two distinct $i, j \in \{0, \ldots, d+1\}$. Assume, henceforth, that $v \notin \bigcup_{x \in X} V(G_x)$. If v does not belong to a path connecting G_x to some clause gadget, then

$$d(v^x_{\phi(x)}, v) \geq \min_{c \in C} d(V(G_x), V(G_c)) \geq 3d > d + 1 \geq d(v^x_{\phi(x)}, u)$$

by Claim 3(iii) and (iv). Suppose therefore that $v \in V(P^{x,\ell}_i)$, where $i \in \{1, 2\}$ and $\ell \in \{c, \bar{c}\}$ for some clause $c = (x \leq a_x, y \leq a_y, z \leq a_z)$ containing x. W.l.o.g., let us assume that $u = v^x_j$ where $j \in \{0, \ldots, d+1\}$.
Assume first that $\ell = c$ and $i = 1$. Let $P^{x,c}_1 = z_0 \ldots z_{4d-a_x}$, where $z_0 = b^c$ and $z_{4d-a_x} = u^x_1$. Let $v = z_k$, where $k \in [4d - a_x - 1]$. If $j \leq a_x - 1$, then by Claim 4(ii), the shortest path from p^c_1 to v^x_j contains $P^{x,c}_1$ as a subpath, which implies in particular that $d(p^c_1, v) < d(p^c_1, u)$. Suppose therefore that $j \geq a_x$. Then, by Claim 4(ii),

$$d(p^c_1, u) - d(p^c_1, v) = 5d + 1 + a_x - j - (d + k + 2).$$

Thus, if $5d + 1 + a_x - j - (d + k + 2) \neq 0$, then $d(p^c_1, u) \neq d(p^c_1, v)$. Now, if $5d + 1 + a_x - j - (d + k + 2) = 0$, then by Claim 4(iii),

$$d(t^{x,c}_1, u) = 5d + 1 + a_x - j = d + k + 2 < d + k + 4 = d(t^{x,c}_1, v).$$

Second, assume that $\ell = c$ and $i = 2$. Let $P_2^{x,c} = z_0 \dots z_{4d+a_x-1}$, where $z_0 = v^c$ and $z_{4d+a_x-1} = u_2^x$. Let $v = z_k$, where $k \in [4d + a_x - 2]$ (note that since v does not belong to the core of a clause gadget or a pendant claw by assumption, in fact $k \geq 2$). If $j \geq a_x$, then by Claim 4(ii), the shortest path from p_1^c to u contains $P_2^{x,c}$ as a subpath, which implies in particular that $d(p_1^c, v) < d(p_1^c, u)$. Otherwise, $j \leq a_x - 1$, in which case

$$d(p_1^c, u) - d(p_1^c, v) = 5d + 2 + j - a_x - (k + 1).$$

Thus, if $5d+2+j-a_x-(k+1) \neq 0$, then $d(p_1^c, u) \neq d(p_1^c, v)$. Now, if $5d+2+j-a_x-(k+1) = 0$, then $j < a_x - 1$ since $k < 5d$, and so, by Claim 4(iii),

$$d(t_1^{x,c}, u) = 5d + 4 + j - a_x = k + 3 > k + 1 = d(t_1^{x,c}, v).$$

Third, assume that $\ell = \overline{c}$ and $i = 1$. Let $P_1^{x,\overline{c}} = z_0 \dots z_{5d-a_x}$, where $z_0 = v^{\overline{c}}$ and $z_{5d-a_x} = u_1^x$. Let $v = z_k$, where $k \in [5d - a_x - 1]$ (note that since v does not belong to the core of a clause gadget or a pendant claw by assumption, in fact $k \geq 2$). If $j \leq a_x + 1$, then by Claim 5(ii), the shortest path from $p_1^{\overline{c}}$ to v_j^x contains $P_1^{x,\overline{c}}$ as a subpath which implies in particular that $d(p_1^{\overline{c}}, v) < d(p_1^{\overline{c}}, u)$. Suppose therefore that $j \geq a_x + 2$. Then, by Claim 5(ii),

$$d(p_1^{\overline{c}}, v_j^x) - d(p_1^{\overline{c}}, z_k) = 5d + 3 + a_x - j - (k + 1).$$

Thus, if $5d + 3 + a_x - j - (k + 1) \neq 0$, then $d(p_1^{\overline{c}}, v_j^x) \neq d(p_1^{\overline{c}}, z_k)$. Now, if $5d + 3 + a_x - j - (k + 1) = 0$, then $j > a_x + 2$ since $k < 5d$, and so, by Claim 5(iii),

$$d(t_1^{x,\overline{c}}, v_j^x) = 5d + 5 + a_x - j = k + 3 > k + 1 = d(t_1^{x,\overline{c}}, z_k).$$

Assume finally that $\ell = \overline{c}$ and $i = 2$. Let $P_2^{x,\overline{c}} = z_0 \dots z_{3d+a_x}$, where $z_0 = b^{\overline{c}}$ and $z_{3d+a_x} = u_2^x$. Let $v = z_k$, where $k \in [3d + a_x - 1]$. If $j \geq a_x + 2$, then by Claim 5(ii), the shortest path from $p_1^{\overline{c}}$ to u contains $P_2^{x,\overline{c}}$ as a subpath, which implies in particular that $d(p_1^{\overline{c}}, v) < d(p_1^{\overline{c}}, u)$. Suppose therefore that $j \leq a_x + 1$. Then, by Claim 5(ii),

$$d(p_1^{\overline{c}}, v_j^x) - d(p_1^{\overline{c}}, z_k) = 5d + 1 + j - a_x - (d + k + 2).$$

Thus, if $5d + 1 + j - a_x - (d + k + 2) \neq 0$, then $d(p_1^{\overline{c}}, v_j^x) \neq d(p_1^{\overline{c}}, z_k)$. Now, if $5d + 1 + j - a_x - (d + k + 2) = 0$, then $j < a_x + 1$ since $k < 4d$, and so, by Claim 5(iii),

$$d(t_1^{x,\overline{c}}, v_j^x) = 5d + 1 + j - a_x = d + k + 2 < d + k + 4 = d(t_1^{x,\overline{c}}, z_k).$$

In all the subcases, we conclude that there exists $w \in S$ such that $d(w, u) \neq d(w, v)$.

Case 4. *None of the above.* First, note that p is distinguished by S since it is the unique vertex of G at distance 1 from t_1. Second, t_2 is distinguished by S since it is the unique vertex of G at distance 2 from t_1 and distance 4 from t_1^c and $t_1^{\overline{c}}$ for all $c \in C$. Thus, in this last case, we can assume that both u and v belong either to paths connecting gadgets or to some path P_{b^ℓ}, where $\ell \in \{c, \overline{c} \mid c \in C\}$. Assume first that $u \in V(P_\ell)$ for some $\ell \in \{c, \overline{c} \mid c \in C\}$. If $v \in V(P_\ell)$ as well, then surely $d(t_1, u) \neq d(t_1, v)$. If $v \in V(P_q)$ for some $q \in \{c, \overline{c} \mid c \in C\}$ different from ℓ, then $d(p_1^\ell, u) < d(p_1^\ell, v)$ since the unique shortest path from p_1^ℓ to v contains P_ℓ as a subpath. Finally, if there exists $q \in \{c, \overline{c} \mid c \in C\}$ such that v belongs to P_{b_q} or to some path connecting H_q to a variable gadget, then $d(t_1, v) > d(t_1, v^q) \geq d(t_1, u)$.

Second, assume that $u \in V(P_i^{x,\ell})$, where $i \in [2]$ and $\ell \in \{c, \overline{c}\}$ for some clause c containing variable x. Note that by the previous paragraph, we may assume that $v \notin \bigcup_{q \in C} V(P_q) \cup V(P_{\overline{q}})$. Suppose first that $v \in V(P_j^{y,q})$, where $j \in [2]$ and $q \in \{c', \overline{c}'\}$ for some clause c' containing variable y. Note that $d(t_1^\ell, u) = d(t_1, u)$. So, if $q \neq \ell$, then either $d(t_1, u) \neq d(t_1, v)$, or

$$d(t_1^\ell, v) - d(t_1^\ell, u) = d(t_1^\ell, p) + d(t_1, v) - 1 - d(t_1^\ell, u) = d(t_1, v) + 2 - d(t_1, u) = 2.$$

Thus, assume that $q = \ell$. Suppose first that $x = y$. If $i = j$, then surely $d(p_1^\ell, u) \neq d(p_1^\ell, v)$. Otherwise, assume, w.l.o.g., that u belongs to the path containing $w^{x,\ell}$. Note that $d(t_1^{x,\ell}, u) = d(p_1^\ell, u)$. Then, either $d(p_1^\ell, u) \neq d(p_1^\ell, v)$, or

$$d(t_1^{x,\ell}, v) - d(t_1^{x,\ell}, u) = d(t_1^{x,\ell}, v^\ell) + d(p_1^\ell, v) - 1 - d(t_1^{x,\ell}, u) = d(p_1^\ell, v) + 2 - d(p_1^\ell, u) = 2.$$

Second, suppose that $x \neq y$. If u belongs to the path containing $w^{x,\ell}$, then we argue as previously. By symmetry, we may also assume that v does not belong to the path containing $w^{y,\ell}$. This implies, in particular, that $i = j$ and b^ℓ is the endpoint in H_ℓ of both $P_i^{x,\ell}$ and $P_j^{y,\ell}$. Thus, $d(v_{\phi(x)}^x, u) \leq d(v_{\phi(x)}^x, u_i^x) + d(u_i^x, b^\ell) \leq d + 4d$. First, note that if a shortest path P from $v_{\phi(x)}^x$ to v contains $P_i^{x,\ell}$ as a subpath, then since $u \in V(P_i^{x,\ell})$, it follows that $d(v_{\phi(x)}^x, u) < d(v_{\phi(x)}^x, v)$. Hence, we may assume that P contains a vertex in G_y or both v^ℓ and b^ℓ. By Claim 3(ii), if P contains a vertex in G_y, then

$$d(v_{\phi(x)}^x, v) > d(V(G_x), V(G_y)) \geq 6d > 5d \geq d(v_{\phi(x)}^x, u).$$

Otherwise, P contains v^ℓ and b^ℓ, and so, letting $t \in [2] \setminus \{i\}$, we get that

$$\mathsf{lgt}(P) \geq d(v_{\phi(x)}^x, u_t^x) + d(u_t^x, v^\ell) + d(v^\ell, b^\ell) + d(b^\ell, v) \geq 1 + 4d + d + 1 > 5d \geq d(v_{\phi(x)}^x, u).$$

Suppose finally that $u \in V(P_{b^\ell})$ for some $\ell \in \{c, \overline{c} \mid c \in C\}$. By the two previous paragraphs, we may assume that $v \in V(P_{b^q})$ for some $q \in \{c, \overline{c} \mid c \in C\}$. If $q = \ell$, then surely $d(p_1^\ell, u) \neq d(p_1^\ell, v)$. Otherwise, by Claim 3(i),

$$d(p_1^\ell, v) \geq d(V(H_\ell), V(H_q)) = 4d > d + 1 \geq d(p_1^\ell, u), \text{ which concludes case 4.}$$

By the above case analysis, we infer that, for any $u, v \in V(G)$, there exists $w \in S$ such that $d(w, u) \neq d(w, v)$, that is, S is a resolving set of G. Since $|S| = k$, it follows that (G, k) is a YES-instance for METRIC DIMENSION.

Conversely, assume that (G, k) is a YES-instance for METRIC DIMENSION and let S be a resolving set of size at most k. By Observation 1, for any clause $c \in C$ and any variable $x \in X$ appearing in c,

$$|S \cap \{p_1^c, p_2^c\}| \geq 1,\ |S \cap \{p_1^{\overline{c}}, p_2^{\overline{c}}\}| \geq 1,\ |S \cap \{t_1^c, t_2^c\}| \geq 1, \text{ and } |S \cap \{t_1^{\overline{c}}, t_2^{\overline{c}}\}| \geq 1. \quad (1)$$

$$|S \cap \{t_1^{x,c}, t_2^{x,c}\}| \geq 1 \text{ and } |S \cap \{t_1^{x,\overline{c}}, t_2^{x,\overline{c}}\}| \geq 1. \quad (2)$$

$$|S \cap \{t_1, t_2\}| \geq 1. \quad (3)$$

Consider now a variable x. Since any path from a vertex in $V(G) \setminus V(G_x)$ to a vertex in $\{v_i^x, w_i^x \mid i \in [d]\}$ contains u_1^x or u_2^x, and, for any $i \in [d]$ and $u \in \{u_1^x, u_2^x\}$, $d(u, v_i^x) = d(u, w_i^x)$, no vertex in $V(G) \setminus \{v_i^x, w_i^x \mid i \in [d]\}$ can resolve v_i^x and w_i^x for any $i \in [d]$. It follows that

$$|S \cap \{v_i^x, w_i^x \mid i \in [d]\}| \geq 1. \tag{4}$$

Now, note that S has size at most $k = |X| + 10|C| + 1$, and so, equality must in fact hold in every inequality of Equations (1)–(4). W.l.o.g., let us assume that $t_1 \in S$ and that, for every clause $c \in C$ and variable $x \in X$ appearing in c, we have that $p_1^c, p_1^{\overline{c}}, t_1^c, t_1^{\overline{c}}, t_1^{x,c}, t_1^{x,\overline{c}} \in S$.

For every variable $x \in X$, assume, w.l.o.g., that $S \cap \{v_i^x, w_i^x \mid i \in [d]\} = S \cap \{v_i^x \mid i \in [d]\}$, and let $i_x \in [d]$ be the index of the vertex in $S \cap \{v_i^x \mid i \in [d]\}$. We contend that the assignment which sets each variable x to i_x satisfies every clause in C. Indeed, consider a clause $c = (x \leq a_x, y \leq a_y, z \leq a_z)$. We first aim to show that, for every $w \in S \setminus \{V(G_x) \cup V(G_y) \cup V(G_z)\}$ and $\ell \in \{c, \overline{c}\}$, $d(w, \ell) = d(w, p_2^\ell)$. Note that it suffices to show that any shortest path from $w \in S \setminus \{V(G_x) \cup V(G_y) \cup V(G_z)\}$ to $\ell \in \{c, \overline{c}\}$ contains v^ℓ, as then $d(w, \ell) = d(w, v^\ell) + 1 = d(w, p_2^\ell)$. Now, if $w \in V(G_t)$ for some $t \in \{c', \overline{c'} \mid c' \in C\}$ different from ℓ, then this readily follows from Claim 3(i); and if $w \in V(G_t)$ for some $t \in X \setminus \{x, y, z\}$, then this readily follows from Claim 3(iv). If $w = t_1^{r,q}$ for some $r \in X$ and $q \in \{c', \overline{c'} \mid c' \in C\}$, then $d(t_1^{r,q}, \ell) = d(t_1^{r,q}, v^q) + d(v_q, \ell)$, and so, by Claim 3(i), any path from w to ℓ contains v^ℓ. Finally, if $w \in \{t_1^{c'}, t_1^{\overline{c}'} \mid c' \in C\}$, then clearly any shortest path from w to ℓ contains v^ℓ.

Since S is a resolving set, it follows that, for every clause $c \in C$, there exist $t, f \in \{x, y, z\}$ such that $d(v_{i_t}^t, c) \neq d(v_{i_t}^t, p_2^c)$ and $d(v_{i_f}^f, \overline{c}) \neq d(v_{i_f}^f, p_2^{\overline{c}})$. Now, by Claim 4(i) and (ii), if $i_t > a_t$, then $d(v_{i_t}^t, c) = 5d + 1 + a_t - i_t = d(v_{i_t}^t, v^c) + 1 = d(v_{i_t}^t, p_2^c)$, a contradiction to our assumption. Therefore, $i_t \leq a_t$. Similarly, if $i_f \leq a_f$, then by Claim 5(i) and (ii), $d(v_{i_f}^f, \overline{c}) = 5d + 1 + i_f - a_f = d(v_{i_f}^f, v^{\overline{c}}) + 1 = d(v_{i_f}^f, p_2^{\overline{c}})$, a contradiction to our assumption. Therefore, $i_f > a_f$, and so, the assignment constructed indeed satisfies every clause in C. ◀

3 The Vertex Cover Number and the Distance to clique

In this section, we prove that Metric Dimension parameterized by either the vertex cover number or the distance to clique does not admit a polynomial kernel unless NP $\subseteq$ coNP$/poly$. Both reductions are similar ones from the SAT problem, in which we are given a conjunctive normal form (CNF) formula ϕ on n variables and m clauses, and we are asked whether there exists an assignment of either true or false to each of the variables, such that ϕ is true (satisfied). SAT is known to not admit a polynomial kernel unless NP $\subseteq$ coNP$/poly$ [16]. We first prove that Metric Dimension parameterized by the vertex cover number does not admit a polynomial kernel unless NP $\subseteq$ coNP$/poly$, with the same result for distance to clique to follow after from a small modification to this reduction.

▶ **Theorem 6.** Metric Dimension *parameterized by the vertex cover number does not admit a polynomial kernel unless* NP $\subseteq$ coNP$/poly$.

Proof. ($\star$) By a reduction from SAT, we prove that Metric Dimension parameterized by the vertex cover number does not admit a polynomial kernel unless NP $\subseteq$ coNP$/poly$. Let ϕ be an instance of SAT, *i.e.*, a SAT formula on n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. Since any SAT formula on n variables trivially has at most $3^n - 1$ unique clauses, we may assume that $m \leq 3^n - 1$.

From ϕ, we construct an instance (G, k) of Metric Dimension as follows. For each $i \in [n]$, construct a cycle $(t_i a_i^1 b_i^1 f_i b_i^2 a_i^2 t_i)$ on 6 vertices, and let $I_i := \{a_i^1, a_i^2, b_i^1, b_i^2\}$. Construct a path $g^1 g g^2$ on 3 vertices and, for each $i \in [n]$, make both f_i and t_i adjacent to g. For each $j \in [m]$, add a pair of vertices c_j^1 and c_j^2, and let $C_j := \{c_j^1, c_j^2\}$. For each $j \in [m]$, make c_j^2 adjacent to both f_i and t_i for each $i \in [n]$. For each $j \in [m]$ and each $i \in [n]$, if $x_i = \texttt{True}$ does not satisfy the clause $\mathcal{C}_j$ in ϕ, then make c_j^1 adjacent to t_i, and if $x_i = \texttt{False}$ does not satisfy $\mathcal{C}_j$, then make c_j^1 adjacent to f_i. Let $\alpha = \lceil n \cdot \log_2 3 \rceil$, and, for each $\ell \in [\alpha]$, construct a path $z_\ell^1 z_\ell z_\ell^2$ on 3 vertices. For each $j \in [m]$, consider the binary representation $\text{bin}(j)$ of j, and connect both c_j^1 and c_j^2 with z_ℓ if $\text{bin}(j)[\ell] = 1$, where $[\ell]$ is the ℓ^{th} bit of j in its binary representation from right to left. Finally, construct a clique on the vertices $z_1, \ldots, z_\alpha, g$. This completes the construction of G (see Figure 4).

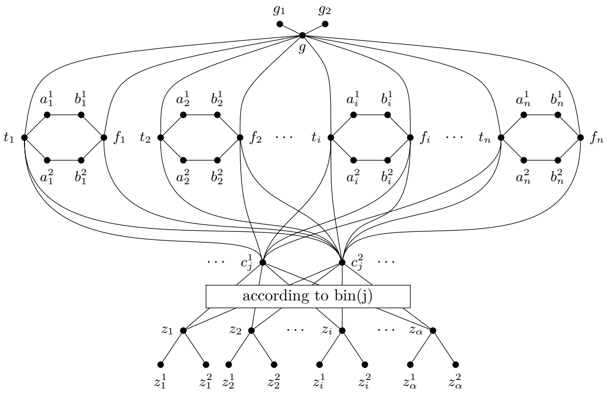


Figure 4 Illustration of the graph G constructed in the proof of Theorem 6. The vertices $z_1, \ldots, z_\alpha, g$ are in a clique that is not drawn. In this particular case, ϕ has a clause $(\overline{x_1} \vee x_2 \vee \overline{x_n})$.

To simplify notation for the proof, let $I := I_1 \cup \cdots \cup I_n$. Set $k = n + \alpha + 1$. Note that the vertex cover number of G is at most $4n + \alpha + 1$ since $\{g\} \cup \{a_i^1, a_i^2, t_i, f_i, z_\ell \mid i \in [n], \ell \in [\alpha]\}$ is a vertex cover of G. We now show that the instance ϕ is satisfiable if and only if (G, k) is a Yes-instance for Metric Dimension. We just sketch the proof from here. To prove that if ϕ is satisfiable, then (G, k) is a Yes-instance for Metric Dimension, we build a resolving set R of G of size k as follows. First, put the vertices of $\{g_1, z_1^1, \ldots, z_1^1\}$ in R. Then, for all $i \in [n]$, if, according to the satisfying truth value assignment of ϕ, $x_i = \texttt{True}$ ($x_i = \texttt{False}$, resp.), then add a_i^1 (b_i^1, resp.) to R. Clearly, $|R| = k$, and it is not difficult to check that R is a resolving set of G.

Now, we prove that if (G, k) is a Yes-instance for Metric Dimension, then ϕ is satisfiable. For any resolving set R of G, one can show that since $|R| \leq n + \alpha + 1$, then $|R \cap \{g^1, g^2\}| = 1$, $|R \cap \{z_\ell^1, z_\ell^2\}| = 1$ for all $\ell \in [\alpha]$, and $|R \cap I_i| = 1$ for all $i \in [n]$. W.l.o.g., assume that $\{g^1, z_1^1, \ldots, z_\alpha^1\} \subset R$. Consider $j \in [m]$. It can be shown that no vertex in $\{g^1, z_1^1, \ldots, z_\alpha^1\}$ can resolve the two vertices of C_j, and thus, there must exist $w \in R \cap I$ such that $d(w, c_j^1) \neq d(w, c_j^2)$. Since for every $i \in [n]$ such that x_i does not appear in the clause C_j, $d(u, c_j^1) = d(u, c_j^2)$ for every $u \in I_i$, there must exist $i \in [n]$ such that x_i appears in the clause C_j and $w \in R \cap I_i$. In particular, c_j^1 must be non-adjacent to one of t_i and f_i. Now, if c_j^1 is non-adjacent to t_i, then c_j^1 is adjacent to f_i, and so, $w \in \{a_i^1, a_i^2\}$, as otherwise $d(w, c_j^1) = d(w, c_j^2)$. Symmetrically, if c_j^1 is non-adjacent to f_i, then c_j^1 is adjacent to t_i, and so, $w \in \{b_i^1, b_i^2\}$, as otherwise $d(w, c_j^1) = d(w, c_j^2)$. So, the truth assignment obtained by setting a variable x_i to $\texttt{True}$ if $R \cap I_i \subseteq \{a_i^1, a_i^2\}$, and to $\texttt{False}$ otherwise, satisfies ϕ. ◀

By making the vertices of $\{C_j \mid j \in [m]\}$ into a clique in the construction of G in the proof of Theorem 6, observe that the distance to clique of the resulting graph is at most $6n + 3\alpha + 3$, and that none of the distances described in the proof change. Then, from the proof of Theorem 6 for this modified G, we obtain the following:

▶ **Theorem 7.** Metric Dimension *parameterized by the distance to clique does not admit a polynomial kernel unless* NP $\subseteq$ coNP/*poly.*

4 The Distance to Cluster and the Distance to co-cluster

In this section, we prove that Metric Dimension is FPT parameterized by either the distance to cluster or the distance to co-cluster. In fact, we show that the problem admits an exponential kernel parameterized by the distance to cluster (or co-cluster). Since the main ideas for these two parameters are the same, we focus on the distance to cluster parameter. Applying Reduction Rule 2 for false twins (instead of true twins) and defining equivalence classes over the independent sets (instead of cliques) for Reduction Rule 3, we get the similar result for the distance to co-cluster. Recall that, for a graph G, the *distance to cluster* of G is the minimum number of vertices of G that need to be deleted so that the resulting graph is a *cluster graph*, *i.e.*, a disjoint union of cliques.

▶ **Theorem 8.** Metric Dimension *is* FPT *parameterized by the distance to cluster.*

Proof. Let (G, k) be an instance of Metric Dimension and let $X \subseteq V(G)$ be such that $G - X$ is a disjoint union of cliques. To obtain a kernel for the problem, we present a set of reduction rules. The safeness of the following reduction rule is trivial.

▶ **Reduction Rule 1.** *If $V(G) \neq \emptyset$ and $k \leq 0$, then return a trivial* No*-instance.*

▶ **Reduction Rule 2.** *If there exist $u, v, w \in V(G)$ such that u, v, w are true (or false) twins, then remove u from G and decrease k by one.*

▷ Claim 9 (⋆). Reduction Rule 2 is safe.

We assume, henceforth, that Reduction Rule 2 has been exhaustively applied to (G, k). This implies, in particular, that for every clique C of $G - X$, there are at most two vertices in C with the same neighborhood in X. Since the number of distinct neighborhoods in X is at most $2^{|X|}$, each clique in $G - X$ has order at most $2^{|X|+1}$. We now aim to bound the number of cliques in $G - X$. To this end, we define a notion of *equivalence classes* over the set of cliques of $G - X$. It will easily be seen that the number of equivalence classes is at most $2^{2^{|X|+1}}$. The number of cliques in each equivalence class will then be bounded by using Reduction Rule 3.

For every clique C of $G - X$, the *signature* $\texttt{sign}(C)$ of C is the *multiset* containing the neighborhoods in X of each vertex of C, that is, $\texttt{sign}(C) = \{N(u) \cap X : u \in C\}$. For any two cliques C_1, C_2 of $G - X$, we say that C_1 and C_2 are *identical*, which we denote by $C_1 \sim C_2$, if and only if $\texttt{sign}(C_1) = \texttt{sign}(C_2)$. It is not difficult to see that $\sim$ is in fact an equivalence relation with at most $2^{2^{|X|+1}}$ equivalence classes: indeed, since the number of distinct neighborhoods in X is at most $2^{|X|}$, and at most two vertices of each clique have the same neighborhood in X, the number of distinct signatures is at most $2^{2^{|X|+1}}$. Consider now an equivalence class $\mathcal{C}$ of $\sim$. Note that since the signature of a clique is a multiset, the number of vertices in each $C \in \mathcal{C}$ is equal to $|\texttt{sign}(C)|$. For any $C_1, C_2 \in \mathcal{C}$, we say that two vertices $u \in C_1$ and $v \in C_2$ are *clones* if $N(u) \cap X = N(v) \cap X$ (in particular, if $C_1 = C_2$ and $u \neq v$, then u, v are true twins). For any $C_1, C_2 \in \mathcal{C}$ and any $u \in C_1$, we denote by $c(u, C_2)$

the set of clones of u in C_2 (note that $|c(u, C_2)| \leq 2$). Now observe that, for any two cliques $C_1, C_2 \in \mathcal{C}$, the number of pairs of true twins in C_1 and C_2 is the same: we let $t(\mathcal{C})$ be the number of pairs of true twins in each clique of $\mathcal{C}$. We highlight that there are exactly $2t(\mathcal{C})$ vertices in each clique of $\mathcal{C}$ that have true twins. The following claim for clones is the analog of Observation 1 for twins.

▷ Claim 10 (⋆). Let C_1 and C_2 be two cliques of an equivalence class $\mathcal{C}$ of $\sim$. Let $u \in C_1$ and $v \in C_2$ be clones. Then, for any $w \in V(G) \setminus (V(C_1) \cup V(C_2))$, $d(u, w) = d(v, w)$, and so, for any resolving set S of G, $S \cap (V(C_1) \cup V(C_2)) \neq \emptyset$.

It follows from the above claim that, for any equivalence class $\mathcal{C}$ of $\sim$ and any resolving set S, S contains at least $|\mathcal{C}| - 1$ vertices in $V(\mathcal{C}) := \bigcup_{C \in \mathcal{C}} V(C)$. We now present an upper bound on the size of $S \cap V(\mathcal{C})$ when $|\mathcal{C}| \geq |X| + 2$.

▷ Claim 11 (⋆). For every equivalence class $\mathcal{C}$ of $\sim$, if $|\mathcal{C}| \geq |X| + 2$, then, for any minimum resolving set S of G, $|S \cap V(\mathcal{C})| \leq |X| + |\mathcal{C}| \cdot \max\{1, t(\mathcal{C})\}$.

Let $\mathcal{C}$ be an equivalence class of $\sim$, and let S be a resolving set of G. For every $i \geq 0$, we denote by $\mathcal{C}^S_{=i}$ ($\mathcal{C}^S_{\geq i}$, resp.) the set of cliques $C \in \mathcal{C}$ such that $|S \cap V(C)| = i$ ($|S \cap V(C)| \geq i$).

▷ Claim 12 (⋆). Let $\mathcal{C}$ be an equivalence class of $\sim$ such that $|\mathcal{C}| \geq |X| + 2$. Then, for any minimum resolving set S of G, the following hold:

(i) if $t(\mathcal{C}) = 0$, then $|\mathcal{C}^S_{=0}| \leq 1$ and $|\mathcal{C}^S_{\geq 2}| \leq |X| + 1$;
(ii) if $t(\mathcal{C}) \neq 0$, then $|\mathcal{C}^S_{\geq t(\mathcal{C})+1}| \leq |X| + 1$.

The above claim states that if some equivalence class $\mathcal{C}$ of $\sim$ contains at least $|X| + 3$ cliques, then, for any minimum resolving set S of G, if $t(\mathcal{C}) = 0$, then $\mathcal{C}^S_{=1} \neq \emptyset$, and otherwise, $\mathcal{C}^S_{=t(\mathcal{C})} \neq \emptyset$. The following reduction rule is based on this claim.

▶ **Reduction Rule 3.** *If there exists an equivalence class $\mathcal{C}$ of $\sim$ such that $|\mathcal{C}| \geq 2^{|X|+2} + |X| + 2$, then remove a clique $C \in \mathcal{C}$ from G and reduce k by* $\max\{1, t(\mathcal{C})\}$.

▷ Claim 13 (⋆). Reduction Rule 3 is safe.

Now observe that once Reduction Rule 3 has been exhaustively applied to (G, k), each equivalence class of $\sim$ contains at most $2^{|X|+2} + |X| + 1$ cliques. Since there are at most $2^{2^{|X|+1}}$ equivalence classes and each clique of $G - X$ has size at most $2^{|X|+1}$, we conclude that G contains at most $2^{2^{|X|+1}} \cdot (2^{|X|+2} + |X| + 1) \cdot 2^{|X|+1} + |X|$ vertices. ◀

5 Conclusion

As the Metric Dimension problem is W[2]-hard when parameterized by the solution size [23], the next natural step is to understand its parameterized complexity under structural parameterizations. We continued this line of research, following in the steps of [3, 13, 20], and more recently [7, 29]. Our most technical result is a proof that the Metric Dimension problem is W[1]-hard when parameterized by the feedback vertex set number of the graph. We thereby improved the result by Bonnet and Purohit [7] that states the problem is W[1]-hard when parameterized by the treewidth, and answered an open question in [23]. It is easy to see that the problem admits an FPT algorithm when parameterized by the larger parameter, the vertex cover number of the graph. On the positive side, we proved that the problem admits FPT algorithms when parameterized by the distance to cluster and the distance to co-cluster, which are smaller parameters than the vertex cover number.

Although this work advances the understanding of structural parameterizations of Metric Dimension, it falls short of completing the picture (see Figure 1). We find it hard to extend the positive results to the parameters like the minimum clique cover, the distance to disjoint paths, feedback edge set, and the bandwidth. It would be interesting to find FPT algorithms or prove that such algorithms are highly unlikely to exist for these parameters. The FPT algorithm parameterized by the treedepth in [20] relies on the meta-result. Is it possible to get an FPT algorithm whose running time is a single or double exponent in the treedepth? It would also be interesting to investigate the problem parameterized by the distance to cograph. Recall that the problem is polynomial-time solvable in cographs [14].

Bonnet and Purohit [7] conjectured that the problem is W[1]-hard even when parameterized by the treewidth plus the solution size. Towards resolving this conjecture, an interesting question would be to investigate whether the problem admits an FPT algorithm when parameterized by the feedback vertex set number plus the solution size. Note that even an XP algorithm parameterized by the feedback vertex set number is not apparent.

References

1 L. Babai. On the complexity of canonical labelling of strongly regular graphs. *SIAM J. Comput.*, 9(1):212–216, 1980.

2 Z. Beerliova, F. Eberhard, T. Erlebach, A. Hall, M. Hoffman, M. Mihalák, and L. S. Ram. Network discovery and verification. *IEEE J. Sel. Area Comm.*, 24(12):2168–2181, 2006.

3 R. Belmonte, F. V. Fomin, P. A. Golovach, and M. S. Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM J. Discrete Math.*, 31(2):1217–1243, 2017.

4 Y. Ben-Haim, S. Gravier, A. Lobstein, and J. Moncel. Adaptive identification in graphs. *J. Comb. Theory, Ser. A*, 115(7):1114–1126, 2008.

5 J. Bensmail, D. Mazauric, F. Mc Inerney, N. Nisse, and S. Pérennes. Sequential metric dimension. *Algorithmica*, 82(10):2867–2901, 2020.

6 J. Bensmail, F. Mc Inerney, and N. Nisse. Metric dimension: from graphs to oriented graphs. *Discrete Applied Mathematics*, in press. doi:10.1016/j.dam.2020.09.013.

7 E. Bonnet and N. Purohit. Metric dimension parameterized by treewidth. *Algorithmica*, 83:2606–2633, 2021.

8 B. Bosek, P. Gordinowicz, J. Grytczuk, N. Nisse, J. Sokól, and M. Sleszynska-Nowak. Centroidal localization game. *Electronic Journal of Combinatorics*, 25(4):P4.62, 2018.

9 B. Bosek, P. Gordinowicz, J. Grytczuk, N. Nisse, J. Sokól, and M. Sleszynska-Nowak. Localization game on geometric and planar graphs. *Discrete Applied Mathematics*, 251:30–39, 2018.

10 K. Bringmann, D. Hermelin, M. Mnich, and E. J. van Leeuwen. Parameterized Complexity Dichotomy for Steiner Multicut. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 157–170, 2015.

11 G. Chartrand, L. Eroh, M. Johnson, and O. Oellermann. Resolvability in graphs and the metric dimension of a graph. *Discrete Applied Mathematics*, 105(1-3):99–113, 2000.

12 J. Díaz, O. Pottonen, M. J. Serna, and E. J. van Leeuwen. Complexity of metric dimension on planar graphs. *J. Comput. Syst. Sci.*, 83(1):132–158, 2017.

13 D. Eppstein. Metric dimension parameterized by max leaf number. *Journal of Graph Algorithms and Applications*, 19(1):313–323, 2015.

14 L. Epstein, A. Levin, and G. J. Woeginger. The (weighted) metric dimension of graphs: Hard and easy cases. *Algorithmica*, 72(4):1130–1171, 2015.

15 A. Estrada-Moreno, J. A. Rodriguez-Velázquez, and I. G. Yero. The k-metric dimension of a graph. *Applied Mathematics and Information Sciences*, 9(6):2829–2840, 2015.

16 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

17 F. Foucaud, R. Klasing, and P. J. Slater. Centroidal bases in graphs. *Networks*, 64(2):96–108, 2014.

18 F. Foucaud, G. B. Mertzios, R. Naserasr, A. Parreau, and P. Valicov. Identification, location-domination and metric dimension on interval and permutation graphs. II. algorithms and complexity. *Algorithmica*, 78(3):914–944, 2017.

19 M. R. Garey and D. S. Johnson. *Computers and Intractability – A guide to NP-completeness.* W.H. Freeman and Company, 1979.

20 T. Gima, T. Hanaka, M. Kiyomi, Y. Kobayashi, and Y. Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 2022.

21 G. Z. Gutin, M. S. Ramanujan, F. Reidl, and M. Wahlström. Alternative parameterizations of metric dimension. *Theoretical Computer Science*, 806:133–143, 2020.

22 F. Harary and R. A. Melter. On the metric dimension of a graph. *Ars Combinatoria*, 2:191–195, 1976.

23 S. Hartung and A. Nichterlein. On the parameterized and approximation hardness of metric dimension. In *Proceedings of the 28th Conference on Computational Complexity, CCC*, pages 266–276. IEEE Computer Society, 2013.

24 J. Haslegrave, R. A. B. Johnson, and S. Koch. Locating a robber with multiple probes. *Discrete Math.*, 341(1):184–193, 2018.

25 S. Hoffmann, A. Elterman, and E. Wanke. A linear time algorithm for metric dimension of cactus block graphs. *Algorithmica*, 72(4):1130–1171, 2015.

26 M. A. Johnson. Structure-activity maps for visualizing the graph variables arising in drug design. *J. Biopharm. Statist.*, 3:203–236, 1993.

27 M. G. Karpovsky, K. Chakrabarty, and L. B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Trans. Information Theory*, 44(2):599–611, 1998.

28 D. Kuziak and I. G. Yero. Metric dimension related parameters in graphs: A survey on combinatorial, computational and applied results. *arXiv*, 2021. arXiv:2107.04877.

29 S. Li and M. Pilipczuk. Hardness of metric dimension in graphs of constant treewidth. In *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *LIPIcs*, pages 24:1–24:13, 2021.

30 R. A. Melter and I. Tomescu. Metric bases in digital geometry. *Comput. Vision Graphics Image Process.*, 25:113–121, 1984.

31 P. J. Slater. Leaves of trees. In *Proceedings of the Sixth Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 549–559. Congressus Numerantium, No. XIV. Utilitas Mathematica, 1975.

32 P. J. Slater. Domination and location in acyclic graphs. *Networks*, 17(1):55–64, 1987.

33 R. C. Tillquist, R. M. Frongillo, and M. E. Lladser. Getting the lay of the land in discrete space: A survey of metric dimension and its applications. *arXiv*, 2021. arXiv:2104.07201.

34 R. C. Tillquist, R. M. Frongillo, and M. E. Lladser. Truncated metric dimension for finite graphs. *arXiv*, 2021. arXiv:2106.14314.

35 R. C. Tillquist and M. E. Lladser. Low-dimensional representation of genomic sequences. *Journal of Mathematical Biology*, 79:1–29, 2019.

36 R. Ungrangsi, A. Trachtenberg, and D. Starobinski. An implementation of indoor location detection systems based on identifying codes. In *Proc. INTELLCOM 2004*, volume 3283 of *LNCS*, pages 175–189, 2004.

Graph Similarity Based on Matrix Norms

Timo Gervens
RWTH Aachen University, Germany

Martin Grohe
RWTH Aachen University, Germany

Abstract

Quantifying the similarity between two graphs is a fundamental algorithmic problem at the heart of many data analysis tasks for graph-based data. In this paper, we study the computational complexity of a family of similarity measures based on quantifying the mismatch between the two graphs, that is, the "symmetric difference" of the graphs under an optimal alignment of the vertices. An important example is similarity based on graph edit distance. While edit distance calculates the "global" mismatch, that is, the number of edges in the symmetric difference, our main focus is on "local" measures calculating the maximum mismatch per vertex.

Mathematically, our similarity measures are best expressed in terms of the adjacency matrices: the mismatch between graphs is expressed as the difference of their adjacency matrices (under an optimal alignment), and we measure it by applying some matrix norm. Roughly speaking, global measures like graph edit distance correspond to entrywise matrix norms like the Frobenius norm and local measures correspond to operator norms like the spectral norm.

We prove a number of strong NP-hardness and inapproximability results even for very restricted graph classes such as bounded-degree trees.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Complexity theory and logic

Keywords and phrases graph similarity, approximate graph isomorphism, graph matching

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.52

Related Version *Full Version*: `https://arxiv.org/abs/2207.00090` [11]

Funding *Timo Gervens*: This work was supported by the DFG grant GR 1492/20-1.

1 Introduction

Graphs are basic models ubiquitous in science and engineering. They are used to describe a diverse range of objects and processes from chemical compounds to social interactions. To understand and classify graph models, we need to compare graphs. Since data and models are not always guaranteed to be exact, it is essential to understand what makes two graphs similar or dissimilar, and to be able to compute similarity efficiently. There are many different approaches to similarity, for example, based on edit-distance (e.g. [3,6,27]), spectral similarity (e.g. [16,29,30]), optimal transport (e.g. [5,23,25]), or behavioral equivalence (e.g. [28,31]). This is only natural, because the choice of a "good" similarity measure will usually depend on the application. While graph similarity has received considerable attention in application areas such as computer vision (see, for example, [7,9]) and network science (see, for example, [8]), theoretical computer scientists have not explored similarity systematically; only specific "special cases" such as isomorphism [14] and bisimilarity [28] have been studied to great depth. Yet it seems worthwhile to develop a *theory of graph similarity* that compares different similarity measures, determines their algorithmic and semantic properties, and thus gives us a better understanding of their suitability for various kinds of applications. We see our paper as one contribution to such a theory.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 52; pp. 52:1–52:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Maybe the simplest graph similarity measure is based on graph edit distance: the *edit distance* between graphs G, H (for simplicity of the same order) is the minimum number of edges that need to be added and deleted from G to obtain a graph ismorphic to H.

In this paper, we study the computational complexity of a natural class of similarity measures that generalize the similarity derived from edit distance of graphs. In general, we view similarity as proximity with respect to some metric. A common way of converting a graph metric d into a similarity measure s is to let $s(G, H) := \exp(-\beta \cdot d(G, H))$ for some constant $\beta > 0$. For our considerations the transformation between distance and similarity is irrelevant, so we focus directly on the metrics. In terms of computational complexity, computing similarity tends to be a hard algorithmic problem. It is known that computing edit distance, exactly or approximately, is NP-hard [3, 13, 18, 27] even on very restricted graph classes. In fact, the problem is closely related to the quadratic assignment problem [22, 26], which is notorious for being a very hard combinatorial optimization problem also for practical, heuristic approaches. Within the spectrum of similarities, the "limit" case of graph isomorphism shows that overall the complexity of graph similarity is far from trivial.

The metrics we study in this paper are based on minimizing the mismatch between two graphs. For graphs G, H with the same vertex set V, we define their *mismatch graph* $G - H$ to be the graph with vertex set V and edge set $E(G) \triangle E(H)$, the symmetric difference between the edge sets of the graphs. We assign *signs* to edges of the mismatch graph to indicate which graph they come from, say, a positive sign to the edges in $E(G) \setminus E(H)$ and a negative sign to the edges in $E(H) \setminus E(G)$. To quantify the mismatch between our graphs we introduce a *mismatch norm* μ on signed graphs that satisfies a few basic axioms such as subadditivity as well as invariance under permutations and under flipping the signs of all edges. Now for graphs G, H, not necessarily with the same vertex set, but for simplicity of the same order,[1] we define the distance $\mathrm{dist}_\mu(G, H)$ to be the minimum of $\mu(G^\pi - H)$, where π ranges over all bijective mappings from $V(G)$ to $V(H)$ and G^π is the image of G under π.

The simplest mismatch norm μ_{ed} just counts the number of edges of the mismatch graph $G - H$, ignoring their signs. Then the associated distance $\mathrm{dist}_{\mathrm{ed}}(G, H)$ is the edit distance between G and H. (Note that we write $\mathrm{dist}_{\mathrm{ed}}$ instead of the clunky $\mathrm{dist}_{\mu_{\mathrm{ed}}}$; we will do the same for other mismatch norms discussed here.) Another simple yet interesting mismatch norm is μ_{deg} measuring the maximum degree of the mismatch graph, again ignoring the signs of the edges. Then $\mathrm{dist}_{\mathrm{deg}}(G, H)$ measures how well we can align the two graphs in order to minimize the "local mismatch" at every node. Hence an alignment where at every vertex there is a mismatch of one edge yields a smaller $\mathrm{dist}_{\mathrm{deg}}$ than an alignment that is perfect at all nodes expect one, where it has a mismatch of, say, $n/2$, where n is the number of vertices. For edit distance it is the other way round. Depending on the application, one or the other may be preferable. Another well-known graph metric that can be described via the mismatch graph is Lovász's *cut distance* (see [19, Chapter 8] and Section 3 of this paper). And, last but not least, for the mismatch norm μ_{iso} defined to be 0 if the mismatch graph has no edges and 1 otherwise, $\mathrm{dist}_{\mathrm{iso}}(G, H)$ is 0 if G and H are isomorphic, and 1 otherwise, so computing the distance between two graphs amounts to deciding if they are isomorphic.

Mathematically, the framework of mismatch norms and the associated distances is best described in terms of the adjacency matrices of the graphs; the adjacency matrix A_{G-H} of the mismatch graph (viewed as a matrix with entries $0, +1, -1$ displaying the signs of the

[1] A general definition that also applies to graphs of distinct order can be found in Section 3, but for the hardness results we prove in this paper we can safely restrict our attention to graphs of the same order; this only makes the results stronger.

edges) is just the difference $A_G - A_H$. Then mismatch norms essentially are just matrix norms applied to to A_{G-H}. It turns out that "global norms" such as edit distance and direct generalizations correspond to entrywise matrix norms (obtained by applying a vector norm to the flattened matrix), and "local norms" such as μ_{deg} correspond to operator matrix norms (see Section 3). Cut distance corresponds to the cut norm of matrices. Instead of adjacency matrices, we can also consider the Laplacian matrices, exploiting that $L_{G-H} = L_G - L_H$, and obtain another interesting family of graph distances.

For every mismatch norm μ, we are interested in the problem $\textsc{Dist}_\mu$ of computing $\text{dist}_\mu(G, H)$ for two given graphs G, H. Note that this is a minimization problem where the feasible solutions are the bijective mappings between the vertex sets of the two graphs. It turns out that the problem is hard for most mismatch norms μ, in particular almost all the natural choices discussed above. The single exception is μ_{iso} related to the graph isomorphism problem. Furthermore, the hardness results usually hold even if the input graphs are restricted to be very simple, for example trees or bounded degree graphs. For edit distance and the related distance based on entrywise matrix norms this was already known [3, 13]. Our focus in this paper is on operator norms. We prove a number of hardness results for different graph classes. One of the strongest ones is the following (see Theorem 19). Here $\textsc{Dist}_p$ denotes the distance measure derived from the ℓ_p-operator norm.

▶ **Theorem 1.** *For $1 \leq p \leq \infty$ there is a constant $c > 1$ such that, unless* $\mathsf{P} = \mathsf{NP}$*, $\textsc{Dist}_p$ has no factor-c approximation algorithm even if the input graphs are restricted to be trees of bounded degree.*

For details and additional results, we refer to Section 4 and 5.

Initially we aimed for a general hardness result that applies to all mismatch norms satisfying some additional natural conditions. However, we found that the hardness proofs, while following the same general strategies, usually have some intricacies exploiting special properties of the specific norms. Furthermore, for cut distance, none of these strategies seemed to work. Nevertheless, we were able to give a hardness proof for cut distance (Theorem 23) that is simply based on the hardness of computing the cut norm of the matrix [1]. This is remarkable in so far as usually the hard part of computing the distance is to find an optimal alignment π, whereas computing $\mu(G^\pi - H)$ is usually easy. For cut norm, it is even hard to compute $\mu(G^\pi - H)$ for a fixed alignment π.

Related Work

Graph similarity has mostly been studied in specific application areas, most importantly computer vision, data mining, and machine learning (see the references above). Of course not all similarity measures are based on mismatch. For example, metrics derived from vector embedding or graph kernels in machine learning (see [17]) provide a completely different approach (see [12] for a broader discussion). Of interest compared to our work (specifically for the ℓ_2-operator norm a.k.a. spectral norm) is the spectral approach proposed by Kolla, Koutis, Madan, and Sinop [16]. Intuitively, instead of the "difference" of two graphs that is described by our mismatch graphs, their approach is based on taking a "quotient".

The complexity of similarity, or "approximate graph isomorphism", or "robust graph isomorphism" has been studied in [2, 3, 13, 15, 16, 18, 27], mostly based on graph edit distance and small variations. Operator norms have not been considered in this context.

2 Preliminaries

We denote the class of real numbers by $\mathbb{R}$ and the nonnegative and positive reals by $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$, respectively. By $\mathbb{N}, \mathbb{N}_{>0}$ we denote the sets of nonnegative resp. positive integers. For every $n \in \mathbb{N}_{>0}$ we let $[n] := \{1, \ldots, n\}$.

We will consider matrices with real entries and with rows and columns indexed by arbitrary finite sets. Formally, for finite sets V, W, a $V \times W$ matrix is a function $A\colon V \times W \to \mathbb{R}$. A standard $m \times n$-matrix is just an $[m] \times [n]$-matrix. We denote the set of all $V \times W$ matrices by $\mathbb{R}^{V \times W}$, and we denote the entries of a matrix A by A_{vw}.

For a square matrix $A \in \mathbb{R}^{V \times V}$ and injective mapping $\pi : V \to W$, we let A^π be the $V^\pi \times V^\pi$-matrix with entries $A_{v^\pi w^\pi} := A_{vw}$. Note that we apply the mapping π from the right and denote the image of v under π by v^π. If $\rho : W \to X$ is another mapping, we denote the composition of π and ρ by $\pi\rho$. We typically use this notation for mappings between matrices and between graphs.

We use a standard graph theoretic notation. We denote the vertex set of a graph G by $V(G)$ and the edge set by $E(G)$. Graphs are finite, simple, and undirected, that is, $E(G) \subseteq \binom{V(G)}{2}$. We denote edges by vw instead of $\{v, w\}$. The *order* of a graph is $|G| := |V(G)|$. The *adjacency matrix* $A_G \in \{0,1\}^{V(G) \times V(G)}$ is defined in the usual way. We denote the class of all graphs by $\mathcal{G}$.

Let G be a graph with vertex set $V := V(G)$. For a mapping $\pi : V \to W$ we let G^π be the graph with vertex set $\{v^\pi \mid v \in V\}$ and edge set $\{v^\pi w^\pi \mid vw \in E(G) \text{ with } v^\pi \neq w^\pi\}$. Then $A_{G^\pi} = A_G^\pi$ if π is injective.

For graphs G, H, we denote the set of all injective mappings $\pi : V(G) \to V(H)$ by $\mathrm{Inj}(G, H)$. Graphs G and H are *isomorphic* (we write $G \cong H$) if there is an $\pi \in \mathrm{Inj}(G, H)$ such that $G^\pi = H$. We think of the mappings in $\mathrm{Inj}(G, H)$, in particular if $|G| = |H|$ and they are bijective, as *alignments* between the graphs. Intuitively, to measure the distance between two graphs, we will align them in an optimal way to minimize the mismatch.

3 Graph Metrics Based on Mismatches

A *graph metric* is a function $\delta : \mathcal{G} \times \mathcal{G} \to \mathbb{R}_{\geq 0}$ such that
(GM0) $\delta(G, H) = \delta(G', H')$ for all G, G', H, H' such that $G \cong G'$ and $H \cong H'$;
(GM1) $\delta(G, G) = 0$ for all G;
(GM2) $\delta(G, H) = \delta(H, G)$ for all G, H;
(GM3) $\delta(F, H) \leq \delta(F, G) + \delta(G, H)$ for all F, G, H.
Note that we do not require $\delta(G, H) > 0$ for all $G \neq H$, not even for $G \not\cong H$, so strictly speaking this is just a *pseudometric*. We are interested in the complexity of the following problem:

DIST$_\delta$

Instance: Graphs G, H, $p, q \in \mathbb{N}_{>0}$
Problem: Decide if $\delta(G, H) \geq \frac{p}{q}$

A *signed graph* is a weighted graph with edge weights $-1, +1$, and for every edge e of a signed graph we denote its *sign* by $\mathrm{sg}(e)$. For a signed graph Δ, we let $E_+(\Delta) := \{e \in E(\Delta) \mid \mathrm{sg}(e) = +1\}$ and $E_-(\Delta) := \{e \in E(\Delta) \mid \mathrm{sg}(e) = -1\}$. *Isomorphisms* of signed graphs must preserve signs. A signed graph Δ is a subgraph of a signed graph Γ (we write $\Delta \subseteq \Gamma$) if $V(\Delta) \subseteq V(\Gamma)$, $E_+(\Delta) \subseteq E_+(\Gamma)$, and $E_-(\Delta) \subseteq E_-(\Gamma)$. We denote the class of all signed graphs by $\mathcal{S}$.

For every $\Delta \in \mathcal{S}$, we let $-\Delta$ be the signed graph obtained from Δ by flipping the signs of all edges. We define the *sum* of $\Delta, \Gamma \in \mathcal{S}$ to be the signed graph $\Delta+\Gamma$ with vertex set $V(\Delta+\Gamma) = V(\Delta) \cup V(\Gamma)$ and signed edge sets $E_+(\Delta+\Gamma) = \big(E_+(\Delta) \cup E_+(\Gamma)\big) \setminus \big(E_-(\Delta) \cup E_-(\Gamma)\big)$ and $E_-(\Delta+\Gamma) = \big(E_-(\Delta) \cup E_-(\Gamma)\big) \setminus \big(E_+(\Delta) \cup E_+(\Gamma)\big)$. The adjacency matrix A_Δ of a signed graph Δ displays the signs of the edges, so $A_\Delta \in \{0, 1, -1\}^{V(\Delta)\times V(\Delta)}$ with $(A_\Delta)_{vw} = \mathrm{sg}(vw)$ if $vw \in E(\Delta)$ and $(A_\Delta)_{vw} = 0$ otherwise. Note that $A_{-\Delta} = -A_\Delta$ for all $\Delta \in \mathcal{S}$ and $A_{\Delta+\Gamma} = A_\Delta + A_\Gamma$ for all $\Delta, \Gamma \in \mathcal{S}$ with $V(\Delta) = V(\Gamma)$ and $E_+(\Delta) \cap E_+(\Gamma) = \emptyset$ and $E_-(\Delta) \cap E_-(\Gamma) = \emptyset$.

The *mismatch graph* of two graphs G, H is the signed graph $G - H$ with vertex set $V(G - H) \coloneqq V(G) \cup V(H)$ and signed edge set $E_+(G - H) \coloneqq E(G) \setminus E(H)$, $E_-(G - H) \coloneqq E(H) \setminus E(G)$. Note that if $V(G) = V(H)$ then for the adjacency matrices we have $A_{G-H} = A_G - A_H$. Observe that every signed graph Δ is the mismatch graph of the graphs $\Delta_+ \coloneqq (V(\Delta), E_+(\Delta))$ and $\Delta_- \coloneqq (V(\Delta), E_-(\Delta))$.

A *mismatch norm* is a function $\mu : \mathcal{S} \to \mathbb{R}_{\geq 0}$ satisfying the following conditions:

(MN0) $\mu(\Delta) = \mu(\Gamma)$ for all $\Delta, \Gamma \in \mathcal{S}$ such that $\Delta \cong \Gamma$;

(MN1) $\mu(\Delta) = 0$ for all $\Delta \in \mathcal{S}$ with $E(\Delta) = \emptyset$;

(MN2) $\mu(\Delta) = \mu(-\Delta)$ for all Δ;

(MN3) $\mu(\Delta+\Gamma) \leq \mu(\Delta)+\mu(\Gamma)$ for all $\Delta, \Gamma \in \mathcal{S}$ with $V(\Delta) = V(\Gamma)$ and $E_+(\Delta) \cap E_+(\Gamma) = \emptyset$ and $E_-(\Delta) \cap E_-(\Gamma) = \emptyset$.

(MN4) $\mu(\Delta) = \mu(\Gamma)$ for all $\Delta, \Gamma \in \mathcal{S}$ with $E_+(\Delta) = E_+(\Gamma)$ and $E_-(\Delta) = E_-(\Gamma)$;

For every mismatch norm μ, we define $\mathrm{dist}_\mu : \mathcal{G} \times \mathcal{G} \to \mathbb{R}_{\geq 0}$ by

$$\mathrm{dist}_\mu(G, H) \coloneqq \begin{cases} \min_{\pi \in \mathrm{Inj}(G,H)} \mu(G^\pi - H) & \text{if } |G| \leq |H|, \\ \min_{\pi \in \mathrm{Inj}(H,G)} \mu(G - H^\pi) & \text{if } |H| < |G|. \end{cases}$$

We write $\textsc{Dist}_\mu$ instead of $\textsc{Dist}_{\mathrm{dist}_\mu}$ to denote the algorithmic problem of computing dist_μ for two graphs G, H.

▶ **Lemma 2.** *For every mismatch norm μ the function* dist_μ *is a graph metric.*

Proof. Conditions (GM0), (GM1), (GM2) follow from (MN0), (MN1), (MN2), respectively. To prove (GM3), let F, G, H be graphs. Padding the graphs with isolated vertices, by (MM4) we may assume that $|F| = |G| = |H|$. By (MN0) we may further assume that $V(F) = V(G) = V(H) \coloneqq V$. Choose $\pi, \rho \in \mathrm{Inj}(V, V)$ such that $\mathrm{dist}_\mu(F, G) = \mu(F^\pi - G)$ and $\mathrm{dist}_\mu(G, H) = \mu(G^\rho - H)$.

Then by (MN0) we have

$$\mu(F^{\pi\rho} - G^\rho) = \mu\big((F^\pi - G)^\rho\big) = \mu(F^\pi - G) = \mathrm{dist}_\mu(F, G).$$

Now consider the two mismatch graphs $\Delta \coloneqq F^{\pi\rho} - G^\rho$ and $\Gamma \coloneqq G^\rho - H$. We have $E_+(\Delta) = E(F^{\pi\rho}) \setminus E(G^\rho)$ and $E_+(\Gamma) = E(G^\rho) \setminus E(H)$. Thus $E_+(\Delta) \cap E(G^\rho) = \emptyset$ and $E_+(\Gamma) \subseteq E(G^\rho)$, which implies $E_+(\Delta) \cap E_+(\Gamma) = \emptyset$. Similarly, $E_-(\Delta) \cap E_-(\Gamma) = \emptyset$. Moreover, $\Delta + \Gamma = F^{\pi\rho} - H$, because

$$A_{\Delta+\Gamma} = A_\Delta + A_\Gamma = (A_F^{\pi\rho} - A_G^\rho) + (A_G^\rho - A_H) = A_F^{\pi\rho} - A_H = A_{F^{\pi\rho}-H}.$$

Thus by (MN3),

$$\mathrm{dist}_\mu(F, H) \leq \mu(F^{\pi\rho} - H) = \mu(\Delta + \Gamma) \leq \mu(\Delta) + \mu(\Gamma) = \mathrm{dist}_\mu(F, G) + \mathrm{dist}_\mu(G, H).$$

This proves (GM3). ◀

▶ Remark 3. None of the five conditions (MN0)–(MN4) on a mismatch norm μ can be dropped if we want to guarantee that dist_μ is a graph metric, but of course we could replace them by other conditions. While (MN0)–(MN3) are very natural and directly correspond to conditions (GM0)–(GM3) for graph metrics, condition (MN4) is may be less so. We chose (MN4) as the simplest condition that allows us to compare graphs of different sizes.

Having said this, it is worth noting that (MN0), (MN1) and (MN3) imply (MN4) for graphs Δ, Γ with $|\Delta| = |\Gamma|$. For graphs Δ, Γ with $|\Delta| < |\Gamma|$ they only imply $\mu(\Delta) \geq \mu(\Gamma)$. Thus as long as we only compare graphs of the same order, (MN4) is not needed. In particular, since our hardness results always apply to graphs of the same order, (MN4) is inessential for the rest of the paper.

However, it is possible to replace (MN4) by other natural conditions. For example, Lovász's metric based on a scaled cut norm [20] does not satisfy (MN4) and instead uses a blowup of graphs to a common size to compare graphs of different sizes.

Let us now consider a few examples of mismatch norms.

▶ **Example 4** (Isomorphism). The mapping $\iota := \mathcal{S} \to \mathbb{R}_{\geq 0}$ defined by $\iota(\Delta) := 0$ if $E(\Delta) = \emptyset$ and $\iota(\Delta) := 1$ otherwise is a mismatch norm. Under the metric dist_ι, all nonisomorphic graphs have distance 1 (and isomorphic graphs have distance 0, as they have under all graph metrics).

▶ **Example 5** (Matrix Norms). Recall that a matrix (pseudo) norm $\|\cdot\|$ associates with every matrix A (say, with real entries) an nonnegative real $\|A\|$ in such a way that $\|N\| = 0$ for matrices N with only 0-entries, $\|aA\| = |a| \cdot \|A\|$ for all matrices A and reals $a \in \mathbb{R}$, and $\|A + B\| \leq \|A\| + \|B\|$ for all matrices A, B of the same dimensions.

Actually, we are only interested in square matrices here. We call a matrix norm $\|\cdot\|$ *permutation invariant* if for all $A \in \mathbb{R}^{V\times V}$ and all bijective $\pi : V \to V$ we have $\|A\| = \|A^\pi\|$. It is easy to see that for every permutation invariant matrix norm $\|\cdot\|$, the mapping $\mu_{\|\cdot\|} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ defined by $\mu_{\|\cdot\|}(\Delta) := \|A_\Delta\|$ satisfies (MN0)–(MN3).

We call a permutation invariant matrix norm $\|\cdot\|$ *paddable* if it is invariant under extending matrices by zero entries, that is, $\|A\| = \|A'\|$ for all $A \in \mathbb{R}^{V\times V}$, $A' \in \mathbb{R}^{V'\times V'}$ such that $V' \supseteq V$, $A_{vw} = A'_{vw}$ for all $v, w \in V$, and $A'_{vw} = 0$ if $v \in V' \setminus V$ or $w \in V' \setminus V$. A paddable matrix norm also satisfies (MN4).

The following common matrix norms are paddable (and thus by definition also invariant). Let $1 \leq p \leq \infty$ and $A \in \mathbb{R}^{V\times V}$.

1. *Entrywise p-norm:* $\|A\|_{(p)} := \Big(\sum_{v,w\in V} |A_{vw}|^p\Big)^{\frac{1}{p}}$. The best-known special case is the *Frobenius norm* $\|\cdot\|_F := \|\cdot\|_{(2)}$.
2. *ℓ_p-operator norm:* $\|A\|_p := \sup_{\boldsymbol{x}\in\mathbb{R}^V\setminus\{\boldsymbol{0}\}} \frac{\|A\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}$, where the vector p-norm is defined by $\|\boldsymbol{a}\|_p := \big(\sum_{v\in V} a_v^p\big)^{\frac{1}{p}}$. In particular, $\|A\|_2$ is known as the *spectral norm.*
3. *Absolute ℓ_p-operator norm:* $\|A\|_{|p|} := \|\operatorname{abs}(A)\|_p$, where abs takes entrywise absolute values. For the mismatch norm, taking entrywise absolute values means that we ignore the signs of the edges.
4. *Cut norm:* $\|A\|_\square := \max_{S,T\subseteq V} \Big|\sum_{v\in S,w\in T} A_{vw}\Big|$.

▶ **Example 6** (Laplacian Matrices). Recall that the *Laplacian matrix* of a weighted graph G with vertex set $V := V(G)$ is the $V \times V$ matrix L_G with off-diagonal entries $(L_G)_{vw}$ being the negative weight of the edge $vw \in E(G)$ if there is such an edge and 0 otherwise and diagonal entries $(L_G)_{vv}$ being the sum of the weights of all edges incident with v. For an unweighted graph we have $L_G = D_G - A_G$, where D_G is the diagonal matrix with the vertex degrees as diagonal entries.

Observe that for a signed graph $\Delta = G - H$ we have $L_\Delta = L_G - L_H$.

It is easy to see that for every paddable matrix norm $\|\cdot\|$, the function $\mu^L_{\|\cdot\|} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ defined by $\mu^L_{\|\cdot\|}(\Delta) := \|L_\Delta\|$ is a mismatch norm.

Clearly, $\textsc{Dist}_\mu$ is not a hard problem for every mismatch norm. For example, $\textsc{Dist}_\nu$ is trivial for the trivial mismatch norm ν defined by $\nu(\Delta) := 0$ for all Δ, and $\textsc{Dist}_\iota$ for ι from Example 4 is equivalent to the graph isomorphism problem and hence in quasipolynomial time [4].

However, for most natural matrix norms the associated graph distance problem is NP-hard. In particular, for every $p \in \mathbb{R}_{>0}$, this holds for the metric $\mathrm{dist}_{(p)}$ based on the entrywise p-norm $\|\cdot\|_{(p)}$.

▶ **Theorem 7** ([13]). *For $p \in \mathbb{R}_{>0}$, $\textsc{Dist}_{(p)}$ is NP-hard even if restricted to trees or bounded-degree graphs.*

The proof in [13] is only given for the Frobenius norm, that is, $\textsc{Dist}_{(2)}$, but it actually applies to all p. In the rest of the paper, we study the complexity of $\textsc{Dist}_p$, $\textsc{Dist}_{|p|}$ and $\textsc{Dist}_\square$.

4 Complexity for Operator Norms

In this section we investigate the complexity of $\textsc{Dist}_p$ and $\textsc{Dist}_{|p|}$ for $1 \leq p \leq \infty$. However, as $\mu_{|p|}$ would only be a special case within the upcoming proofs, we omit to mention it explicitly. We also omit to specify the possible values for p.

Given graphs G and H, an alignment from G to H, and a node $v \in V(G)$ we refer to the nodes whose adjacency is not preserved by π as the *π-mismatches of v*. If π is clear from the context we might omit it. We call $G^\pi - H$ the *mismatch graph of π*.

For all ℓ_p-operator norms the value of $\mu_p(G^\pi - H)$ strongly depends on the maximum degree in the mismatch graph of π. We capture this property with the following definition.

▶ **Definition 8.** *Let G, H be graphs of the same order and π an alignment from G to H. The* π-mismatch count (π-MC) *of a node $v \in V(G)$ is defined as:*

$$\mathrm{MC}(v, \pi) := |\{w \in V(G) \mid w \textit{ is a } \pi\textit{-mismatch of } v\}|\,.$$

We use MC for nodes in H analogously. The *maximum mismatch count* (π-MMC) of π is defined as:

$$\mathrm{MMC}(\pi) := \max_{v \in V(G)} \mathrm{MC}(v, \pi).$$

Again we might drop the π if it is clear from the context. Note that the MMC corresponds to the maximum degree in the mismatch graph and that we use a slightly abbreviated notation in which we assume the graphs are given by the alignment.

The ℓ_1-operator norm and the ℓ_∞-operator norm measure exactly the maximum mismatch count. Due to the relation between the ℓ_p-operator norms we can derive an upper bound for μ_p. The proof of Lemma 9 can be found in the full version [11].

▶ **Lemma 9.** *Let G, H be graphs of the same order and π an alignment from G to H. Then*

$$\mu_1(G^\pi - H) = \mu_\infty(G^\pi - H) = \mathrm{MMC}(\pi),$$
$$\mu_p(G^\pi - H) \leq \mathrm{MMC}(\pi).$$

Next, we observe that μ_p is fully determined by the connected component of the mismatch graph with the highest mismatch norm. The proof of Lemma 10 can be found in the full version [11].

▶ **Lemma 10.** *Let G, H be graphs of the same order, π an alignment from G to H, and $\mathcal{C}$ the set of all connected components in $G^\pi - H$. Then*

$$\mu_p(G^\pi - H) = \max_{C \in \mathcal{C}} \mu_p(C).$$

For the sake of readability, we introduce a function as abbreviation for our upcoming bounds.

▶ **Definition 11.** *For all $1 \leq p \leq \infty$ we define the function* bound_p *as follows:*

$$\mathrm{bound}_p(c) \coloneqq \max\left(c^{1/p}, c^{1-1/p}\right).$$

In particular, $\mathrm{bound}_2(c) = \sqrt{c}$. Now we derive our lower bound. The proof of Lemma 12 can be found in the full version [11].

▶ **Lemma 12.** *Let G, H be graphs of the same order, π an alignment from G to H, and $v \in V(G)$.*
If v has at least c mismatches, then

$$\mu_p(G^\pi - H) \geq \mathrm{bound}_p(c).$$

If $G^\pi - H$ is a star, then

$$\mu_p(G^\pi - H) = \mathrm{bound}_p(\mathrm{MMC}(\pi)).$$

While μ_1 and μ_∞ simply measure the MMC, μ_2 also considers the connectedness of the mismatches around the node with the highest MC. As Lemma 9 and Lemma 12 tell us, μ_2 ranges between the $\sqrt{\mathrm{MMC}}$ and the MMC. In fact, the lower bound is tight for stars and the upper bound is tight for regular graphs. This is intuitive as these are the extreme cases in which no mismatch can be removed/added without decreasing/increasing the MMC, respectively. Other ℓ_p-operator norms interpolate between μ_2 and μ_1 / μ_∞ in terms of how much they value the connectedness within the mismatch component of the node with the highest MC.

The lower bound for $\mu_p(G^\pi - H)$ gives us lower bound for $\mathrm{dist}_p(G, H)$.

▶ **Lemma 13.** *Let G, H be graphs of the same order and π an alignment from G to H. If all alignments from G to H have a node with at least c mismatches, then*

$$\mathrm{dist}_p(G, H) \geq \mathrm{bound}_p(c).$$

Proof. This follows directly from the first claim of Lemma 12. ◀

The following upper bound might seem to have very restrictive conditions but is actually used in several hardness proofs.

▶ **Lemma 14.** *Let G, H be graphs of the same order and π an alignment from G to H. If the mismatch graph of π consists only of stars, then*

$$\mathrm{dist}_p(G, H) \ \leq \mu_p(G^\pi - H) \ = \mathrm{bound}_p(\mathrm{MMC}(\pi)).$$

Proof. This follows directly from Lemma 10 and the second claim of Lemma 12. ◀

The last tool we need to prove the hardness is that we can distinguish two alignments by their MMC as long as the mismatch graph of the alignment with the lower MMC consists only of stars. The proof of Lemma 15 can be found in the full version [11].

▶ **Lemma 15.** *For all $c, d \in \mathbb{N}$ with $c < d$ it holds that* $\mathrm{bound}_p(c) < \mathrm{bound}_p(d)$.

The graph isomorphism problem becomes solvable in polynomial time if restricted to graphs of bounded degree [21]. In contrast to this, $\textsc{Dist}_F$ is NP-hard even under this restriction [13]. We show that the same applies to $\textsc{Dist}_p$.

The reduction in the hardness proof works for any mismatch norm which can, said intuitively, distinguish the mismatch norm of the 1-regular graph of order n from any other n-nodes mismatch graph in which every node has at least one -1 edge but at least one node has an additional -1 edge and $+1$ edge. In particular, the construction also works for $\textsc{Dist}_{(p)}$. However, it does not work for the cut-distance, for which we independently prove the hardness in Section 6.

▶ **Theorem 16.** $\textsc{Dist}_p$ *and* $\textsc{Dist}_{|p|}$ *are* NP*-hard for* $1 \leq p \leq \infty$ *even if both graphs have bounded degree.*

Proof. The proof is done by reduction from the NP-hard Hamiltonian Cycle problem in 3-regular graphs ($\textsc{Ham-Cycle}$) [10]. Given a 3-regular graph G of order n as an instance of $\textsc{Ham-Cycle}$, the reduction uses the n-nodes cycle C_n and G as inputs for $\textsc{Dist}_p$. We claim G has a Hamiltonian cycle if and only if $\mathrm{dist}_p(C_n, G) \leq \mathrm{bound}_p(1)$.

Assume that G has a Hamiltonian cycle. Then there exists a bijection $\pi : V(C_n) \to V(G)$ that aligns the cycle C_n perfectly with the Hamiltonian cycle in G. Each node in G has three neighbors, two of which are matched correctly by π as they are part of the Hamiltonian cycle. Hence, each node has a π-mismatch count of 1 and obviously the MMC of π is 1 as well. According to Lemma 14, we get $\mathrm{dist}_p(C_n, G) \leq \mathrm{bound}_p(1)$.

Conversely, assume that G has no Hamiltonian cycle. Then, for any alignment π' from C_n to G, there exists at least one edge vw in C_n that is not mapped to an edge in G. Hence, only one of the three edges incident to $\pi'(v)$ in G can be matched correctly. In total, v has at least one mismatch from C_n to G and two mismatches from G to C_n, which implies $\mathrm{MC}(v, \pi') \geq 3$. Using Lemma 13, we get $\mathrm{dist}_p(C_n, G) \geq \mathrm{bound}_p(3)$. And then $\mathrm{dist}_p(C_n, G) > \mathrm{bound}_p(1)$ according to Lemma 15. ◀

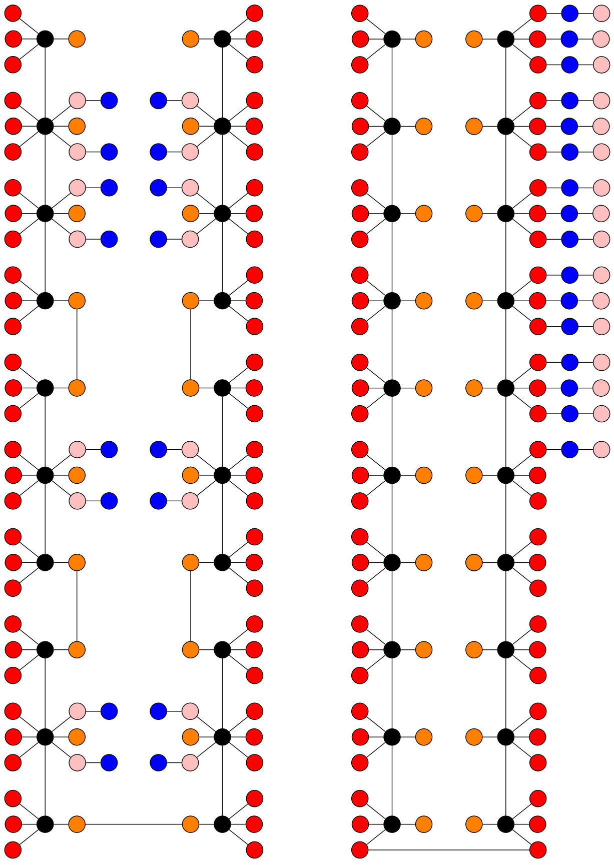
Next, we modify the construction to get an even stronger NP-hardness result. The proof can be found in the full version [11].

▶ **Theorem 17.** $\textsc{Dist}_p$ *and* $\textsc{Dist}_{|p|}$ *are* NP*-hard for* $1 \leq p \leq \infty$ *even if restricted to a path and a graph of maximum degree* 3*.*

Similar to bounded degree input graphs, restricting graph isomorphism to trees allows it to be solved in polynomial time [24] but $\textsc{Dist}_F$ is NP-hard for trees [13]. We show that $\textsc{Dist}_p$ remains NP-hard for trees even when applying the bounded degree restriction simultaneously.

▶ **Theorem 18.** $\textsc{Dist}_p$ *and* $\textsc{Dist}_{|p|}$ *are* NP*-hard for* $1 \leq p \leq \infty$ *even if restricted to bounded-degree trees.*

Proof. The proof is done by reduction from the NP-hard $\textsc{Three-Partition}$ problem [10] that is defined as follows. Given the integers A and $a_1, \ldots, a_{3m}$ in unary representation, such that $\sum_{i=1}^{3m} a_i = mA$ and $A/4 < a_i < A/2$ for $1 \leq i \leq 3m$, decide whether there exists a



(a) T_1.

(b) T_2.

Figure 1 Example of the construction in the proof of Theorem 18 where m=2, A=10, a_1=a_2=4 and a_3=a_4=a_5=a_6=3. Best viewed in color.

partition of $a_1, \ldots, a_{3m}$ into m groups of size 3 such that the elements in each group sum up to exactly A. For technical reasons, we restrict the reduction to $A \geq 8$. However, the ignored cases are trivial. Precisely, for $A \in \{6, 7\}$ the answer is always YES and for $A \in [5]$ there exist no valid instances.

Given an instance of Three-Partition with $A \geq 8$, we construct two trees T_1 and T_2 such that the given instance has answer YES, if and only if $\text{dist}_p(T_1, T_2) \leq \text{bound}_p(2)$. Figure 1 shows an example of T_1 and T_2 for $m = 2$ and $A = 10$. For illustrative reasons we assign each node a color during the construction. The colors are used in the example and we will refer to certain nodes by their color later in the proof. However, they do not restrict the possible alignments in any way.

Initialize T_1 as the disjoint union of paths $p_1^1, \ldots, p_{3m}^1$ such that p_i^1 has a_i black nodes; initialize T_2 as the disjoint union of paths $p_1^2, \ldots, p_m^2$ consisting of A black nodes each. In the following we refer to one endpoint of p_i^k as $e_1(p_i^k)$ and the other endpoint as $e_2(p_i^k)$. We attach three new red leaves and one new orange leaf to each black node in both T_1 and T_2. Next we modify the graphs into trees by connecting the paths. For $1 \leq i \leq 3m - 1$ we add an edge between the orange leaf adjacent to $e_1(p_i^1)$ and the orange leaf adjacent to $e_2(p_{i+1}^1)$. For $1 \leq i \leq m - 1$ we add an edge between one of the red leaves adjacent to $e_1(p_i^2)$ and one of the red leaves adjacent to $e_2(p_{i+1}^2)$.

Next we attach two new pink leaves to each inner (non-endpoint) path node in T_1 and attach a new blue leaf to each pink node. Then we add the same number of blue nodes to T_2 and connect each blue node to one of the red nodes with degree 1. Finally, we attach a new pink leaf to each blue node. Note that both T_1 and T_2 are trees with bounded degree. Precisely, the highest degree in T_1 is 8 and 6 in T_2 independent of the problem instance.

Intuitively, the construction ensures that every inner path node has already 2 mismatches just because of the degree difference. If there is no partition, at least one path in T_1 cannot be mapped contiguously into a path in T_2 which raises the MC of some inner path node to at least 3. Simultaneously, the construction ensures that there is an alignment for which the mismatch graph consists only of stars with maximum degree 2 if there is an alignment.

The formal continuation of this proof can be found in the full version [11]. ◀

5 Approximability for Operator Norms

In this section we investigate the approximability of $\textsc{Dist}_p$ and $\textsc{Dist}_{|p|}$ for $1 \leq p \leq \infty$. Again, we omit specifying the possible values for p and mentioning $\textsc{Dist}_{|p|}$ explicitly as the proofs work the same for it. We consider the following two possibilities to measure the error of an approximation algorithm for a minimization problem. An algorithm $\mathcal{A}$ has *multiplicative error* $\alpha > 1$, if for any instance $\mathcal{I}$ of the problem with an optimum $\mathrm{OPT}(\mathcal{I})$, $\mathcal{A}$ outputs a solution with value $\mathcal{A}(\mathcal{I})$ such that $\mathrm{OPT}(\mathcal{I}) \leq \mathcal{A}(\mathcal{I}) \leq \alpha\mathrm{OPT}(\mathcal{I})$. In this case we call $\mathcal{A}$ an *α-approximation algorithm*. An algorithm $\mathcal{B}$ has *additive error* $\varepsilon > 0$, if $\mathrm{OPT}(\mathcal{I}) \leq \mathcal{B}(\mathcal{I}) \leq \mathrm{OPT}(\mathcal{I}) + \varepsilon$ for any instance $\mathcal{I}$.

Approximating $\textsc{Dist}_p$ with multiplicative error is at least as hard as the graph isomorphism problem (GI) since such an approximation algorithm $\mathcal{A}$ could be used to decide GI considering that $\mathcal{A}(G, H) = 0$ if and only if G is isomorphic to H.

Furthermore, we can deduce thresholds under which the α-approximation is NP-hard using the gap between the decision values of the reduction in each hardness proof from Section 4.

▶ **Theorem 19.** *For $1 \leq p \leq \infty$ and any $\varepsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial time approximation algorithm for $\textsc{Dist}_p$ or $\textsc{Dist}_{|p|}$ with a multiplicative error guarantee of*

1. $\mathrm{bound}_p(3) - \varepsilon$*, even if both input graphs have bounded degree,*
2. $\mathrm{bound}_p(2) - \varepsilon$*, even if one input graph is a path and the other one has bounded degree,*
3. $\frac{\mathrm{bound}_p(3)}{\mathrm{bound}_p(2)} - \varepsilon$*, even if both input graphs are trees with bounded degree.*

Proof. We recall the proof of Theorem 16. If G has a Hamiltonian cycle, then $\mathrm{dist}_p(C_n, G) \leq \mathrm{bound}_p(1) = 1$. Otherwise $\mathrm{dist}_p(C_n, G) \geq \mathrm{bound}_p(3)$. Assume there is a polynomial time approximation algorithm $\mathcal{A}$ with a multiplicative error guarantee of $\mathrm{bound}_p(3) - \varepsilon$ for $\varepsilon > 0$. Then we can distinguish the two cases by checking whether $\mathcal{A}(C_n, G) < \mathrm{bound}_p(3)$ and therefore decide $\textsc{Ham-Cycle}$ on 3-regular graphs in polynomial time. The same argument can be used for the other bounds using the proofs of Theorem 17 and Theorem 18, respectively. ◀

In particular, this implies that there is no polynomial time approximation scheme (PTAS) for $\textsc{Dist}_p$ or $\textsc{Dist}_{|p|}$ under the respective restrictions.

Next we show the additive approximation hardness by scaling up the gap between the two decision values of the reduction in the proof of Theorem 16. For this we replace each edge with a colored gadget and then modify the graph so that an optimal alignment has to be color-preserving. The proof of Theorem 20 can be found in the full version [11].

▶ **Theorem 20.** *For $1 \leq p \leq \infty$ there is no polynomial time approximation algorithm for $\textsc{Dist}_p$ with any constant additive error guarantee unless $\mathsf{P} = \mathsf{NP}$.*

However, the approximation of $\textsc{Dist}_p$ becomes trivial once we restrict the input to graphs of bounded degree, although $\textsc{Dist}_p$ stays NP-hard under this restriction. The proof of Theorem 21 can be found in the full version [11].

▶ **Theorem 21.** *For $1 \leq p \leq \infty$, if one graph has maximum degree d, then there is a polynomial time approximation algorithm for $\textsc{Dist}_p$ and $\textsc{Dist}_{|p|}$ with*

1. *a constant additive error guarantee $2d$,*
2. *a constant multiplicative error guarantee $1 + 2d$.*

6 Complexity for Cut Norm

Finally, we show the hardness for $\textsc{Dist}_\square$ which corresponds to the cut distance $\hat{\delta}_\square$ (see [19, Chapter 8]. For any signed graph G and $V \subseteq V(G)$ the induced subgraph $G[V]$ is the signed graph with vertex set V, $E_+(G[V]) = \{vw \in E_+(G) \mid v, w \in V)\}$, and $E_-(G[V]) = \{vw \in E_-(G) \mid v, w \in V)\}$.

▶ **Lemma 22.** *Let Δ be a signed graph and $W \subseteq V(\Delta)$. Then $\mu_\square(\Delta) \geq \mu_\square(\Delta[W])$.*

Proof. Let $V := V(\Delta)$, $A := A_\Delta$, $B := A_{\Delta[W]}$ and $S', T' := \operatorname{argmax}_{S,T \subseteq W} \left|\sum_{v \in S, w \in T} B_{vw}\right|$. Then

$$\|\Delta[W]\|_\square = \left|\sum_{v \in S', w \in T'} B_{vw}\right| = \left|\sum_{v \in S', w \in T'} A_{vw}\right| \leq \max_{S,T \subseteq V} \left|\sum_{v \in S, w \in T} A_{vw}\right| = \|\Delta\|_\square . \qquad ◀$$

Our hardness proof for $\textsc{Dist}_\square$ is based on the hardness of computing the cut norm. Intuitively, the construction enforces a specific alignment by modifying the nodes degrees.

▶ **Theorem 23.** *The problem $\textsc{Dist}_\square$ is NP-hard.*

Proof. The proof is done by reduction from the NP-hard $\textsc{Max-Cut}$ problem on unweighted graphs [10]. First, we recall how Alan and Naor [1] construct a matrix A for any graph G so that $\|A\|_\square = MAXCUT(G)$. Orient G in an arbitrary manner, let $V(G) = \{v_1, v_2, \ldots, v_n\}$ and $E(G) = \{e_1, e_2, \ldots, e_m\}$. Then A is the $2m \times n$ matrix defined as follows. For each $1 \leq i \leq m$, if e_i is oriented from v_j to v_k, then $A_{2i-1,j} = A_{2i,k} = 1$ and $A_{2i-1,k} = A_{2i,j} = -1$. The rest of the entries in A are all 0.

Next, we observe that the matrix

$$B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

has the property $\|B\|_\square = 2\,\|A\|_\square = 2 \cdot MAXCUT(G)$ since the cut norm is invariant under transposition and the two submatrices A, A^T have no common rows or columns in B.

We interpret B as the adjacency matrix of a signed graph Δ' and construct the two unsigned graphs $F' := (V(\Delta), E_+(\Delta))$, $H' := (V(\Delta), E_-(\Delta))$. Then $\mu_\square(F' - H') = \mu_\square(\Delta') = \|B\|_\square = 2 \cdot MAXCUT(G)$. Next, we modify F', H' into the graphs F, H by adding $i \cdot \left(\left\lceil \frac{n^2}{4} \right\rceil + n\right)$ leaves to node v_i for $1 \leq i \leq n$. The reduction follows from the claim that $\operatorname{dist}_\square(F, H) = 2 \cdot MAXCUT(G)$, which we prove in the following.

Let π be an alignment that maps v_i to v_i for $1 \leq i \leq n$ and each leaf in F to a leaf in H so that its adjacency is preserved; let Δ be the mismatch graph of π. Then $E(\Delta) = E(\Delta')$ and therefore $\mu_\square(F^\pi - H) = 2 \cdot MAXCUT(G)$. We conclude $\operatorname{dist}_\square(F, H) \leq 2 \cdot MAXCUT(G)$.

It remains to show that no alignment can lead to a lower mismatch norm. First, let σ be any alignment that maps v_i to v_i for $1 \leq i \leq n$; let Λ be the mismatch graph of σ. Then $\Lambda[\{v_1, \ldots, v_n\}] = \Delta'$ and we get $\mu_\square(\Lambda) \geq 2 \cdot \mathit{MAXCUT}(G)$ from Lemma 22.

Conversely, let ρ be any alignment that maps v_i to v_j for $i \neq j$ and $i, j \leq n$; let Γ be the mismatch graph of ρ. The number of leaves adjacent to v_i in F and to v_j in H differs at least by $\left\lceil \frac{n^2}{4} \right\rceil + n$. Without restriction we can assume there are at least $l := \left\lceil \frac{n^2}{4} \right\rceil$ leaves $w_1, \ldots, w_l$ adjacent to v_i in F that are mismatched by ρ. Let $S := \{v_i^\rho, w_1^\rho, \ldots, w_l^\rho\}$. Then $\Gamma[S]$ has exactly l edges all of which have the same sign. It is easy to see that $\mu_\square(\Gamma[S]) = 2l$ which implies $\mu_\square(\Gamma) \geq 2l$ according to Lemma 22. We chose l so that $2l \geq \frac{n^2}{2} \geq 2 \cdot \mathit{MAXCUT}(G)$. After considering all alignments, we get $\operatorname{dist}_\square(F, G) \geq 2 \cdot \mathit{MAXCUT}(G)$. This proves our claim and therefore concludes the reduction. ◀

7 Concluding Remarks

We study the computational complexity of a class of graph metrics based on mismatch norms, or equivalently, matrix norms applied to the difference of the adjacency matrices of the input graphs under an optimal alignment between the vertex sets. We find that computing the distance between graphs under these metrics (at least for the standard, natural matrix norms) is NP-hard, often already on simple input graphs such as trees. This was essentially known for entrywise matrix norms. We prove it for operator norms and also for the cut norm.

We leave it open to find (natural) general conditions on a mismatch norm such that the corresponding distance problem becomes hard. Maybe more importantly, we leave it open to find meaningful tractable relaxations of the distance measures.

Measuring similarity via mismatch norms is only one approach. There are several other, fundamentally different ways to measure similarity. We are convinced that graph similarity deserves a systematic and general theory that compares the different approaches and studies their semantic as well as algorithmic properties. Our paper is one contribution to such a theory.

References

1 Noga Alon and Assaf Naor. Approximating the cut-norm via grothendieck's inequality. *SIAM J. Comput.*, 35(4):787–803, April 2006. doi:10.1137/S0097539704441629.

2 S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming*, 92(1):1–36, 2002.

3 V. Arvind, J. Köbler, S. Kuhnert, and Y. Vasudev. Approximate graph isomorphism. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS 2012)*, volume 7464, pages 100–111, 2012.

4 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. 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 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

5 Amélie Barbe, Marc Sebban, Paulo Gonçalves, Pierre Borgnat, and Rémi Gribonval. Graph diffusion wasserstein distances. In Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part II*, volume 12458 of *Lecture Notes in Computer Science*, pages 577–592. Springer, 2020. doi:10.1007/978-3-030-67661-2_34.

6 H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18:689–694, 1997.

7 D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, May 2004.

8 F. Emmert-Streib, M. Dehmer, and Y. Shi. Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, 346:180–197, 2016.

9 P. Foggia, G. Percannella, and M. Vento. Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(1), 2014.

10 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

11 Timo Gervens and Martin Grohe. Graph similarity based on matrix norms, 2022. doi:10.48550/ARXIV.2207.00090.

12 M. Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In D. Suciu, Y. Tao, and Z. Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2020. doi:10.1145/3375395.3387641.

13 M. Grohe, G. Rattan, and G. J. Woeginger. Graph similarity and approximate isomorphism. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117, pages 20:1–20:16, 2018.

14 Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Communications of the ACM*, 63(11):128–134, 2020. doi:10.1145/3372123.

15 P. Keldenich. Random robust graph isomorphism. Master's thesis, Department of Computer Science, RWTH Aachen University, 2015.

16 A. Kolla, I. Koutis, V. Madan, and A. K. Sinop. Spectrally Robust Graph Isomorphism. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 84:1–84:13, 2018.

17 N.M. Kriege, F.D Johansson, and C. Morris. A survey on graph kernels. *ArXiv*, arXiv:1903.11835 [cs.LG], 2019.

18 C.-L. Lin. Hardness of approximating graph transformation problem. In D.-T. Du and X.-S. Zhang, editors, *Proceedings of the 5th International Symposium on Algorithms and Computation*, volume 834 of *Lecture Notes in Computer Science*, pages 74–82. Springer, 1994.

19 L. Lovász. *Large Networks and Graph Limits.* American Mathematical Society, 2012.

20 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications.* American Mathematical Society, 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

21 E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

22 K. Makarychev, R. Manokaran, and M. Sviridenko. Maximum quadratic assignment problem: Reduction from maximum label cover and lp-based approximation algorithm. *ACM Transactions on Algorithms*, 10(4):18, 2014.

23 Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. GOT: an optimal transport framework for graph comparison. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13876–13887, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/fdd5b16fc8134339089ef25b3cf0e588-Abstract.html.

24 D. W. Matula. Subtree isomorphism in $o(n^{5/2})$. In B. Alspach, P. Hell, and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2, pages 91–106. Elsevier, 1978.

25 Facundo Mémoli. Gromov-wasserstein distances and the metric approach to object matching. *Found. Comput. Math.*, 11(4):417–487, 2011. doi:10.1007/s10208-011-9093-5.

26 V. Nagarajan and M. Sviridenko. On the maximum quadratic assignment problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 516–524, 2009.

27 R. O'Donnell, J. Wright, C. Wu, and Y. Zhou. Hardness of robust graph isomorphism, lasserre gaps, and asymmetry of random graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1659–1677, 2014.

28 D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

29 A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller. Netlsd: Hearing the shape of a graph. In Y. Guo and F. Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2347–2356, 2018.

30 S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

31 R.J. van Glabbeek. The linear time - branching time spectrum I. The semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.

Approximation Algorithms for Covering Vertices by Long Paths

Mingyang Gong
Department of Computing Science, University of Alberta, Edmonton, Canada

Jing Fan
Department of Computing Science, University of Alberta, Edmonton, Canada
College of Arts and Sciences, Shanghai Polytechnic University, Shanghai, China

Guohui Lin[1]
Department of Computing Science, University of Alberta, Edmonton, Canada

Eiji Miyano
Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, Japan

Abstract

Given a graph, the general problem to cover the maximum number of vertices by a collection of vertex-disjoint long paths seemingly escapes from the literature. A path containing at least k vertices is considered long. When $k \leq 3$, the problem is polynomial time solvable; when k is the total number of vertices, the problem reduces to the Hamiltonian path problem, which is NP-complete. For a fixed $k \geq 4$, the problem is NP-hard and the best known approximation algorithm for the weighted set packing problem implies a k-approximation algorithm. To the best of our knowledge, there is no approximation algorithm directly designed for the general problem; when $k = 4$, the problem admits a 4-approximation algorithm which was presented recently. We propose the first $(0.4394k + O(1))$-approximation algorithm for the general problem and an improved 2-approximation algorithm when $k = 4$. Both algorithms are based on local improvement, and their performance analyses are done via amortization.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Path cover, k-path, local improvement, amortized analysis, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.53

Funding *Jing Fan*: A visiting scholarship from the Shanghai Polytechnic University
Guohui Lin: NSERC Canada
Eiji Miyano: KAKENHI Grants JP21K11755 and JP17K00016 and JST CREST JPMJR1402

1 Introduction

Path Cover (PC) is one of the most well-known NP-hard optimization problems in algorithmic graph theory [11], in which given a simple undirected graph $G = (V, E)$ one wishes to find a minimum collection of vertex-disjoint paths that cover all the vertices, that is, every vertex of V is on one of the paths. It has numerous applications from the real life, such as transportation networks, communication networks and networking security. In particular, it includes the Hamiltonian path problem [11] as a special case, which asks for the existence of a single path covering all the vertices.

The Hamiltonian path problem is NP-complete; therefore, the PC problem cannot be approximated within ratio 2 if P $\neq$ NP. In fact, to the best of our knowledge, there is no $o(|V|)$-approximation algorithm for the PC problem. In the literature, several alternative

[1] Correspondence author.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 53; pp. 53:1–53:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

objective functions have been proposed and studied [3, 1, 18, 2, 19, 4, 12, 5]. For example, Berman and Karpinski [3] tried to maximize the number of edges on the paths in a path cover, which is equal to $|V|$ minus the number of paths, and proposed a 7/6-approximation algorithm. Chen et al. [4, 5] showed that finding a path cover with the minimum total number of order-1 and order-2 paths (where the order of a path is the number of vertices on the path; i.e., singletons and edges) can be done in polynomial time, but it is NP-hard to find a path cover with the minimum number of paths of order at most ℓ when $\ell \geq 3$.

Recently, Kobayashi et al. [15] generalized the problem studied by Chen et al. [4, 5] to assign a weight representing its profit or cost to each order-ℓ path, with the goal of finding a path cover of the maximum weight or the minimum weight, respectively. For instance, when the weight $f(\ell)$ of an order-ℓ path is $f(\ell) = 1$ for any $\ell \leq k$ and $f(\ell) = 0$ for any $\ell \geq k + 1$, where k is a fixed integer, the *minimization* problem reduces to the problem studied by Chen et al. [4, 5]; when $f(\ell)$ is $f(\ell) = 1$ for any $\ell \leq k$ and $f(\ell) = +\infty$ for any $\ell \geq k + 1$, the *minimization* problem is the so-called k-path partition problem [20, 16, 8, 7, 9, 6]; when $f(\ell)$ is $f(\ell) = 0$ for any $\ell \leq |V| - 1$ but $f(|V|) \neq 0$, the *maximization* problem reduces to the Hamiltonian path problem.

Given an integer $k \geq 4$, in the special case where $f(\ell) = 0$ for any $\ell < k$ but $f(\ell) = \ell$ for any $\ell \geq k$, the *maximization* problem can be re-phrased as to find a set of vertex-disjoint paths of order at least k to cover the most vertices, denoted as MaxP^{k+}PC. The MaxP^{4+}PC problem (i.e., $k = 4$) is complementary to finding a path cover with the minimum number of paths of order at most 3 [4, 5], and thus it is NP-hard. Kobayashi et al. [15] presented a 4-approximation algorithm for MaxP^{4+}PC by greedily adding an order-4 path or extending an existing path to the longest possible.

For a fixed integer $k \geq 4$, the MaxP^{k+}PC problem is NP-hard too [15]; to the best of our knowledge there is no approximation algorithm designed directly for it. Nevertheless, the MaxP^{k+}PC problem can be cast as a special case of the Maximum Weighted $(2k - 1)$-Set Packing problem [11], by constructing a set of ℓ vertices when they are traceable (that is, they can be formed into a path) in the given graph and assigning its weight ℓ, for every $\ell = k, k + 1, \ldots, 2k - 1$. (This upper bound $2k - 1$ will become clear in the next section.) The Maximum Weighted $(2k - 1)$-Set Packing problem is APX-complete [13] and the best known approximation guarantee is $k - \frac{1}{63{,}700{,}992} + \epsilon$ for any $\epsilon > 0$ [17].

In this paper, we study the MaxP^{k+}PC problem from the approximation algorithm perspective. The problem and its close variants have many motivating real-life applications in various areas such as various (communication, routing, transportation, optical etc.) network design [14]. For example, when a local government plans to upgrade its subway infrastructures, the given map of rail tracks is to be decomposed into multiple disjoint lines of stations, each of which will be taken care of by a team of workers. Besides being disjoint so that while some lines are under construction the other lines can function properly, each line is expected long enough for the team to work on continuously during a shift without wasting time and efforts to move themselves and materials from one point to another. Viewing the map as a graph, the goal of planning is to find a collection of vertex-disjoint long paths to cover the most vertices (and of course, possibly under some other real traffic constraints).

We contribute two approximation algorithms for the MaxP^{k+}PC problem, the first of which is a $(0.4394k + O(1))$-approximation algorithm for any fixed integer $k \geq 4$, denoted as Approx1. We note that Approx1 is the first approximation algorithm directly designed for the MaxP^{k+}PC problem, and it is a local improvement algorithm that iteratively applies one of the three operations, addition, replacement and double-replacement, each takes $O(|V|^k)$ time and covers at least one more vertex. While the addition and the replacement operations have appeared in the 4-approximation algorithm for the MaxP^{4+}PC problem [15], the double-replacement operation is novel and it replaces one existing path in the current path collection with two new paths. At termination, that is, when none of the local improvement operations is applicable, we show by an amortization scheme that each path P in the computed solution is attributed with at most $\rho(k)n(P)$ vertices covered in an optimal solution, where $n(P)$ denotes the order of the path P and $\rho(k) \leq 0.4394k + 0.6576$ for any $k \geq 4$.

The second $O(n^8)$-time algorithm, denoted as Approx2, is for the MaxP^{4+}PC problem. Besides the three operations in Approx1, we design two additional operations, re-cover and look-ahead. The re-cover operation aims to increase the number of 4-paths in the solution, and the look-ahead operation covers at least one more vertex by trying multiple paths equivalent to an existing path in the current solution in order to execute a replacement operation. With these two more local improvement operations, we design a refined amortization scheme to show that, on average, each vertex covered in the computed solution is attributed with at most two vertices covered in an optimal solution. That is, Approx2 is a 2-approximation algorithm for the MaxP^{4+}PC problem. We also show a lower bound of $\frac{16}{9}$ on the worst-case performance ratio of Approx2.

The rest of the paper is organized as follows. In Section 2, we introduce the basic notations and definitions. Section 3 is devoted to the MaxP^{k+}PC problem, where we present the Approx1 algorithm and its performance analysis. In Section 4, we present the Approx2 algorithm for the MaxP^{4+}PC problem, and outline the performance analysis. Due to space limit, while we provide most part of the performance analyses for both algorithms here to convince the readers, some technical details are left out to the full version of the paper. We conclude the paper in the last section with some possible future work.

2 Preliminaries

For a fixed integer $k \geq 4$, in the MaxP^{k+}PC problem, we are given a simple undirected graph and want to find a collection of vertex-disjoint paths of order at least k to cover the maximum number of vertices.

We consider simple undirected graphs in this paper and we fix a graph G for discussion. Let $V(G)$ and $E(G)$ denote its vertex set and edge set in the graph G, respectively. We simplify $V(G)$ and $E(G)$ as V and E, respectively, when the underlying graph is clear from the context. We use $n(G)$ to denote the *order* of G, that is, $n = |V|$ is the number of vertices in the graph. A subgraph S of G is a graph such that $V(S) \subseteq V(G)$ and $E(S) \subseteq E(G)$; and likewise, $n(S) = |V(S)|$ denotes its order. Given a subset of vertices $R \subseteq V$, the subgraph of G *induced* on R is denoted as $G[R]$, of which the vertex set is R and the edge set contains all the edges of E each connecting two vertices of R. A (simple) path P in G is a subgraph of which the vertices can be ordered into $(v_1, v_2, \ldots, v_{n(P)})$ such that $E(P) = \{\{v_i, v_{i+1}\}, i = 1, 2, \ldots, n(P) - 1\}$. A path of order ℓ is called an *ℓ-path* (also often called a *length-$(\ell - 1)$ path* in the literature).

In this paper we are most interested in paths of order at least 4. In the sequel, given an ℓ-path P with $\ell \geq 4$, we let u_j denote the vertex of P at distance j from one ending vertex of P, for $0 \leq j \leq \lceil \frac{\ell}{2} \rceil - 1$, and v_j denote the vertex of P at distance j from the other ending vertex, for $0 \leq j \leq \lfloor \frac{\ell}{2} \rfloor - 1$. When ℓ is odd, then the center vertex of the path is $u_{\frac{\ell-1}{2}}$. This way, a $(2s + 1)$-path is represented as u_0-u_1-$\cdots$-u_{s-1}-u_s-v_{s-1}-$\cdots$-v_1-v_0, and a $(2s)$-path is represented as u_0-u_1-$\cdots$-u_{s-1}-v_{s-1}-$\cdots$–v_1-v_0. Though the path is undirected and the vertex naming is often arbitrary, sometimes we will pick a particular endpoint of the path to be the vertex u_0.

Given another path Q, let $Q - P$ denote the subgraph of Q by removing those vertices in $V(P)$, and the edges of $E(Q)$ incident at them, from Q. Clearly, if $V(P) \cap V(Q) = \emptyset$, then $Q - P = Q$; otherwise, $Q - P$ is a collection of sub-paths of Q each has at least one endpoint that is adjacent to some vertex on P through an edge of $E(Q)$. For a collection $\mathcal{P}$ of vertex-disjoint paths, it is also a subgraph of G, with its vertex set $V(\mathcal{P}) = \cup_{P\in\mathcal{P}} V(P)$ and edge set $E(\mathcal{P}) = \cup_{P\in\mathcal{P}} E(P)$. We similarly define $Q - \mathcal{P}$ to be the collection of sub-paths of Q after removing those vertices in $V(\mathcal{P})$ from $V(Q)$, together with the edges of $E(Q)$ incident at them. Furthermore, for another collection $\mathcal{Q}$ of vertex-disjoint paths, we can define $\mathcal{Q} - \mathcal{P}$ analogously, that is, $\mathcal{Q} - \mathcal{P}$ is the collection of sub-paths of the paths in $\mathcal{Q}$ after removing those vertices in $V(\mathcal{P})$ from $V(\mathcal{Q})$, together with the edges of $E(\mathcal{Q})$ incident at them.

▶ **Definition 1** (Associatedness). *Given two collections $\mathcal{P}$ and $\mathcal{Q}$ of vertex-disjoint paths, if a path S of $\mathcal{Q} - \mathcal{P}$ has an endpoint that is adjacent to a vertex v in $V(\mathcal{P}) \cap V(\mathcal{Q})$ in $\mathcal{Q}$, then we say S is* associated *with v.*

One sees that a path S of $\mathcal{Q} - \mathcal{P}$ can be associated with zero to two vertices in $V(\mathcal{P}) \cap V(\mathcal{Q})$, and conversely, a vertex of $V(\mathcal{P}) \cap V(\mathcal{Q})$ can be associated with zero to two paths in $\mathcal{Q} - \mathcal{P}$.

If the paths of the collection $\mathcal{P}$ all have order at least k, then the vertices of $V(\mathcal{P})$ are said *covered* by the paths of $\mathcal{P}$, or simply by $\mathcal{P}$. Let $R = V - V(\mathcal{P})$. For any vertex $v \in V(\mathcal{P})$, *an extension* $e(v)$ at the vertex v is a path in the subgraph $G[R]$ of G induced on R which has an endpoint adjacent to v in G. Note that there could be many extensions at v, and we use $n(v) = \max_{e(v)} n(e(v))$ to denote the order of the longest extensions at the vertex v.

▶ **Lemma 2.** *Given two collections $\mathcal{P}$ and $\mathcal{Q}$ of vertex-disjoint paths in the graph G, for every vertex $v \in V(\mathcal{P})$, any path of $\mathcal{Q} - \mathcal{P}$ associated with v has order at most $n(v)$.*

Proof. The lemma holds since $\mathcal{Q} - \mathcal{P}$ is a subgraph of the induced subgraph $G[V - V(\mathcal{P})]$, that is, every path of $\mathcal{Q} - \mathcal{P}$ is a path in $G[V - V(\mathcal{P})]$, and the associatedness (see Definition 1) is a special adjacency through an edge of $E(\mathcal{Q})$. ◀

Our goal is to compute a collection of vertex-disjoint paths of order at least k, such that it covers the most vertices. In our local improvement algorithms below, we start with the empty collection $\mathcal{P} = \emptyset$ to iteratively expand $V(\mathcal{P})$ through one of a few operations, to be defined later. Notice that for an ℓ-path with $\ell \ge 2k$, one can break it into a k-path and an $(\ell - k)$-path by deleting an edge. Since they cover the same vertices, we assume without loss of generality hereafter that any collection $\mathcal{P}$ inside our algorithms contains vertex-disjoint paths of order in between k and $2k - 1$, inclusive.

3 A $(0.4394k + 0.6576)$-approximation algorithm for MaxP^{k+}PC

For a given integer $k \ge 4$, the best known approximation algorithm for the Maximum Weighted $(2k - 1)$-Set Packing problem leads to an $O(n^{2k-1})$-time $(k - \frac{1}{63{,}700{,}992} + \epsilon)$-approximation algorithm for the MaxP^{k+}PC problem, for any $\epsilon > 0$ [17]. In this section, we define three local improvement operations for our algorithm for the MaxP^{k+}PC problem, denoted as Approx1. We show later that its time complexity is $O(n^{k+1})$ and its approximation ratio is at most $0.4394k + 0.6576$.

For the current path collection $\mathcal{P}$, if there is a path covering k-vertices outside of $V(\mathcal{P})$, then the following operation adds the k-path into $\mathcal{P}$.

▶ **Operation 3.** *For a k-path P in the induced subgraph $G[V - V(\mathcal{P})]$, the* Add(P) *operation adds P to $\mathcal{P}$.*

Since finding a k-path in the induced subgraph $G[V - V(\mathcal{P})]$, for any $\mathcal{P}$, can be done in $O(n^k)$ time, determining whether or not an addition operation is applicable, and if so then applying it, can be done in $O(n^k)$ time too. Such an operation increases $|V(\mathcal{P})|$ by k.

Recall that a path $P \in \mathcal{P}$ is represented as u_0-u_1-$\cdots$-v_1-v_0. Though it is undirected, we may regard u_0 the *head* vertex of the path and v_0 the *tail* vertex for convenience. The next operation seeks to extend a path of $\mathcal{P}$ by replacing a prefix (or a suffix) with a longer one.

▶ **Operation 4.** *For a path $P \in \mathcal{P}$ such that there is an index t and an extension $e(u_t)$ with $n(e(u_t)) \geq t+1$ (an extension $e(v_t)$ with $n(e(v_t)) \geq t+1$, respectively), the* Rep(P) *operation replaces the prefix u_0-u_1-$\cdots$-u_{t-1} of P by $e(u_t)$ (the suffix v_{t-1}-$\cdots$-v_1-v_0 of P by $e(v_t)$, respectively).*

Similarly, one sees that finding an extension $e(u_t)$ (of order at most $k-1$, or otherwise an Add operation is applicable) in the induced subgraph $G[V - V(\mathcal{P})]$, for any vertex $u_t \in P \in \mathcal{P}$, can be done in $O(n^{k-1})$ time. Therefore, determining whether or not a prefix or a suffix replacement operation is applicable, and if so then applying it, can be done in $O(n^k)$ time. Note that such an operation increases $|V(\mathcal{P})|$ by at least 1.

The third operation tries to use a prefix and a non-overlapping suffix of a path in $\mathcal{P}$ to grow them into two separate paths of order at least k.

▶ **Operation 5.** *For a path $P \in \mathcal{P}$ such that*

(i) *there are two indices t and j with $j \geq t+1$ and two vertex-disjoint extensions $e(u_t)$ and $e(u_j)$ with $n(e(u_t)) \geq k-(t+1)$ and $n(e(u_j)) \geq k-(n(P)-j)$, the* DoubleRep$(P)$ *operation replaces P by two new paths $P_1 = u_0$-u_1-$\cdots$-u_t-$e(u_t)$ and $P_2 = e(u_j)$-u_j-$\cdots$-v_1-v_0;*

(ii) *or there are two indices t and j and two vertex-disjoint extensions $e(u_t)$ and $e(v_j)$ with $n(e(u_t)) \geq k-(t+1)$ and $n(e(v_j)) \geq k-(j+1)$, the* DoubleRep(P) *operation replaces P by two new paths $P_1 = u_0$-u_1-$\cdots$-u_t-$e(u_t)$ and $P_2 = e(v_j)$-v_j-$\cdots$-v_1-v_0.*

Note that finding an extension $e(u_t)$ (of order at most t, or otherwise a Rep operation is applicable) can be limited to those indices $t \geq \frac{k-1}{2}$. Furthermore, we only need to find an extension $e(u_t)$ of order at most $\frac{k-1}{2}$ (equal to $\frac{k-1}{2}$ only if $t = \frac{k-1}{2}$). For the same reason, we only need to find an extension $e(v_j)$ of order at most $\frac{k-1}{2}$ for those indices $j \geq \frac{k-1}{2}$ (equal to $\frac{k-1}{2}$ only if $j = \frac{k-1}{2}$). Since $k \leq n(P) \leq 2k-1$ for any $P \in \mathcal{P}$, we only need to find an extension $e(u_j)$ of order at most $\frac{k-1}{2}$ too (equal to $\frac{k-1}{2}$ only if $j = \frac{n(P)-1}{2}$ and $n(P) = k$). In summary, finding the two vertex-disjoint extensions $e(u_t)$ and $e(u_j)$, or $e(u_t)$ and $e(v_j)$, in the induced subgraph $G[V - V(\mathcal{P})]$ can be done in $O(n^{k-1})$ time (in $\Theta(n^{k-1})$ for at most 2 pairs of t and j). It follows that determining whether or not a double replacement operation is applicable, and if so then applying it, can be done in $O(n^k)$ time. Also, such an operation increases $|V(\mathcal{P})|$ by at least 1 as the total number of vertices covered by the two new paths P_1 and P_2 is at least $2k$. We summarize the above observations on the three operations into the following lemma.

▶ **Lemma 6.** *Given a collection $\mathcal{P}$ of vertex-disjoint paths of order in between k and $2k-1$, determining whether or not one of the three operations* Add, Rep *and* DoubleRep *is applicable, and if so then applying it, can be done in $O(n^k)$ time. Each operation increases $|V(\mathcal{P})|$ by at least 1.*

Given a graph $G = (V, E)$, our approximation algorithm for the MaxP^{k+}PC problem, denoted as Approx1, is iterative. It starts with the empty collection $\mathcal{P} = \emptyset$; in each iteration, it determines whether any one of the three operations Add, Rep and DoubleRep is applicable, and if so then it applies the operation to update $\mathcal{P}$. During the entire process, $\mathcal{P}$ is maintained to be a collection of vertex-disjoint paths of order in between k and $2k-1$. The algorithm terminates if none of the three operations is applicable for the current $\mathcal{P}$, and returns it as the solution. A simple high level description of the algorithm Approx1 is given in Algorithm 1. From Lemma 6, we see that each operation improves the collection $\mathcal{P}$ to cover at least one more vertex. Therefore, the overall running time of Approx1 is in $O(n^{k+1})$.

Algorithm 1 Approx1 (high level description).

```
Input: A graph $G = (V, E)$;
1. initialize $\mathcal{P} = \emptyset$;
2. while (one of the operations Add, Rep, DoubleRep is applicable)
   2.1 apply the operation to update $\mathcal{P}$;
   2.2 break any path of order $2k$ or above into two paths, one of which is a $k$-path;
3. return the final $\mathcal{P}$.
```

Below we fix $\mathcal{P}$ to denote the collection of paths returned by our algorithm Approx1. The next three lemmas summarize the structural properties of $\mathcal{P}$, which are useful in the performance analysis.

▶ **Lemma 7.** *For any path Q of order at least k in the graph G, $V(Q) \cap V(\mathcal{P}) \neq \emptyset$.*

Proof. The lemma holds due to the termination condition of the algorithm Approx1, since otherwise an Add operation is applicable. ◀

▶ **Lemma 8.** *For any path $P \in \mathcal{P}$, $n(u_j) \le j$ and $n(v_j) \le j$ for any index j.*

Proof. The lemma holds due to the termination condition of the algorithm Approx1, since otherwise a Rep operation is applicable. ◀

▶ **Lemma 9.** *Suppose there is a vertex u_t on a path $P \in \mathcal{P}$ and an extension $e(u_t)$ with $n(e(u_t)) \ge k - t - 1$. Then,*

(i) *for any vertex u_j with $j \ge t + 1$, $j \le k - 2$ and every extension $e(u_j)$ vertex-disjoint to $e(u_t)$ has order $n(e(u_j)) \le k - j - 2$;*

(ii) *for any index j, every extension $e(v_j)$ vertex-disjoint to $e(u_t)$ has order $n(e(v_j)) \le k - j - 2$.*

Proof. The lemma holds due to the termination condition of the algorithm Approx1.

First, for any vertex u_j with $j \ge t + 1$, an extension $e(u_j)$ vertex-disjoint to $e(u_t)$ has order $n(e(u_j)) \le k - (n(P) - j) - 1$ since otherwise a DoubleRep operation is applicable. Using the fact that $n(P) \ge 2j + 1$, $n(e(u_j)) \le k - j - 2$. Then by $k - j - 2 \ge 0$, we have $j \le k - 2$.

Next, similarly, for any index j, a vertex-disjoint extension $e(v_j)$ to $e(u_t)$ has order $n(e(v_j)) \le k - j - 2$ since otherwise a DoubleRep operation is applicable. ◀

We next examine the performance of the algorithm Approx1. We fix $\mathcal{Q}$ to denote an optimal collection of vertex-disjoint paths of order at least k that covers the most vertices. We apply an amortization scheme to *assign* the vertices of $V(\mathcal{Q})$ to the vertices of $V(\mathcal{P}) \cap V(\mathcal{Q})$. We will show that, using the structural properties of $\mathcal{P}$ in Lemmas 7–9, the average number of vertices *received* by a vertex of $V(\mathcal{P})$ is upper bounded by $\rho(k)$, which is the approximation ratio of Approx1.

In the amortization scheme, we assign the vertices of $V(\mathcal{Q})$ to the vertices of $V(\mathcal{P}) \cap V(\mathcal{Q})$ as follows: Firstly, assign each vertex of $V(\mathcal{Q}) \cap V(\mathcal{P})$ to itself. Next, recall that $\mathcal{Q} - \mathcal{P}$ is the collection of sub-paths of the paths of $\mathcal{Q}$ after removing those vertices in $V(\mathcal{Q}) \cap V(\mathcal{P})$. By Lemma 7, each path S of $\mathcal{Q} - \mathcal{P}$ is associated with one or two vertices in $V(\mathcal{Q}) \cap V(\mathcal{P})$. If the path S is associated with only one vertex v of $V(\mathcal{Q}) \cap V(\mathcal{P})$, then all the vertices on S are assigned to the vertex v. If the path S is associated with two vertices v_1 and v_2 of $V(\mathcal{Q}) \cap V(\mathcal{P})$, then a half of the vertices on S are assigned to each of the two vertices v_1 and v_2. One sees that in the amortization scheme, all the vertices of $V(\mathcal{Q})$ are assigned to the vertices of $V(\mathcal{P}) \cap V(\mathcal{Q})$; conversely, each vertex of $V(\mathcal{P}) \cap V(\mathcal{Q})$ receives itself, plus some fraction of or all the vertices on one or two paths of $\mathcal{Q} - \mathcal{P}$. (We remark that the vertices of $V(\mathcal{P}) - V(\mathcal{Q})$, if any, receive nothing.)

▶ **Lemma 10.** *For any vertex u_j on a path $P \in \mathcal{P}$ with $j \le k-2$, if $n(u_j) \le k-j-2$, then u_j receives at most $\frac{3}{2}\min\{j, k-j-2\}+1$ vertices.*

Proof. By Lemma 8, we have $n(u_j) \le j$. Therefore, $n(u_j) \le \min\{j, k-j-2\}$. By Lemma 2, any path of $\mathcal{Q} - \mathcal{P}$ associated with u_j contains at most $\min\{j, k-j-2\}$ vertices.

If there is at most one path of $\mathcal{Q} - \mathcal{P}$ associated with u_j, then the lemma is proved.

Consider the remaining case where there are two paths of $\mathcal{Q} - \mathcal{P}$ associated with u_j. Since $2\min\{j, k-j-2\}+1 \le j + (k-j-2)+1 = k-1$, while the path $Q \in \mathcal{Q}$ containing the vertex u_j has order at least k, we conclude that one of these two paths of $\mathcal{Q} - \mathcal{P}$ is associated with another vertex of $V(\mathcal{Q}) \cap V(\mathcal{P})$. It follows from the amortization scheme that u_j receives at most $\frac{3}{2}\min\{j, k-j-2\}+1$ vertices. ◀

▶ **Lemma 11.** *Suppose s is an integer such that $\lfloor \frac{k}{2} \rfloor - 1 \le s \le k-2$. Then we have*

$$\sum_{j=1}^{s} \min\{j, k-j-2\} = ks + \frac{3}{2}k - \frac{1}{4}k^2 - \frac{5}{2}s - \frac{1}{2}s^2 - 2 - \frac{1}{4}(k \bmod 2),$$

where mod *is the modulo operation.*

Proof. The formula can be directly validated by distinguishing the two cases where k is even or odd, and using the fact that $\min\{j, k-j-2\} = j$ if and only if $j \le \lfloor \frac{k}{2} \rfloor - 1$. ◀

▶ **Theorem 12.** *The algorithm* Approx1 *is an $O(|V|^{k+1})$-time $\rho(k)$-approximation algorithm for the* MaxP^{k+}PC *problem, where $k \ge 4$ and*

$$\rho(k) = \begin{cases} \frac{3k+1}{2} - \frac{1}{4}\sqrt{18k^2-3}, & \text{if } k \text{ is odd;} \\ \frac{3k+1}{2} - \frac{1}{4}\sqrt{18k^2-21}, & \text{if } k \text{ is even.} \end{cases}$$

Proof. Recall that all the vertices of $V(\mathcal{Q})$ are assigned to the vertices of $V(\mathcal{Q}) \cap V(\mathcal{P})$, through our amortization scheme. Below we estimate for any path $P \in \mathcal{P}$ the total number of vertices received by the vertices of $V(P) \cap V(\mathcal{Q})$, denoted as $r(P)$, and we will show that $\frac{r(P)}{n(P)} \le \rho(k)$.

We fix a path $P \in \mathcal{P}$ for discussion. If it exists, we let t denote the smallest index j such that the vertex u_j on the path P is associated with a path $e(u_j)$ of $\mathcal{Q} - \mathcal{P}$ with order $n(e(u_j)) \ge k-j-1$. Note that if necessary we may rename the vertices on P, so that the non-existence of t implies any path of $\mathcal{Q} - \mathcal{P}$ associated with the vertex u_j or v_j has order at most $k-j-2$, for any index j. Furthermore, by Lemma 9, if t exists, then any path of $\mathcal{Q} - \mathcal{P}$, except $e(u_t)$, associated with the vertex u_j or v_j has order at most $k-j-2$, for any index $j \ne t$. We remark that $e(u_t)$ could be associated with another vertex on the path P. When $n(P) = 2k-1$, t exists and $t = k-1$.

We distinguish three cases for $n(P)$ based on its parity and on whether it reaches the maximum value $2k-1$. Due to space limit, below we only discuss the first case in detail.

Case 1. $n(P) = 2s+1$ where $\frac{k-1}{2} \le s \le k-2$.

Since $s \ge \frac{k-1}{2}$, we have $\min\{s, k-s-2\} = k-s-2$. If the index t does not exist, that is, any path of $\mathcal{Q}-\mathcal{P}$ associated with the vertex u_j or v_j has order at most $k-j-2$, for any index j, then by Lemma 10 each of the vertices u_j and v_j receives at most $\frac{3}{2}\min\{j, k-j-2\}+1$ vertices. Hence we have

$$\begin{aligned}
r(P) &\le 2\sum_{j=0}^{s-1}\left(\frac{3}{2}\min\{j,k-j-2\}+1\right)+\frac{3}{2}\min\{s,k-s-2\}+1\\
&= 3\sum_{j=0}^{s}\min\{j,k-j-2\}-\frac{3}{2}\min\{s,k-s-2\}+(2s+1)\\
&\le 3\sum_{j=1}^{s}\min\{j,k-j-2\}-\frac{3}{2}k+\frac{7}{2}s+4. && (1)
\end{aligned}$$

If the index t exists, then by Lemmas 8 and 2, $n(e(u_t)) \le t$ and thus the vertex u_t receives at most $2t+1$ vertices. Note that if $e(u_t)$ is associated with another vertex on the path P, then we count all the vertices of $e(u_t)$ towards u_t but none to the other vertex (that is, we could overestimate $r(P)$). It follows from Lemma 10, the above Eq. (1), $t \le s$, and $s \ge \frac{k-1}{2}$ that

$$\begin{aligned}
r(P) &\le 3\sum_{j=1}^{s}\min\{j,k-j-2\}-\frac{3}{2}k+\frac{7}{2}s+4-\left(\frac{3}{2}\min\{t,k-t-2\}+1\right)+(2t+1)\\
&= 3\sum_{j=1}^{s}\min\{j,k-j-2\}-\frac{3}{2}k+\frac{7}{2}s+4+\left(2t-\frac{3}{2}\min\{t,k-t-2\}\right)\\
&= 3\sum_{j=1}^{s}\min\{j,k-j-2\}-\frac{3}{2}k+\frac{7}{2}s+4+\max\left\{\frac{1}{2}t,\frac{7}{2}t-\frac{3}{2}k+3\right\}\\
&\le 3\sum_{j=1}^{s}\min\{j,k-j-2\}-\frac{3}{2}k+\frac{7}{2}s+4+\max\left\{\frac{1}{2}s,\frac{7}{2}s-\frac{3}{2}k+3\right\}\\
&= 3\sum_{j=1}^{s}\min\{j,k-j-2\}-\frac{3}{2}k+\frac{7}{2}s+4+\left(\frac{7}{2}s-\frac{3}{2}k+3\right)\\
&= 3\sum_{j=1}^{s}\min\{j,k-j-2\}-3k+7s+7. && (2)
\end{aligned}$$

Combining Eqs. (1, 2), and by Lemma 11, in Case 1 we always have

$$r(P) \le 3\sum_{j=1}^{s}\min\{j,k-j-2\}-3k+7s+7 = 3ks+\frac{3}{2}k-\frac{3}{4}k^2-\frac{1}{2}s-\frac{3}{2}s^2+1-\frac{3}{4}(k \bmod 2).$$

Therefore, using $n(P) = 2s+1$ we have

$$\frac{r(P)}{n(P)} \le \begin{cases} \frac{3k+1}{2}-\frac{3}{8}(2s+1)-\frac{6k^2-1}{8(2s+1)} \le \frac{3k+1}{2}-\frac{1}{4}\sqrt{18k^2-3}, & \text{if } k \text{ is odd;}\\ \frac{3k+1}{2}-\frac{3}{8}(2s+1)-\frac{6k^2-7}{8(2s+1)} \le \frac{3k+1}{2}-\frac{1}{4}\sqrt{18k^2-21}, & \text{if } k \text{ is even,}\end{cases}$$

where the upper bound $\rho(k)$ is achieved when $n(P) \approx \sqrt{2}k$.

Case 2 where $n(P) = 2k - 1$ and Case 3 where $n(P) = 2s$ with $\frac{k}{2} \le s \le k - 1$ are discussed similarly, and the estimated upper bounds on $\frac{r(P)}{n(P)}$ are slightly less than the above $\rho(k)$. Therefore, they together prove that the worst-case performance ratio of the algorithm Approx1 is at most $\rho(k)$, for any $k \ge 4$. Note that $\rho(k) \le \frac{3k+1}{2} - \frac{1}{4}\sqrt{18k^2 - 21} \le 0.4394k + 0.6576$. ◀

4 A 2-approximation algorithm for MaxP^{4+}PC

One sees that the algorithm Approx1 is an $O(n^5)$-time 2.4150-approximation algorithm for the MaxP^{4+}PC problem, which improves the previous best $O(n^5)$-time 4-approximation algorithm proposed in [15] and $O(n^7)$-time $(4 - \frac{1}{63{,}700{,}992} + \epsilon)$-approximation algorithm implied by [17]. In the amortized analysis for Approx1, when a path of $\mathcal{Q} - \mathcal{P}$ is associated with two vertices of $V(\mathcal{Q}) \cap V(\mathcal{P})$, one half of the vertices on this path is assigned to each of these two vertices. We will show that when $k = 4$, that is, for the MaxP^{4+}PC problem, the assignment can be done slightly better by observing where these two vertices are on the paths of $\mathcal{P}$ and then assigning vertices accordingly. To this purpose, we will need to refine the algorithm using two more local improvement operations, besides the three Add, Rep and DoubleRep operations. We denote our algorithm for the MaxP^{4+}PC problem as Approx2.

We again use $\mathcal{P}$ to denote the path collection computed by Approx2. Since Approx2 employs the three Add, Rep and DoubleRep operations, all of which are not applicable at termination, the structural properties stated in Lemmas 7–9 continue to hold, and we summarize them specifically using $k = 4$.

▶ **Lemma 13.** *For any path $P \in \mathcal{P}$,*

(1) $4 \le n(P) \le 7$;
(2) $n(u_j) \le j$ *and* $n(v_j) \le j$*, for any valid index* j;
(3) *if* $n(P) = 6$, $n(u_2) > 0$ *and* $n(v_2) > 0$*, then both* u_2 *and* v_2 *are adjacent to a unique vertex in* $V - V(\mathcal{P})$*, and hence they have common extensions;*
(4) *if* $n(P) = 7$*, then* $n(u_2) = n(v_2) = 0$.

We also fix $\mathcal{Q}$ to denote an optimal collection of vertex-disjoint paths of order at least 4 that covers the most vertices. Using Lemma 13, we may attach a longest possible extension in $\mathcal{Q} - \mathcal{P}$ to every vertex of a path of $\mathcal{P}$, giving rise to the *worst cases* illustrated in Figure 1, with respect to the order of the path. Since a vertex of a path $P \in \mathcal{P}$ can be associated with up to two paths in $\mathcal{Q} - \mathcal{P}$, the worst-case performance ratio of the algorithm Approx2 is at most $\max\{\frac{17}{7}, \frac{14}{6}, \frac{13}{5}, \frac{8}{4}\} = 2.6$. Our next two local improvement operations are designed to deal with three of the four worst cases where the path orders are $5, 6$ and 7. Afterwards, we will show by an amortization scheme that the average number of vertices assigned to a vertex of $V(\mathcal{P})$ is at most 2.

Since the average number of vertices assigned to a vertex on a 4-path in $\mathcal{P}$ is already at most 2, the first operation is employed to construct more 4-paths in $V(\mathcal{P})$, whenever possible.

▶ **Operation 14.** *For any two paths P and P' in $\mathcal{P}$ of order at least 5 such that their vertices are covered exactly by a set of paths in G of order at least 4 and of which at least one path has order 4, the* Re-cover(P, P') *operation replaces P and P' by this set of paths.*

In other words, the Re-cover(P, P') operation removes the two paths P and P' from $\mathcal{P}$, and then uses the set of paths to re-cover the same vertices. Since there are $O(n^2)$ possible pairs of paths P and P' in $\mathcal{P}$, and by $|V(P) \cup V(P')| \le 14$ the existence of a set of paths re-covering $V(P) \cup V(P')$ can be checked in $O(1)$ time, we conclude that determining whether

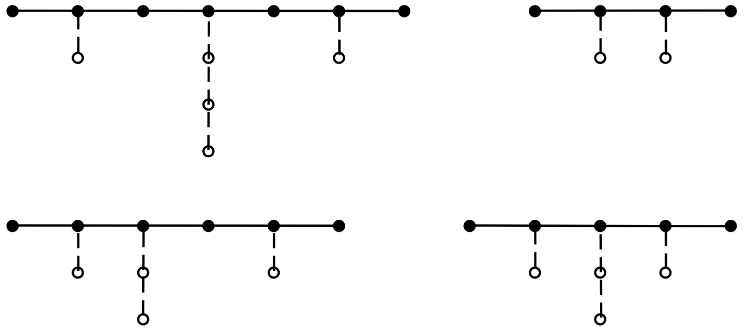


Figure 1 The worst case of a path $P \in \mathcal{P}$ with respect to its order, where a vertex on the path P is attached with a longest possible extension in $\mathcal{Q} - \mathcal{P}$, with its edges dashed. The edges on the path P are solid and the vertices are shown filled, while the vertices on the extensions are unfilled.

or not a Re-cover operation is applicable, and if so then applying it, can be done in $O(n^2)$ time. Note that such an operation does not change $|V(\mathcal{P})|$, but it increases the number of 4-paths by at least 1. For example, if $n(P) = 5$ and $n(P') = 7$, and we use u'_j's and v'_j's to label the vertices on P', then a Re-cover operation is applicable when u_0 is adjacent to any one of u'_0, u'_2, v'_2, v'_0 (resulting in three 4-paths); or if $n(P) = n(P') = 6$, then a Re-cover operation is applicable when u_0 is adjacent to any one of u'_0, u'_1, v'_1, v'_0 (resulting in three 4-paths).

▶ **Operation 15.** *For a path $P \in \mathcal{P}$ such that there is an index $t \in \{2, 3\}$ and an extension $e(u_t)$ with $n(e(u_t)) = t$,*

- **(i)** *if replacing P by the path $e(u_t)$-u_t-$\cdots$-v_1-v_0 enables a Rep operation, then the* Look-ahead(P) *operation first replaces P by $e(u_t)$-u_t-$\cdots$-v_1-v_0 and next executes the Rep operation;*
- **(ii)** *if $n(P) = 6$ and one of v_0 and v_2 is adjacent to a vertex w, such that w is on $e(u_2)$ or on u_0-u_1 or on another path of $\mathcal{P}$ but at distance at most 1 from one end, then the* Look-ahead(P) *operation first replaces P by the path u_0-u_1-u_2-$e(u_2)$ and next uses v_0-v_1-v_2 as an extension $e(w)$ to execute a Rep operation.*

In some sense, the Look-ahead(P) operation looks one step ahead to see whether or not using the extension $e(u_t)$ in various ways would help cover more vertices. Recall that we can rename the vertices on a path of $\mathcal{P}$, if necessary, and thus the above definition of a Look-ahead operation applies to the vertex v_2 symmetrically, if there is an extension $e(v_2)$ with $n(e(v_2)) = 2$. Also, when u_t is the center vertex of the path P (i.e., $n(P) = 5, 7$), one should also examine replacing P by the path u_0-u_1-$\cdots$-u_t-$e(u_t)$.

When the first case of a Look-ahead operation applies, its internal Rep operation must involve at least two of the t vertices $u_0, u_1, \ldots, u_{t-1}$[2] because otherwise an Add or a Rep operation would be applicable before this Look-ahead operation. Since there are $O(n^3)$ possible extensions at the vertex u_t, it follows from Lemma 6 that determining whether or not the first case of a Look-ahead operation is applicable, and if so then applying it, can be done in $O(n^6)$ time. Note that such an operation increases $|V(\mathcal{P})|$ by at least 1.

[2] First, at least one of these t vertices $u_0, u_1, \ldots, u_{t-1}$, say u, should be on the extension used in the internal Rep operation; then the extension extends to the/a vertex adjacent to u.

When the second case of a Look-ahead operation applies, we see that after the replacement (which reduces $|V(\mathcal{P})|$ by 1), the vertex w is always on a path of $\mathcal{P}$ at distance at most 1 from one end; and thus the succeeding Rep operation increases $|V(\mathcal{P})|$ by at least 2. The net effect is that such a Look-ahead operation increases $|V(\mathcal{P})|$ by at least 1. Since there are $O(n^2)$ possible extensions at the vertex u_2, determining whether or not the second case of a Look-ahead operation is applicable, and if so then applying it, can be done in $O(n^4)$ time.

We summarize the above observations on the two new operations into the following lemma.

▶ **Lemma 16.** *Given a collection $\mathcal{P}$ of vertex-disjoint paths of order in between 4 and 7, determining whether or not one of the two operations* Re-cover *and* Look-ahead *is applicable, and if so then applying it, can be done in $O(n^6)$ time. Each operation either increases $|V(\mathcal{P})|$ by at least 1, or keeps $|V(\mathcal{P})|$ unchanged and increases the number of 4-paths by at least 1.*

We are now ready to present the algorithm Approx2 for the MaxP^{4+}PC problem, which in fact is very similar to Approx1. It starts with the empty collection $\mathcal{P} = \emptyset$; in each iteration, it determines in order whether any one of the five operations Add, Rep, DoubleRep, Re-cover, and Look-ahead is applicable, and if so then it applies the operation to update $\mathcal{P}$. During the entire process, $\mathcal{P}$ is maintained to be a collection of vertex-disjoint paths of order in between 4 and 7. The algorithm terminates if none of the five operations is applicable for the current $\mathcal{P}$, and returns it as the solution. A simple high level description of the algorithm Approx2 is given in Algorithm 2. From Lemmas 6 and 16, the overall running time of Approx2 is in $O(n^8)$. Next we show that the worst-case performance ratio of Approx2 is at most 2, and thus its higher running time is paid off. The performance analysis is done through a similar but more careful amortization scheme.

Algorithm 2 Approx2 (high level description).

```
Input: A graph $G = (V, E)$;
1. initialize $\mathcal{P} = \emptyset$;
2. while (one of Add, Rep, DoubleRep, Re-cover and Look-ahead is applicable)
   2.1 apply the operation to update $\mathcal{P}$;
3. return the final $\mathcal{P}$.
```

The amortization scheme assigns the vertices of $V(\mathcal{Q})$ to the vertices of $V(\mathcal{P}) \cap V(\mathcal{Q})$. Firstly, each vertex of $V(\mathcal{Q}) \cap V(\mathcal{P})$ is assigned to itself. Next, recall that $\mathcal{Q} - \mathcal{P}$ is the collection of sub-paths of the paths of $\mathcal{Q}$ after removing those vertices in $V(\mathcal{Q}) \cap V(\mathcal{P})$. By Lemma 7, each path S of $\mathcal{Q} - \mathcal{P}$ is associated with one or two vertices in $V(\mathcal{Q}) \cap V(\mathcal{P})$.

When the path S of $\mathcal{Q} - \mathcal{P}$ is associated with two vertices v and v' of $V(\mathcal{Q}) \cap V(\mathcal{P})$, a half of all the vertices on S are assigned to each of v and v', except the *first* special case below. In this *first* special case, $n(S) = 1$, $v = u_1$ (or $v = v_1$, respectively) and $v' = u'_3$ on some paths $P, P' \in \mathcal{P}$ with $n(P) \geq 5$ and $n(P') = 7$, respectively; then the whole vertex on S is assigned to u'_3 (that is, none is assigned to u_1, or v_1, respectively).

When the path S of $\mathcal{Q} - \mathcal{P}$ is associated with only one vertex v of $V(\mathcal{Q}) \cap V(\mathcal{P})$, all the vertices on S are assigned to the vertex v, except the *second* and the *third* special cases below where $n(S) = 1$ and $v = u_1$ (or $v = v_1$, respectively) on a path $P \in \mathcal{P}$ with $n(P) \geq 5$. In the *second* special case, S-u_1-$[r]$-u'_3 is a subpath of some path $Q \in \mathcal{Q}$, where u'_3 is the center vertex of some 7-path $P' \in \mathcal{P}$, and $[r]$ means the vertex r might not exist but if it exists then it is not the center vertex of any 7-path in $\mathcal{P}$; in this case, a half of the vertex on S

is assigned to each of u_1 and u'_3.[3] In the *third* special case, S-u_1-u'_2 is a subpath of some path $Q \in \mathcal{Q}$, where $u'_2 \neq u_2$ is on some path $P' \in \mathcal{P}$ of order 5 or 6; in this case, a half of the vertex on S is assigned to each of u_1 and u'_2.

One sees that in the amortization scheme, all the vertices of $V(\mathcal{Q})$ are assigned to the vertices of $V(\mathcal{P}) \cap V(\mathcal{Q})$; conversely, each vertex of $V(\mathcal{P}) \cap V(\mathcal{Q})$ receives itself, plus some or all the vertices on its associated paths of $\mathcal{Q} - \mathcal{P}$, and some u_2, v_2 vertices on 5-/6-paths and some u_3 vertices on 7-paths could receive some additional vertices not on their associated paths, through the three special cases. (We remark that a vertex and its received vertices are on the same path in $\mathcal{Q}$, and that the vertices of $V(\mathcal{P}) - V(\mathcal{Q})$ receive nothing.)

The following lemma presents the joint effect of the above amortization scheme, of which the proof is omitted due to space limit.

▶ **Lemma 17.**

1. *For any path $P \in \mathcal{P}$, each of the vertices u_0 and v_0 receives at most 1 vertex; each of the vertices u_1 and v_1 receives at most $\frac{5}{2}$ vertices; when $n(P) = 7$, each of the vertices u_2 and v_2 receives at most 1 vertex.*
2. *For any 5-path $P \in \mathcal{P}$, if the vertex u_2 receives more than 3 vertices, then each of the vertices u_1 and v_1 receives at most $\frac{3}{2}$ vertices.*
3. *For any 6-path $P \in \mathcal{P}$, if the vertices u_2 and v_2 together receive more than 5 vertices, then the total number of vertices received by all the vertices of P is at most 12.*
4. *For any 7-path $P \in \mathcal{P}$, if the vertex u_3 receives more than 5 vertices, then the vertices u_0 and u_1 (v_0 and v_1, respectively) together receive at most $\frac{5}{2}$ vertices.*

▶ **Theorem 18.** *The algorithm* Approx2 *is an $O(|V|^8)$-time 2-approximation algorithm for the* MaxP^{4+}PC *problem, and $\frac{16}{9}$ is a lower bound on its performance ratio.*

Proof. Similar to the proof of Theorem 12, we will show that $\frac{r(P)}{n(P)} \le 2$ for any path $P \in \mathcal{P}$, where $r(P)$ denotes the total number of vertices received by all the vertices on P through the amortization scheme. We do this by differentiating the path order $n(P)$, which is in between 4 and 7.

Case 1. $n(P) = 4$. By Lemma 17.1, we have $r(P) \le 1 + \frac{5}{2} + \frac{5}{2} + 1 = 7$. It follows that $\frac{r(P)}{n(P)} \le \frac{7}{4}$.

Case 2. $n(P) = 5$. If the vertex u_2 receives at most 3 vertices, then by Lemma 17.1, we have $r(P) \le 3 + 7 = 10$. Otherwise, u_2 receives at most 5 vertices, and by Lemma 17.2 each of u_1 and v_1 receives at most $\frac{3}{2}$ vertices. It follows again by Lemma 17.1 that $r(P) \le 5 + 5 = 10$. That is, either way we have $r(P) \le 10$ and thus $\frac{r(P)}{n(P)} \le 2$.

Case 3. $n(P) = 6$. If the vertices u_2 and v_2 together receive at most 5 vertices, then by Lemma 17.1, we have $r(P) \le 5 + 7 = 12$. Otherwise, by Lemma 17.3 we have $r(P) \le 12$. That is, either way we have $r(P) \le 12$ and thus $\frac{r(P)}{n(P)} \le 2$.

Case 4. $n(P) = 7$. If the vertex u_3 receives at most 5 vertices, then by Lemma 17.1, we have $r(P) \le 5 + 2 + 7 = 14$. Otherwise, u_3 receives at most 7 vertices, and by Lemma 17.4 the vertices u_0 and u_1 (v_0 and v_1, respectively) together receive at most $\frac{5}{2}$ vertices. It follows again by Lemma 17.1 that $r(P) \le 7 + 2 + 5 = 14$. That is, either way we have $r(P) \le 14$ and thus $\frac{r(P)}{n(P)} \le 2$.

This proves that Approx2 is a 2-approximation algorithm.

[3] In the second special case, if r exists and $r \notin V(\mathcal{P})$, then it falls into the first special case and, as a result, the vertex u_1 receives only $\frac{1}{2}$ vertex and the vertex u'_3 receives $\frac{3}{2}$ vertices.

One might wonder whether the performance analysis can be done better. Though we are not able to show the tightness of the performance ratio 2, we give below a graph to show that $\frac{16}{9}$ is a lower bound.

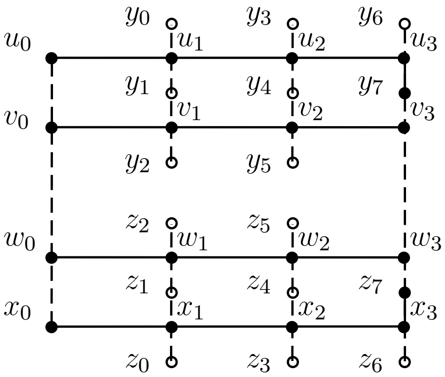


Figure 2 A graph of order 32 to show that the performance ratio of the algorithm APPROX2 is lower bounded by $\frac{16}{9}$. All the edges in the graph are shown, either solid or dashed. The 18 filled vertices are covered by the collection of two 5-paths and two 4-paths computed by APPROX2, and the edges on these paths are shown solid; the edges on an optimal collection of paths, which covers all the vertices, are shown dashed.

The graph displayed in Figure 2 contains 32 vertices. An optimal solution covers all the vertices, and its paths are (vertical) u_0-v_0-w_0-x_0, y_0-u_1-y_1-v_1-y_2, z_0-x_1-z_1-w_1-z_2, y_3-u_2-y_4-v_2-y_5, z_3-x_2-z_4-w_2-z_5, y_6-u_3-y_7-v_3, and w_3-z_7-x_3-z_6. Assuming the algorithm APPROX2 adds four 4-paths (horizontal) u_0-u_1-u_2-u_3, v_0-v_1-v_2-v_3, w_0-w_1-w_2-w_3, x_0-x_1-x_2-x_3, and then extends the first and the last to u_0-u_1-u_2-u_3-y_7, x_0-x_1-x_2-x_3-z_7. Then none of the five operations can be applied to improve the solution, which covers a total of 18 vertices only. That is, the algorithm APPROX2 only achieves a performance ratio of $\frac{16}{9}$ on the graph. ◀

5 Conclusion

In this paper, we studied the general vertex covering problem MAXP^{k+}PC, where $k \geq 4$, to find a collection of vertex-disjoint paths of order at least k to cover the most vertices in the input graph. The problem seemingly escapes from the literature, but it admits a k-approximation algorithm by reducing to the weighted $(2k-1)$-set packing problem [17]. We proposed the first direct $(0.4394k + O(1))$-approximation algorithm and an improved 2-approximation algorithm when $k = 4$. Both algorithms are local improvement based on a few operations, and we proved their approximation ratios via amortized analyses.

We suspect our amortized analyses are tight, and it would be interesting to either show the tightness or improve the analyses. For designing improved approximation algorithms, one can look into whether the two new operations in APPROX2 for $k = 4$ can be helpful for $k \geq 5$; other different ideas might also work, for example, one can investigate whether or not a maximum path-cycle cover [10] can be taken advantage of.

On the other hand, we haven't addressed whether or not the MAXP^{k+}PC problem, for a fixed $k \geq 4$, is APX-hard, and if it is so, then it is worthwhile to show some non-trivial lower bounds on the approximation ratio, even only for $k = 4$.

References

1 K. Asdre and S. D. Nikolopoulos. A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs. *Networks*, 50:231–240, 2007.

2 K. Asdre and S. D. Nikolopoulos. A polynomial solution to the k-fixed-endpoint path cover problem on proper interval graphs. *Theoretical Computer Science*, 411:967–975, 2010.

3 P. Berman and M. Karpinski. 8/7-approximation algorithm for (1,2)-TSP. In *ACM-SIAM Proceedings of the Seventeenth Annual Symposium on Discrete Algorithms (SODA'06)*, pages 641–648, 2006.

4 Y. Cai, G. Chen, Y. Chen, R. Goebel, G. Lin, L. Liu, and An Zhang. Approximation algorithms for two-machine flow-shop scheduling with a conflict graph. In *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON 2018)*, LNCS 10976, pages 205–217, 2018.

5 Y. Chen, Y. Cai, L. Liu, G. Chen, R. Goebel, G. Lin, B. Su, and A. Zhang. Path cover with minimum nontrivial paths and its application in two-machine flow-shop scheduling with a conflict graph. *Journal of Combinatorial Optimization*, 2021. Accepted on August 2, 2021.

6 Y. Chen, Z.-Z. Chen, C. Kennedy, G. Lin, Y. Xu, and A. Zhang. Approximation algorithms for the directed path partition problems. In *Proceedings of FAW 2021*, LNCS 12874, pages 23–36, 2021.

7 Y. Chen, R. Goebel, G. Lin, L. Liu, B. Su, W. Tong, Y. Xu, and A. Zhang. A local search 4/3-approximation algorithm for the minimum 3-path partition problem. In *Proceedings of FAW 2019*, LNCS 11458, pages 14–25, 2019.

8 Y. Chen, R. Goebel, G. Lin, B. Su, Y. Xu, and A. Zhang. An improved approximation algorithm for the minimum 3-path partition problem. *Journal of Combinatorial Optimization*, 38:150–164, 2019.

9 Y. Chen, R. Goebel, B. Su, W. Tong, Y. Xu, and A. Zhang. A 21/16-approximation for the minimum 3-path partition problem. In *Proceedings of ISAAC 2019*, LIPIcs 149, pages 46:1–46:20, 2019.

10 H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC'83)*, pages 448–456, 1983.

11 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

12 R. Gómez and Y. Wakabayashi. Nontrivial path covers of graphs: Existence, minimization and maximization. *Journal of Combinatorial Optimization*, 39:437–456, 2020.

13 D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

14 N. Immorlica, M. Mahdian, and V. Mirrokni. Cycle cover with short cycles. In *Proceedings of STACS 2005*, LNCS 3404, pages 641–653, 2005.

15 K. Kobayashi, G. Lin, E. Miyano, T. Saitoh, A. Suzuki, T. Utashima, and T. Yagita. Path cover problems with length cost. In *Proceedings of WALCOM 2022*, LNCS 13174, pages 396–408, 2022.

16 J. Monnot and S. Toulouse. The path partition problem and related problems in bipartite graphs. *Operations Research Letters*, 35:677–684, 2007.

17 M. Neuwohner. An improved approximation algorithm for the maximum weight independent set problem in d-claw free graphs. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, LIPIcs 187, pages 53:1–53:20, 2021.

18 L. L. Pao and C. H. Hong. The two-equal-disjoint path cover problem of matching composition network. *Information Processing Letters*, 107:18–23, 2008.

19 R. Rizzi, A. I. Tomescu, and V. Mäkinen. On the complexity of minimum path cover with subpath constraints for multi-assembly. *BMC Bioinformatics*, 15:S5, 2014.

20 J.-H. Yan, G. J. Chang, S. M. Hedetniemi, and S. T. Hedetniemi. k-path partitions in trees. *Discrete Applied Mathematics*, 78:227–233, 1997.

The Hamilton Compression of Highly Symmetric Graphs

Petr Gregor
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Arturo Merino
Department of Mathematics, TU Berlin, Germany

Torsten Mütze
Department of Computer Science, University of Warwick, United Kingdom
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Abstract

We say that a Hamilton cycle $C = (x_1, \ldots, x_n)$ in a graph G is k-symmetric, if the mapping $x_i \mapsto x_{i+n/k}$ for all $i = 1, \ldots, n$, where indices are considered modulo n, is an automorphism of G. In other words, if we lay out the vertices $x_1, \ldots, x_n$ equidistantly on a circle and draw the edges of G as straight lines, then the drawing of G has k-fold rotational symmetry, i.e., all information about the graph is compressed into a $360°/k$ wedge of the drawing. We refer to the maximum k for which there exists a k-symmetric Hamilton cycle in G as the *Hamilton compression of* G. We investigate the Hamilton compression of four different families of vertex-transitive graphs, namely hypercubes, Johnson graphs, permutahedra and Cayley graphs of abelian groups. In several cases we determine their Hamilton compression exactly, and in other cases we provide close lower and upper bounds. The cycles we construct have a much higher compression than several classical Gray codes known from the literature. Our constructions also yield Gray codes for bitstrings, combinations and permutations that have few tracks and/or that are balanced.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Combinatorics

Keywords and phrases Hamilton cycle, Gray code, hypercube, permutahedron, Johnson graph, Cayley graph, abelian group, vertex-transitive

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.54

Related Version Full proofs are provided in the preprint version of this extended abstract:
Full Version: `https://arxiv.org/abs/2205.08126` [15]

Funding This work was supported by Czech Science Foundation grant GA 22-15272S.
Petr Gregor: Partially supported by ARRS project P1-0383.
Arturo Merino: German Science Foundation grant 413902284 and ANID Becas Chile 2019-72200522
Torsten Mütze: German Science Foundation grant 413902284

Acknowledgements We thank Fedor Petrov and Michal Koucký for ideas that simplified some proofs in the full version. We also thank the reviewers for their helpful comments.

1 Introduction

A *Hamilton cycle* in a graph is a cycle that visits every vertex of the graph exactly once. This concept is named after the Irish mathematician and astronomer Sir William Rowan Hamilton (1805–1865), who invented the Icosian game, in which the objective is to find a Hamilton cycle along the edges of the dodecahedron. Figure 1 shows the dodecahedron with a Hamilton cycle on the circumference. Hamilton cycles have been studied intensively from

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 54; pp. 54:1–54:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

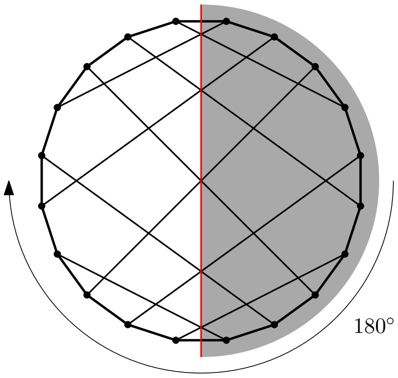


Figure 1 The dodecahedron with a 2-symmetric Hamilton cycle.

various different angles, such as graph theory (necessary/sufficient conditions, packing and covering etc. [11, 12, 13, 18]), optimization (shortest tours, approximation [2]), algorithms (complexity [10], exhaustive generation [22, 25]) and algebra (Cayley graphs [5, 19, 24, 28]). In this work we introduce a new graph parameter that quantifies how symmetric a Hamilton cycle in a graph can be. For example, the cycle in the dodecahedron shown in Figure 1 is 2-symmetric, as the drawing has 2-fold (i.e., $360°/2 = 180°$) rotational symmetry.

1.1 Hamilton cycles with rotational symmetry

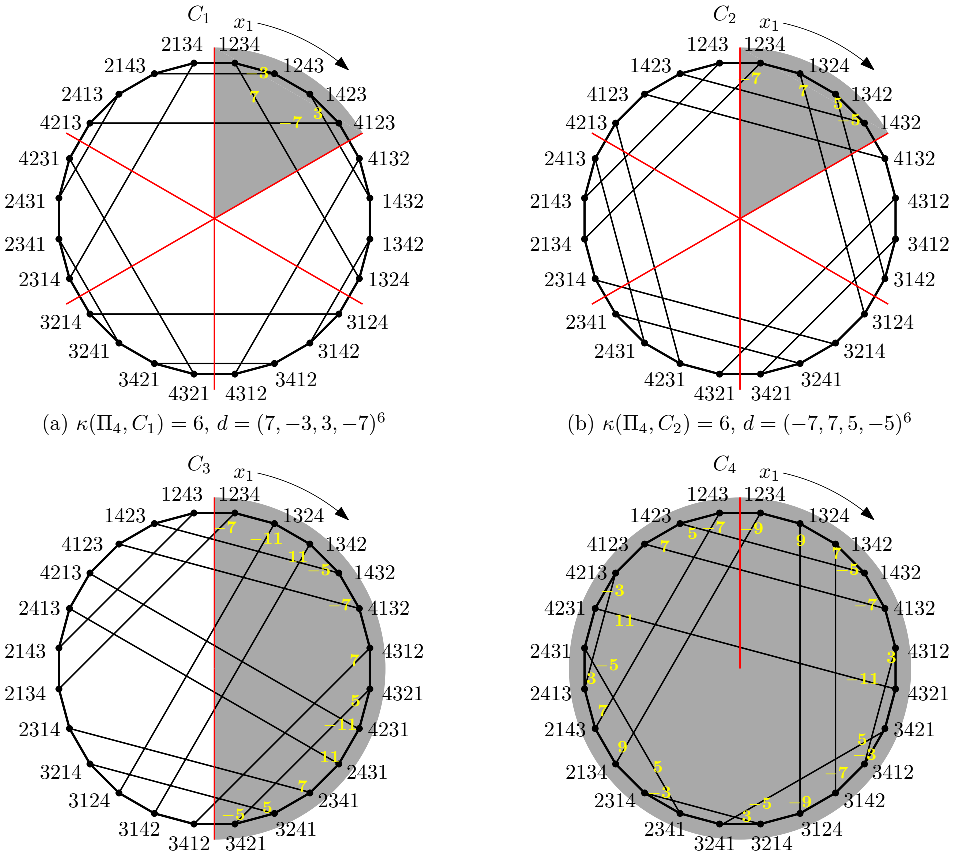
Formally, let $G = (V, E)$ be a graph with n vertices. We say that a Hamilton cycle $C = (x_1, \ldots, x_n)$ is *k-symmetric* if the mapping $f : V \to V$ defined by $x_i \mapsto x_{i+n/k}$ for all $i = 1, \ldots, n$, where indices are considered modulo n, is an automorphism of G. I.e., we have

$$C = P, f(P), f^2(P), \ldots, f^{k-1}(P) \quad \text{for the path} \quad P := (x_1, \ldots, x_{n/k}). \tag{1}$$

The idea is that the entire cycle C can be reconstructed from the path P, which contains only a $1/k$-fraction of all vertices, by repeatedly applying the automorphism f to it. In other words, if we lay out the vertices $x_1, \ldots, x_n$ equidistantly on a circle, and draw edges of G as straight lines, then we obtain a drawing of G with k-fold rotational symmetry, i.e., f is a rotation by $360°/k$; see Figure 2. We refer to the maximum k for which the Hamilton cycle C of G is k-symmetric as the *compression factor of* C, and we denote it by $\kappa(G, C)$.

1.2 Connection to LCF notation

There is yet another interesting interpretation of the compression factor in terms of the LCF notation of a graph, named after its inventors Lederberg, Coxeter and Frucht (see [9]). The idea is to describe a 3-regular Hamiltonian graph (such as the dodecahedron) concisely by considering one of its Hamilton cycles $C = (x_1, \ldots, x_n)$. Each vertex x_i has the neighbors x_{i-1} and x_{i+1} (modulo n) in the graph, plus a third neighbor x_j, which is $d_i := j - i$ (modulo n) steps away from x_i along the cycle. The *LCF sequence* of G is the sequence $d = (d_1, \ldots, d_n)$, where each d_i is chosen so that $-n/2 < d_i \leq n/2$. Clearly, we also have $d_i \notin \{-1, 0, +1\}$. Note that if C is k-symmetric, then the LCF sequence d of G is k-periodic, i.e., it has the form $d = (d_1, \ldots, d_{n/k})^k$, where the k in the exponent denotes k-fold repetition; see Figure 2. While LCF notation is only defined for 3-regular graphs, we can easily extend it to arbitrary graphs with a Hamilton cycle $C = (x_1, \ldots, x_n)$, by considering a sequence of sets $D = (D_1, \ldots, D_n)$, where D_i is the set of distances to all neighbors of x_i on the cycle except x_{i-1} and x_{i+1}; see Figure 3 (a)+(d). As before, if C is k-symmetric, then the corresponding sequence D is k-periodic, i.e., it has the form $D = (D_1, \ldots, D_{n/k})^k$. Frucht [9] writes:



(a) $\kappa(\Pi_4, C_1) = 6$, $d = (7, -3, 3, -7)^6$

(b) $\kappa(\Pi_4, C_2) = 6$, $d = (-7, 7, 5, -5)^6$

(c) $\kappa(\Pi_4, C_3) = 2$,
$d = (-7, -11, 11, -5, -7, 7, 5, -11, 11, 7, 5, -5)^2$

(d) $\kappa(\Pi_4, C_4) = 1$,
$d = (-9, 9, 7, -5, -7, 3, -11, 5, -3, -7, -9, 3,$
$-5, 5, -3, 9, 7, 3, -5, 11, -3, 7, 5, -7)$

Figure 2 Hamilton cycles $C_1, \ldots, C_4$ in the 4-permutahedron Π_4 with different LCF sequences and compression factors.

> "What happens with the LCF notation if we replace one hamiltonian circuit by another one? The answer is: nearly everything can happen! Indeed the LCF notation for a graph can remain unaltered or it can change completely [...] In such cases we should choose of course the shortest of the existing LCF notations."

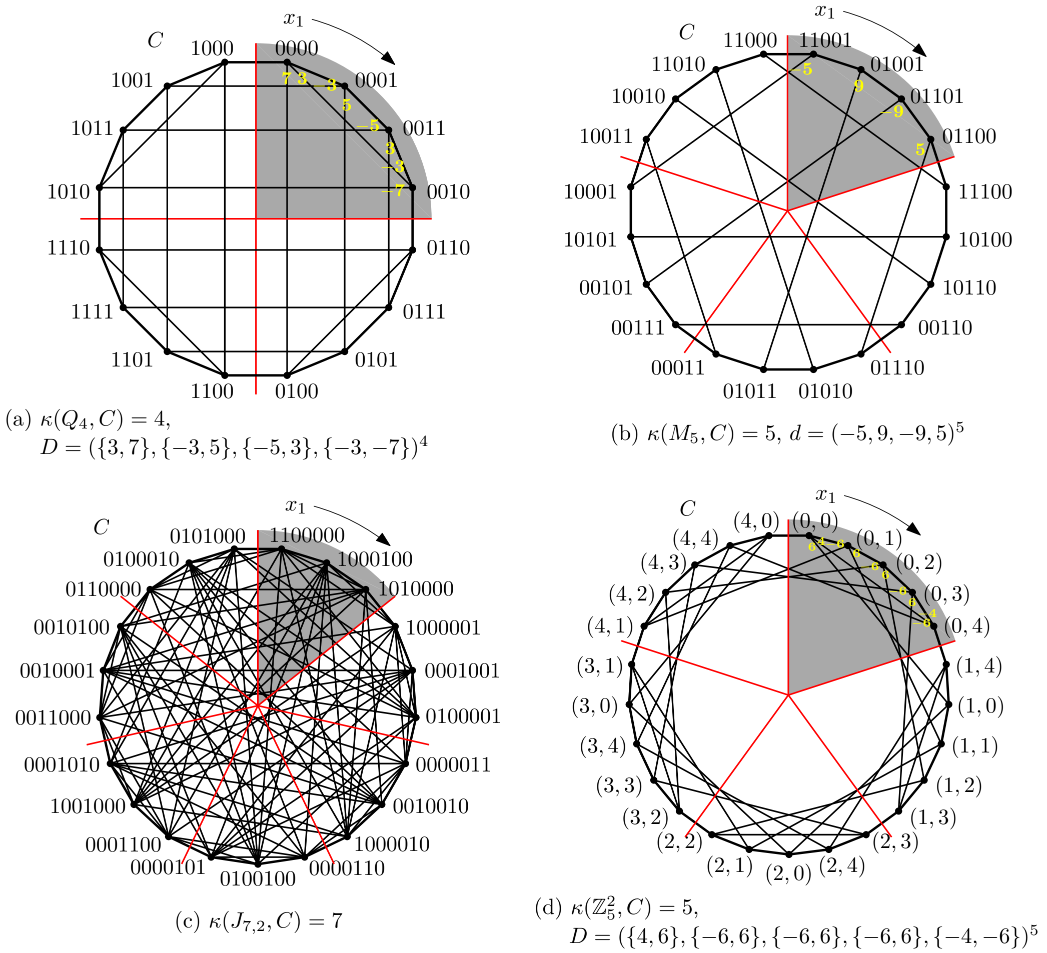
This observation is illustrated in Figure 2, which shows four different Hamilton cycles of the same graph G that have different LCF sequences and compression factors.

1.3 Hamilton compression

Frucht's suggestion is to search for a Hamilton cycle C in G whose compression factor $\kappa(G, C)$ is as large as possible. Formally, for any graph G we define

$$\kappa(G) := \max \big\{\kappa(G, C) \mid C \text{ is a Hamilton cycle in } G\big\}, \tag{2}$$

and we refer to this quantity as the *Hamilton compression of* G. If G has no Hamilton cycle, then we define $\kappa(G) := 0$. While the maximization in (2) is simply over all Hamilton cycles in G, and the automorphisms arise as possible rotations of those cycles, this definition is somewhat impractical to work with. In our arguments, we rather consider all automorphisms



(a) $\kappa(Q_4, C) = 4$,
$D = (\{3, 7\}, \{-3, 5\}, \{-5, 3\}, \{-3, -7\})^4$

(b) $\kappa(M_5, C) = 5$, $d = (-5, 9, -9, 5)^5$

(c) $\kappa(J_{7,2}, C) = 7$

(d) $\kappa(\mathbb{Z}_5^2, C) = 5$,
$D = (\{4, 6\}, \{-6, 6\}, \{-6, 6\}, \{-6, 6\}, \{-4, -6\})^5$

Figure 3 Symmetric Hamilton cycles in the (a) 4-cube; (b) middle levels of the 5-cube; (c) Johnson graph $J_{7,2}$; (d) abelian Cayley graph $(\mathbb{Z}_5^2, \{(0, 1), (1, 0)\})$.

of G, and then search for a Hamilton cycle that is k-symmetric under the chosen automorphism. Specifically, proving a lower bound of $\kappa(G) \geq k$ amounts to finding an automorphism f of G and a k-symmetric Hamilton cycle under f. To prove an upper bound of $\kappa(G) < k$, we need to argue that there is no k-symmetric Hamilton cycle in G, for any choice of f.

By what we said in the beginning, the quantity $\kappa(G)$ can be seen as a measure for the nicest (i.e., most symmetric) way of drawing the graph G on a circle. Thus, our paper contains many illustrations that convey the aesthetic appeal of this problem.

1.4 Easy observations and bounds

We collect a few basic observations about the quantity $\kappa(G)$. Trivially, we have $0 \leq \kappa(G) \leq n$, where n is the number of vertices of G. The upper bound n can be improved to

$$\kappa(G) \leq \max_{f \in \mathrm{Aut}(G)} \mathrm{ord}(f), \tag{3}$$

where $\mathrm{Aut}(G)$ is the automorphism group of G, and $\mathrm{ord}(f)$ is the order of f. An immediate consequence of (1) is that all orbits of the automorphism f must have the same size n/k, and the path $P = (x_1, \ldots, x_{n/k})$ visits every orbit exactly once. This can be used to improve (3) further by restricting the maximization to automorphisms from $\mathrm{Aut}(G)$ whose orbits all have the same size. Furthermore, as k must divide n, we obtain that $\kappa(G) \in \{0, 1, n\}$ for prime n.

Clearly, every Hamilton cycle of a graph G is 1-symmetric, by taking the identity mapping $f = \mathrm{id}$ as automorphism. Consequently, we have $\kappa(G) \geq 1$ for any Hamiltonian graph. On the other hand, if G is Hamiltonian and highly symmetric, i.e., if it has a rich automorphism group, then intuitively G should have a large value of $\kappa(G)$, i.e., it should admit highly symmetric Hamilton cycles. For example, for the cycle C_n on n vertices and the complete graph K_n on n vertices we have $\kappa(C_n) = \kappa(K_n) = n$. More generally, note that $\kappa(G) = n$ if and only if G is a special circulant graph, namely the vertices of G can be labeled with $1, \ldots, n$ such that vertex i is adjacent to all vertices $j = i + d$ (modulo n) with $d \in L$, where L is a fixed list with $1 \in L$. Note that general circulant graphs do not require that $1 \in L$, but the aforementioned characterization requires this assumption.

2 Our results

Vertex-transitive graphs are a prime example of highly symmetric graphs. A graph is *vertex-transitive* if for any two vertices there is an automorphism that maps the first vertex to the second one. In other words, the automorphism group of the graph acts transitively on the vertices. In this paper we investigate the Hamilton compression $\kappa(G)$ of four families of vertex-transitive graphs G, namely hypercubes, Johnson graphs, permutahedra, and Cayley graphs of abelian groups. In the following definitions the letter n denotes a graph parameter and not the number of vertices of the graph as in Section 1. The *n-dimensional hypercube* Q_n, or *n-cube* for short, has as vertices all bitstrings of length n, and an edge between any two strings that differ in a single bit; see Figure 3 (a). The *Johnson graph* $J_{n,m}$ has as vertices all bitstrings of length n with fixed Hamming weight m, and an edge between any two strings that differ in a transposition of a 0 and 1; see Figure 3 (c). The *n-permutahedron* Π_n, has as vertices all permutations of $[n] := \{1, \ldots, n\}$, and an edge between any two permutations that differ in an adjacent transposition, i.e., a swap of two neighboring entries of the permutations in one-line notation; see Figure 2. For a group Γ and generating set $S \subseteq \Gamma$, the Cayley graph $G(\Gamma, S)$ has Γ as its vertex set and undirected edges $\{x, y\}$ for all $x, y \in \Gamma$ and $s \in S$ with $y = xs$; see Figure 3 (d). Note that the hypercube is isomorphic to a Cayley graph of the abelian group $\mathbb{Z}_2^n$.

Hamilton cycles with various additional properties in the aforementioned families of graphs have been the subject of a long line of previous research under the name of *combinatorial Gray codes* [22, 25]. We will see that some classical constructions of such cycles have a non-trivial small compression factor, and we construct cycles with much higher compression factor that we show to be optimal or near-optimal. Along the way, many interesting number-theoretic and algebraic phenomena arise. Due to space constraints, in this extended abstract we only mention our main results, while all proofs can be found in the preprint [15].

2.1 Hypercubes

One of the classical constructions of a Hamilton cycle in Q_n is the well-known *binary reflected Gray code (BRGC)* [14]. This cycle in Q_n is defined inductively by $\Gamma_0 := \varepsilon$ and $\Gamma_n := 0\Gamma_{n-1}, 1\overleftarrow{\Gamma}_{n-1}$ for all $n \geq 1$, where ε is the empty sequence and $\overleftarrow{\Gamma}_{n-1}$ denotes the reversal of the sequence Γ_{n-1}. In words, the cycle Γ_n is obtained by concatenating the vertices of Γ_{n-1} prefixed by 0 with the vertices of Γ_{n-1} in reverse order prefixed by 1. The cycle Γ_n is shown in Figure 3 (a) and Figure 4 (a) for $n = 4$ and $n = 8$, and these drawings have 4-fold rotational symmetry.

▶ **Proposition 1.** *The BRGC Γ_n has compression $\kappa(Q_n, \Gamma_n) = 4$ for $n \geq 2$.*

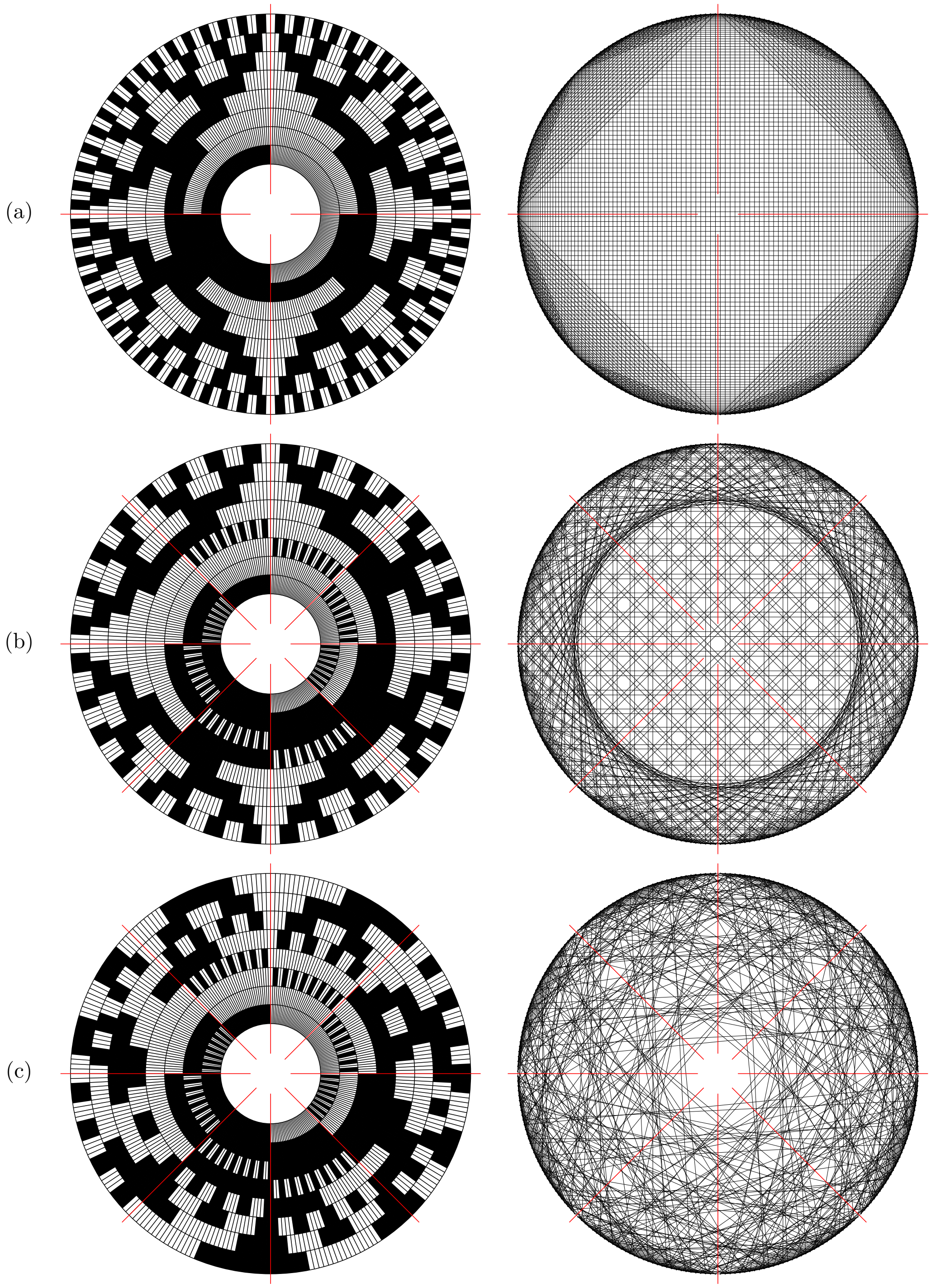


Figure 4 Symmetric Hamilton cycles in Q_8. Cycles are on the left (0=white, 1=black), with the first and last bit on the inner and outer track, respectively. The full graph Q_8 is on the right, with vertices arranged in cycle order and edges drawn as straight lines. (a) Binary reflected Gray code Γ_8 with compression 4; (b) Hamilton cycle with compression 8 from Theorem 2; (c) 2-track Hamilton cycle with compression 8 from Theorem 8.

We improve upon this by constructing new Hamilton cycles in Q_n that have optimal linear Hamilton compression; see Figure 4 (b).

▶ **Theorem 2.** *We have* $\kappa(Q_2) = 4$ *and* $\kappa(Q_n) = 2^{\lceil \log_2 n \rceil}$ *for all* $n \geq 3$.

Note that $n \leq \kappa(Q_n) < 2n$ for $n \geq 2$, in particular $\kappa(Q_n) = \Theta(n)$, i.e., the optimal compression grows linearly with n.

2.2 Johnson graphs and relatives

Our definition of Hamilton compression is inspired by a variant of the well-known middle levels problem raised by Knuth in Problem 56 in Section 7.2.1.3 of his book [17]. Let M_{2n+1} denote the subgraph of Q_{2n+1} induced by all bitstrings with Hamming weight n or $n+1$. In other words, M_{2n+1} is the subgraph of the cover graph of the Boolean lattice of dimension $2n+1$ induced by the middle two levels. There is a natural automorphism of M_{2n+1} all of whose orbits have the same size, namely cyclic left-shift of the bitstrings by one position. Knuth asked whether M_{2n+1} admits a $(2n+1)$-symmetric Hamilton cycle

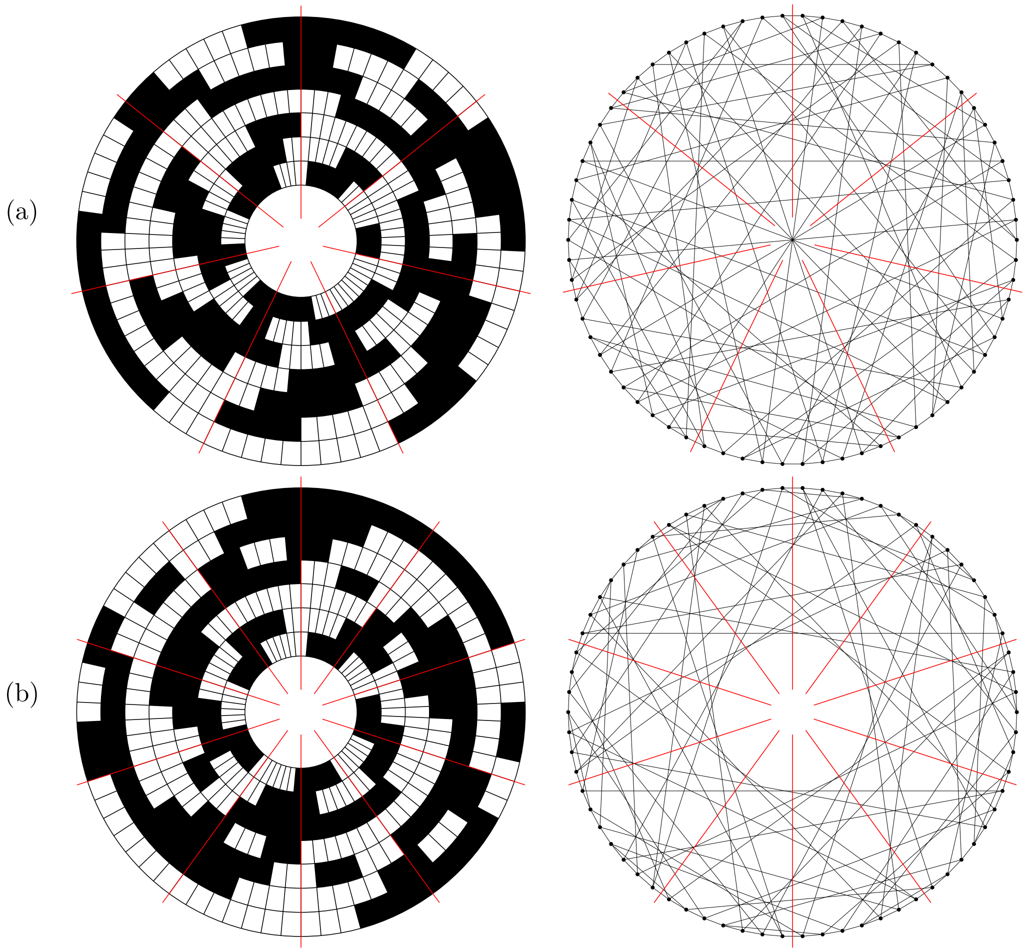


Figure 5 Symmetric Hamilton cycles in the middle levels graph M_7: (a) A solution to Knuth's problem with compression 7 for f being cyclic left-shift; (b) Hamilton cycle with compression 10 for f being left-shift of the last 5 bits and complementation of all bits.

under this automorphism, and he rated this the hardest open problem in his book, with a difficulty rating of 49/50. Such cycles are shown in Figure 3 (b) and Figure 5 (a) for the graphs M_5 and M_7, respectively. Knuth's problem was answered affirmatively in full generality in [21], which establishes the lower bound $\kappa(M_{2n+1}) \geq 2n+1$. We show that this is at most a factor of 2 away from optimality.

▶ **Theorem 3.** *For all $n \geq 1$ we have $2n+1 \leq \kappa(M_{2n+1}) \leq 2(2n+1)$.*

Interestingly, it seems that both bounds in Theorem 3 can be improved. For example, for $n=3$ we can take the automorphism f of M_7 defined by $x_1 \cdots x_7 \mapsto \overline{x_1 x_2 x_4 x_5 x_6 x_7 x_3}$, which fixes the first two bits, cyclically left-shifts the remaining five bits by one position, and then complements all bits. A 10-symmetric Hamilton cycle under this f is shown in Figure 5 (b), whereas the lower and upper bounds are 7 and 14, respectively. In fact, computer experiments show that $\kappa(M_7) = 10$.

For the Johnson graphs $J_{n,m}$, we obtain the following exact results and bounds. Part (i) and (ii) of the theorem are illustrated in Figure 6 (a) and (b), respectively.

▶ **Theorem 4.** *The Hamilton compression of the Johnson graph $J_{n,m}$, where $n > m > 0$, has the following properties:*

(i) *If n and m are coprime, we have $\kappa(J_{n,m}) = n$.*
(ii) *If n and m are not coprime and $n \neq 2m$, we have $n/2 < \max\{m, n-m\} < \kappa(J_{n,m}) \leq n$.*
(iii) *If n and m are not coprime and $n = 2m$, we have $n/2 < \kappa(J_{n,m}) \leq 2n$.*
(iv) *For any $\varepsilon > 0$ there is an n_0 such that for all $n > n_0$ with $n \neq 2m$ we have $(1-\varepsilon)n \leq \kappa(J_{n,m}) \leq n$. In particular, we have $\kappa(J_{n,m}) = (1-o(1))n$ for $n \neq 2m$.*

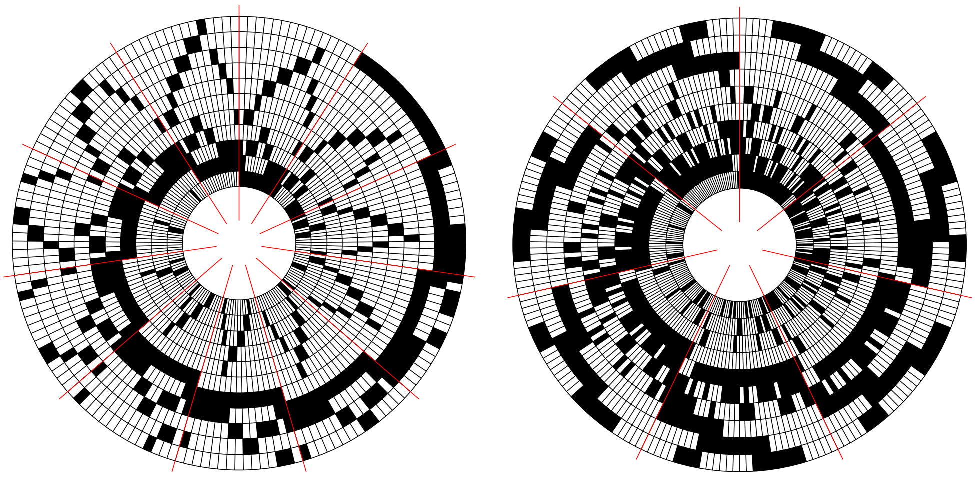


Figure 6 Symmetric Hamilton cycles in Johnson graphs: (a) Balanced 1-track Hamilton cycle in $J_{11,3}$ with compression $n = 11$; the automorphism left-shifts all n bits; (b) 4-track Hamilton cycle in $J_{10,4}$ with compression $q = 7$; the automorphism left-shifts the first q bits.

2.3 Permutahedra

Another classical Gray code is produced by the *Steinhaus-Johnson-Trotter (SJT) algorithm*, which generates permutations by adjacent transpositions. This algorithm computes a Hamilton cycle in Π_n, which can be described inductively as follows: $\Lambda_1 := 1$ and for all

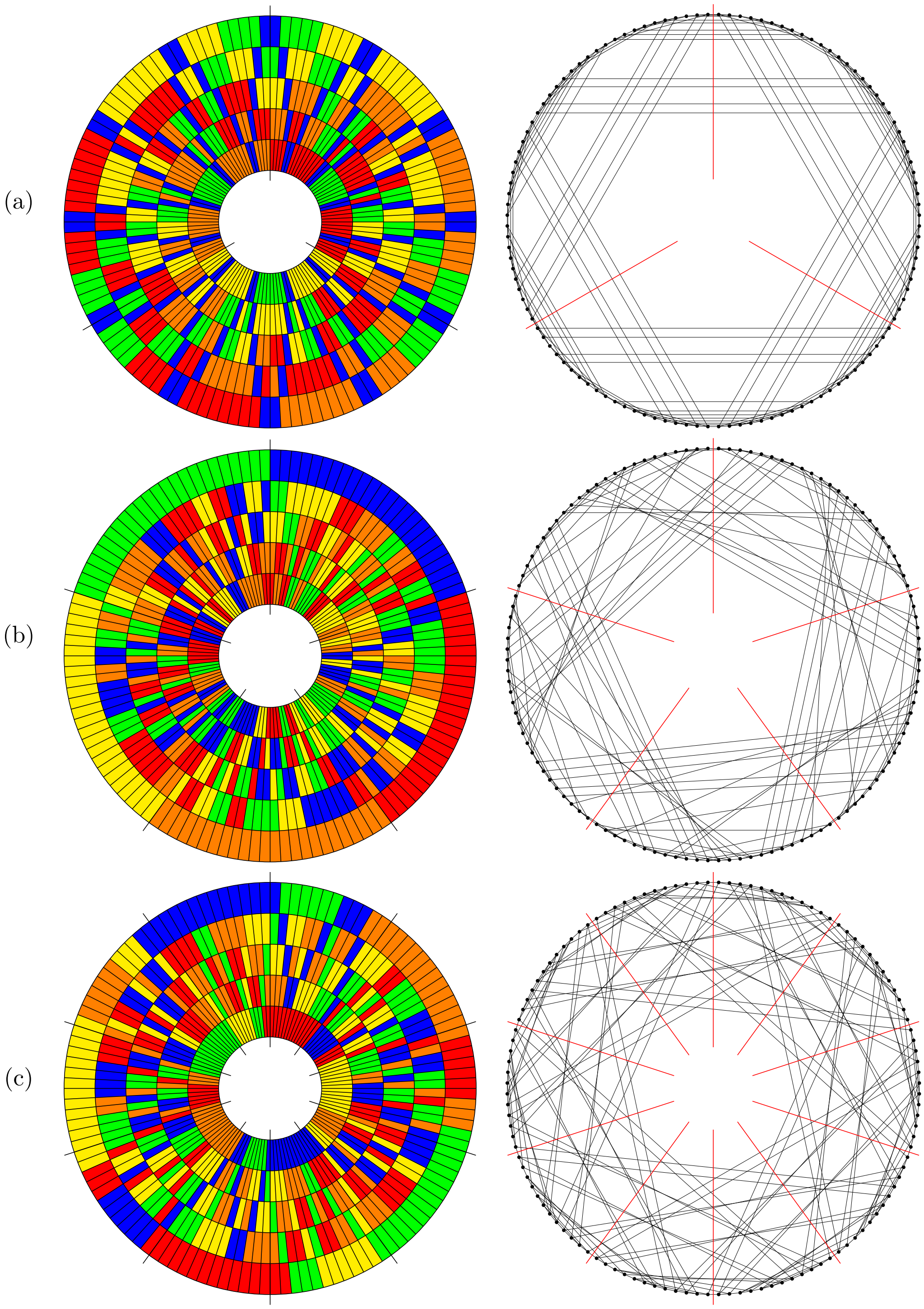


Figure 7 Symmetric Hamilton cycles in Π_5 (1=red, 2=orange, 3=yellow, 4=green, 5=blue): (a) Steinhaus-Johnson-Trotter cycle Λ_5 with compression 3; (b) Cycle with compression 5; (c) Cycle with optimal compression 10.

$n \geq 2$ the cycle Λ_n is obtained from Λ_{n-1} by replacing each permutation of length $n-1$ by the n permutations given by inserting n in every possible position, alternatingly from right to left or vice versa. The cycle Λ_n is shown in Figure 2 (a) and Figure 7 (a) for $n = 4$ and $n = 5$, respectively, and these drawings have 6-fold or 3-fold rotational symmetry.

▶ **Proposition 5.** *The SJT cycle Λ_n has compression $\kappa(\Pi_n, \Lambda_n) = 6$ for $n = 3, 4$ and $\kappa(\Pi_n, \Lambda_n) = 3$ for $n \geq 5$.*

We improve upon this by constructing new Hamilton cycles in Π_n that have mildly exponential Hamilton compression; see Figure 7 (b)+(c). Specifically, the growth of the optimum compression is determined by *Landau's function* $\lambda(n)$, which is defined as the maximum order of an element in the symmetric group S_n.

▶ **Theorem 6.** *We have $\kappa(\Pi_n) = \Theta(\lambda(n)) = e^{(1+o(1))\sqrt{n \ln n}}$.*

The lower and upper bounds in the proof of Theorem 6 differ at most by a factor of 2 for every $n \geq 3$. Moreover, we achieve the optimal compression in infinitely many cases, in particular for the following values of $n \leq 100$: $n = 3, 4, 5, 15, 22, 46, 49, 51, 52, 53, 55, 68, 69, 72, 73, 74, 75, 80, 82, 85, 87, 88, 89, 91, 92, 93, 96, 97, 99, 100$.

2.4 Abelian Cayley graphs

A classical folklore result asserts that every Cayley graph of an abelian group has a Hamilton cycle. The Chen-Quimpo theorem [4] asserts that in fact much stronger Hamiltonicity properties hold. It is thus natural to ask whether Cayley graphs of abelian groups have highly symmetric Hamilton cycles.

▶ **Theorem 7.** *Let Γ be an abelian group.*

(i) *If $|\Gamma|$ is a product of distinct odd primes, then for the canonical generating set $S \subseteq \Gamma$, the Cayley graph $G = G(\Gamma, S)$ has compression $\kappa(G) = 1$.*

(ii) *If $|\Gamma| \geq 3$ is even or divisible by a square greater than 1, then for any generating set $S \subseteq \Gamma$, the Cayley graph $G = G(\Gamma, S)$ has compression $\kappa(G) \geq 2$.*

In particular, toroidal grids $\mathbb{Z}_p \times \mathbb{Z}_q$ for distinct odd primes p, q have only compression 1. The canonical generating set of a finite abelian group Γ is the set of unit vectors in the decomposition of Γ as a product of cyclic groups given by the structure theorem of finite abelian groups.

3 Related problems

We proceed to discuss some applications of our results to closely related problems.

3.1 Lovász' conjecture

A well-known question of Lovász' [20] asks whether there are infinitely many vertex-transitive graphs that do not admit a Hamilton cycle. So far only five such graphs are known, namely K_2, the Petersen graph, the Coxeter graph, and the graphs obtained from the latter two by replacing every vertex by a triangle. Vertex-transitive graphs have a lot of automorphisms, and we may take the quantity $\kappa(G)$ as a measure of how strongly G is Hamiltonian. In particular, Lovász' question may be rephrased as 'Are there infinitely many vertex-transitive graphs G with $\kappa(G) = 0$?' More generally, we may ask: 'Are there infinitely many vertex-transitive graphs G with $\kappa(G) = k$, for each fixed integer k?' A particularly relevant subclass

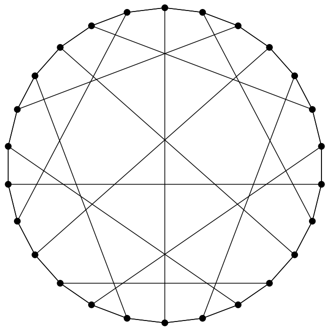


Figure 8 One of the smallest vertex-transitive non-Cayley graphs G with $\kappa(G) = 1$.

of vertex-transitive graphs are Cayley graphs, so we may ask the same question about Cayley graphs. From our results mentioned in Section 2.4 we obtain an infinite family of Cayley graphs G with $\kappa(G) = 1$. Computer experiments show that the smallest vertex-transitive non-Cayley graphs G with $\kappa(G) = 1$ have 26 vertices, and one of them is shown in Figure 8.

The path P in (1) is a Hamilton path in the quotient graph G/f obtained by collapsing each orbit of f into a single vertex. The idea of constructing a Hamilton cycle in G by constructing a Hamilton cycle in the much smaller graph G/f that is then "lifted" to the full graph is well known in the literature, and has been used to solve some special cases of Lovász' problem affirmatively; see e.g. [1, 6, 19, 21, 27]. It is particularly useful for computer searches, as it reduces the search space dramatically.

3.2 *t*-track and balanced Gray codes

We say that a sequence C of strings of length n consists of t *tracks* if in the $|C| \times n$ matrix corresponding to C there are t columns such that every other column is a cyclically shifted copy of one of these columns. This property is relevant for applications, as it saves hardware when implementing Gray-coded rotary encoders. Instead of using n tracks and n reading heads aligned at the same angle (each reading one track), one can use only t tracks, and place some of the n reading heads at appropriately rotated positions.

Hiltgen, Paterson, and Brandestini [16] showed that the length of any 1-track cycle in Q_n must be a multiple of $2n$. In particular, such a cycle cannot be a Hamilton cycle unless n is a power of 2. For the case $n = 2^r$, $r \geq 3$, Etzion and Paterson [7] showed that there is 1-track cycle of length $2^n - 2n$, and Schwartz and Etzion [26] subsequently showed that the length $2^n - 2n$ is best possible. Taken together, these results show that there is no 1-track Hamilton cycle in Q_n for any $n \geq 3$. We complement this negative result by constructing a 2-track Hamilton cycle in Q_n, for every n that is a sum of two powers of 2; see Figure 4 (c).

▶ **Theorem 8.** *For every $n = 2^r$ and $m = 2^s$, where $r \geq 2$ and $r \geq s \geq 0$, there is a $2n$-symmetric Hamilton cycle in Q_{n+m} that has 2 tracks.*

More generally, we obtain t-track Hamilton cycles in Q_n for every n that is a sum of $t \geq 2$ powers of 2. In particular, every dimension n admits a Hamilton cycle with at most logarithmically many tracks.

▶ **Theorem 9.** *For every $n = 2^r$ and $(m_1, \ldots, m_{t-1}) = (2^{s_1}, \ldots, 2^{s_{t-1}})$, where $r, t \geq 2$ and $r \geq s_1 \geq \cdots \geq s_{t-1} \geq 0$, there is a $2n$-symmetric Hamilton cycle in $Q_{n+m_1+\cdots+m_{t-1}}$ that has t tracks.*

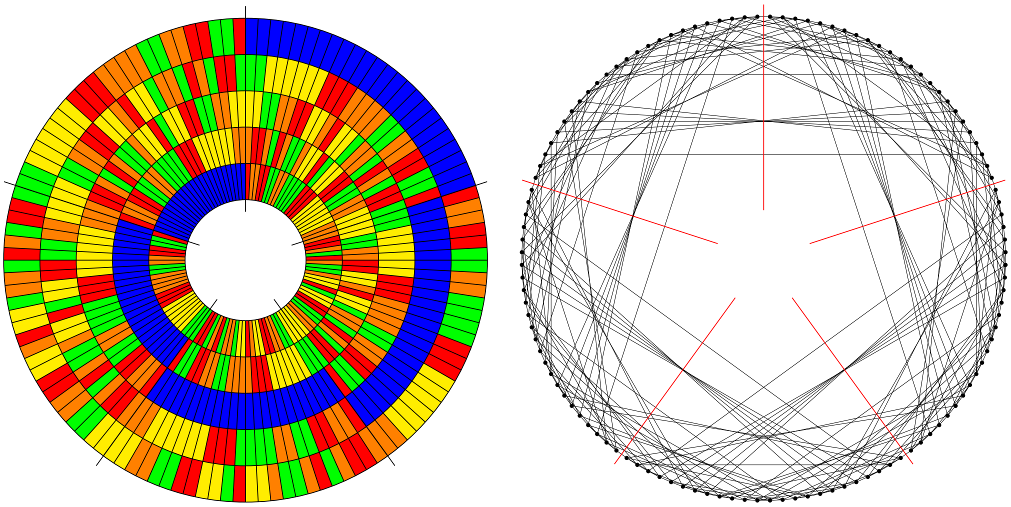


Figure 9 Balanced 1-track Hamilton cycle in Π_5^+ with compression 5 from Theorem 11 (cyclically adjacent transpositions).

From our construction in the Johnson graph $J_{n,m}$ when n and m are coprime, we obtain 1-track Hamilton cycles that are also *balanced*, i.e., each bit is flipped equally often (cf. [3, 8]); see Figure 6 (a).

▶ **Theorem 10.** *Let $n > m > 0$ be such that n and m are coprime. Then $J_{n,m}$ has an n-symmetric Hamilton cycle that has 1 track and is balanced, i.e., each bit is flipped equally often ($\binom{n}{m}/n$ many times).*

We write Π_n^+ for the graph obtained from the permutahedron Π_n by adding edges that correspond to transpositions of the first and last entry of a permutation, i.e., we allow cyclically adjacent transpositions. The next theorem is illustrated in Figure 9.

▶ **Theorem 11.** *For every odd $n \geq 3$ there is an n-symmetric Hamilton cycle in Π_n^+ that has 1 track and is balanced, i.e., each of the n transpositions is used equally often ($(n-1)!$ many times).*

4 Open questions

The Hamilton compression $\kappa(G)$ is a newly introduced graph parameter, so many natural follow-up questions arise. We conclude this paper by listing several of these problems.

- Can the Gray codes constructed in this paper be computed efficiently? While our proofs translate straightforwardly into algorithms whose running time is polynomial in the size of the graph, a more ambitious goal are algorithms whose running time per generated vertex is polynomial in the length of the vertex labels (bitstrings, permutations, etc.).
- What is the Hamilton compression of the middle levels graph (recall Theorem 3)?
- For any integer $n \geq 1$, the odd graph O_n has as vertices all bitstrings of length $2n+1$ with Hamming weight n, and an edge between any two strings that have no 1s in common. Odd graphs O_n, $n \geq 3$, were shown to have a Hamilton cycle in [23], so $\kappa(O_n) \geq 1$. We can use cyclic shifts as the automorphism, and it is easy to see that $\kappa(O_n) \leq 2n+1$. We conjecture that $\kappa(O_n) = 2n+1$ for all $n \geq 4$, which we confirmed for $n = 4$.

- In view of Section 2.4, the main open question here is whether the Cayley graph $G = G(\Gamma, S)$, where Γ is an abelian group such that $|\Gamma|$ is a product of distinct odd primes and S is a non-canonical set of generators, has $\kappa(G)$ equal to 1 or exceeding 1.
- What is the Hamilton compression of the associahedron, which has as automorphism group the dihedral group of a regular n-gon? For $n = 5, 6, 7, 8$ we determined the values $5, 2, 7, 2$ by computer, and we suspect that the primality of n plays a role.
- Instead of asking about the largest number $k = \kappa(G)$ such that $\mathrm{Aut}(G, C)$ (automorphisms of G that preserve C) contains the cyclic subgroup of order k for some Hamilton cycle C in G, we may ask for the dihedral subgroup of the largest order, which would allow not only for rotations of the drawings but also reflections.
- Is there a 1-track Hamilton cycle in Π_n (recall Theorem 11)? Equivalently, can all $n!$ permutations be listed by adjacent transpositions so that every column is a cyclic shift of every other column?
- Is there a balanced Hamilton cycle in Π_n? Equivalently, can all $n!$ permutations be listed using each of the $n - 1$ adjacent transpositions equally often? Alternatively, what about using each of the $\binom{n}{2}$ transpositions equally often (see [8])? For $n = 5$, we found orderings satisfying the constraints of both questions.

References

1 B. Alspach. Lifting Hamilton cycles of quotient graphs. *Discrete Math.*, 78(1-2):25–36, 1989. doi:10.1016/0012-365X(89)90157-X.

2 D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

3 G. S. Bhat and C. D. Savage. Balanced Gray codes. *Electron. J. Combin.*, 3(1):Paper 25, 11 pp., 1996. URL: http://www.combinatorics.org/Volume_3/Abstracts/v3i1r25.html.

4 C. C. Chen and N. F. Quimpo. On strongly Hamiltonian abelian group graphs. In *Combinatorial mathematics, VIII (Geelong, 1980)*, volume 884 of *Lecture Notes in Math.*, pages 23–34. Springer, Berlin-New York, 1981.

5 S. J. Curran and J. A. Gallian. Hamiltonian cycles and paths in Cayley graphs and digraphs – A survey. *Discrete Math.*, 156(1-3):1–18, 1996. doi:10.1016/0012-365X(95)00072-5.

6 S. Du, K. Kutnar, and D. Marušič. Resolving the Hamiltonian problem for vertex-transitive graphs of order a product of two primes. *Combinatorica*, 41(4):507–543, 2021. doi:10.1007/s00493-020-4384-6.

7 T. Etzion and K. G. Paterson. Near optimal single-track Gray codes. *IEEE Trans. Inform. Theory*, 42(3):779–789, 1996. doi:10.1109/18.490544.

8 S. Felsner, L. Kleist, T. Mütze, and L. Sering. Rainbow cycles in flip graphs. *SIAM J. Discrete Math.*, 34(1):1–39, 2020. doi:10.1137/18M1216456.

9 R. Frucht. A canonical representation of trivalent Hamiltonian graphs. *J. Graph Theory*, 1(1):45–60, 1977. doi:10.1002/jgt.3190010111.

10 M. R. Garey and D. S. Johnson. *Computers and intractability.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness.

11 R. J. Gould. Updating the Hamiltonian problem – A survey. *J. Graph Theory*, 15(2):121–157, 1991. doi:10.1002/jgt.3190150204.

12 R. J. Gould. Advances on the Hamiltonian problem – A survey. *Graphs Combin.*, 19(1):7–52, 2003. doi:10.1007/s00373-002-0492-x.

13 R. J. Gould. Recent advances on the Hamiltonian problem: Survey III. *Graphs Combin.*, 30(1):1–46, 2014. doi:10.1007/s00373-013-1377-x.

14 F. Gray. Pulse code communication, 1953. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.

15 P. Gregor, A. Merino, and T. Mütze. The Hamilton compression of highly symmetric graphs. Full preprint version of the present article, 2022. arXiv:2205.08126.

16 A. P. Hiltgen, K. G. Paterson, and M. Brandestini. Single-track gray codes. *IEEE Trans. Inf. Theory*, 42(5):1555–1561, 1996. doi:10.1109/18.532900.

17 D. E. Knuth. *The Art of Computer Programming. Vol. 4A. Combinatorial Algorithms. Part 1.* Addison-Wesley, Upper Saddle River, NJ, 2011.

18 D. Kühn and D. Osthus. A survey on Hamilton cycles in directed graphs. *European J. Combin.*, 33(5):750–766, 2012. doi:10.1016/j.ejc.2011.09.030.

19 K. Kutnar and D. Marušič. Hamilton cycles and paths in vertex-transitive graphs – Current directions. *Discrete Math.*, 309(17):5491–5500, 2009. doi:10.1016/j.disc.2009.02.017.

20 L. Lovász. Problem 11. In *Combinatorial Structures and Their Applications (Proc. Calgary Internat. Conf., Calgary, AB, 1969).* Gordon and Breach, New York, 1970.

21 A. I. Merino, O. Mička, and T. Mütze. On a combinatorial generation problem of Knuth. In D. Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 735–743. SIAM, 2021. doi:10.1137/1.9781611976465.46.

22 T. Mütze. Combinatorial Gray codes – An updated survey, 2022. arXiv:2202.01280.

23 T. Mütze, J. Nummenpalo, and B. Walczak. Sparse Kneser graphs are Hamiltonian. *J. Lond. Math. Soc. (2)*, 103(4):1253–1275, 2021. doi:10.1112/jlms.12406.

24 I. Pak and R. Radoičić. Hamiltonian paths in Cayley graphs. *Discrete Math.*, 309(17):5501–5508, 2009. doi:10.1016/j.disc.2009.02.018.

25 C. D. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295272.

26 M. Schwartz and T. Etzion. The structure of single-track Gray codes. *IEEE Trans. Inform. Theory*, 45(7):2383–2396, 1999. doi:10.1109/18.796379.

27 I. Shields, B. Shields, and C. D. Savage. An update on the middle levels problem. *Discrete Math.*, 309(17):5271–5277, 2009. doi:10.1016/j.disc.2007.11.010.

28 D. Witte and J. A. Gallian. A survey: Hamiltonian cycles in Cayley graphs. *Discrete Math.*, 51(3):293–304, 1984. doi:10.1016/0012-365X(84)90010-4.

Dispersing Obnoxious Facilities on Graphs by Rounding Distances

Tim A. Hartmann
Department of Computer Science, RWTH Aachen University, Germany

Stefan Lendl
Department of Operations and Information Systems, University of Graz, Austria

Abstract

We continue the study of δ-dispersion, a continuous facility location problem on a graph where all edges have unit length and where the facilities may also be positioned in the interior of the edges. The goal is to position as many facilities as possible subject to the condition that every two facilities have distance at least δ from each other.

Our main technical contribution is an efficient procedure to "round-up" distance δ. It transforms a δ-dispersed set S into a $\delta^\star$-dispersed set $S^\star$ of same size where distance $\delta^\star$ is a potentially slightly larger rational $\frac{a}{b}$ with a numerator a upper bounded by the longest (not-induced) path in the input graph.

Based on this rounding procedure and connections to the distance-d independent set problem we derive a number of algorithmic results. When parameterized by treewidth, the problem is in XP. When parameterized by treedepth the problem is FPT and has a matching lower bound on its time complexity under ETH. Moreover, we can also settle the parameterized complexity with the solution size as parameter using our rounding technique: δ-Dispersion is FPT for every $\delta \leq 2$ and W[1]-hard for every $\delta > 2$.

Further, we show that δ-dispersion is NP-complete for every fixed irrational distance δ, which was left open in a previous work.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases facility location, parameterized complexity, packing

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.55

Related Version *Full Version*: https://arxiv.org/abs/2206.11337

1 Introduction

We study the algorithmic behavior of a continuous dispersion problem. Consider an undirected graph G, whose edges have unit length. Let $P(G)$ be the continuum set of points on all the edges and vertices. For two points $p, q \in P(G)$, we denote by $d(p, q)$ the length of a shortest path containing p and q in the underlying metric space. A subset $S \subseteq P(G)$ is *δ-dispersed* for some positive real number δ, if every distinct points $p, q \in S$ have distance at least $d(p, q) \geq \delta$. Our goal is, for a given graph G and a positive real number δ, to compute a maximum cardinality subset $S \subseteq P(G)$ that is δ-dispersed. We denote by δ-*disp*(G) the maximum size of a δ-dispersed set of G. The decision problem Dispersion asks for a δ-dispersed set of size at least k, where additionally integer $k \geq 0$ is part of the input. When δ is fixed and not part of the input, we refer to the problem as δ-Dispersion.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 55; pp. 55:1–55:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Known and Related Results

The area of obnoxious facility location goes back to seminal articles of Goldman & Dearing [6] and Church & Garfinkel [3]. The area includes a wide variety of objectives and models. For example, purely geometric variants have been studied by Abravaya & Segal [1], Ben-Moshe, Katz & Segal [2], and Katz, Kedem & Segal [14]. Recently, van Ee studied the approximability of a generalized covering problem in a metric space that also involves dispersion constraints [19]. Another direction is a graph-theoretic model, where every edge of the given graph G is rectifiable and has some individual length. Tamir discusses the complexity and approximability of several optimization problems. For example, when G is a tree, then a δ-dispersed set can be computed in polynomial time [18]. Another task is to place a single obnoxious facility in a network while maximizing, for example, the smallest distance from the facility to certain clients, as studied by Segal [16].

In a previous work, the complexity of Dispersion was studied for every rational distance δ. When δ is a rational number with numerator 1 or 2, the problem is polynomial time solvable, while it is NP-complete for all other rational values of δ [7, 8]. The complexity when δ is irrational was left as an open problem.

A closely related facility location problem is δ-covering. The objective is to place as few locations as possible on $P(G)$ subject to the condition that any point in $P(G)$ is in distance at most δ to a placed location. This problem is polynomial time solvable whenever δ is a unit fraction, while it is NP-hard for all non unit fractions δ [10]. Furthermore, the parameterized complexity with the parameter solution size k is studied. δ-covering is fixed parameter tractable when $\delta < \frac{3}{2}$, while for $\delta \geq \frac{3}{2}$ the problem is W[2]-complete [10]. Tamir [17] showed that for δ-covering only certain distances δ are of interest. For every amount of points p the distance $\max\{\delta^\star : |\delta^\star\text{-cover}(G)| = p\}$ is of the form $\frac{L'}{2p'}$ where $p' \in \{1, \ldots, p\}$ and L' is roughly at most twice the length of a non-induced path in G.

1.2 Our Contribution

Our main technical contribution is an efficient and constructive rounding procedure. Given a δ-dispersed set S for some distance value $\delta > 0$, it transforms S into a $\delta^\star$-dispersed set $S^\star$ of equal size with a slightly larger well-behaving distance value $\delta^\star \geq \delta$. The new distance $\delta^\star$ is a rational $\frac{a}{b}$ with small numerator a. More precisely, the numerator is upper bounded by the length of the longest (not-induced) path L, hence upper bounded asymptotically by the number of vertices n of the input graph (see Section 5).

Our second technical contribution relates the optimal solution for distance δ and $\frac{\delta}{\delta+1}$ for $\delta \leq 3$. A δ-dispersed set translates to a $\frac{\delta}{\delta+1}$-dispersed set by placing one more point on every edge, and vice versa by removing one point (see Section 3).

Further we explore a connection of Dispersion and an independent set problem (see Section 4). The combination of that connection with our technical contributions yields several algorithmic results for Dispersion (see Section 6 and Section 7):

- Dispersion is NP-hard even for chordal graphs of diameter 4.
- Dispersion is FPT for the graph parameter treedepth $\mathsf{td}(G)$ with a run time matching a lower bound under ETH. We complement this result by showing that δ-Dispersion is W[1]-hard for the slightly more general graph parameter pathwidth $\mathsf{pw}(G)$, even for the combined parameter $\mathsf{pw}(G) + k$. Similarly, δ-Dispersion is W[1]-hard for the graph parameter $\mathsf{fvs}(G)$, the minimum size of a feedback vertex set.
- Dispersion is XP for the parameter treewidth $\mathsf{tw}(G)$, with a running time of $(2L)^{\mathsf{tw}(G)} n^{\mathcal{O}(1)}$, where n is the number of vertices and L is an upper bound on the length

of the longest path in G. We complement this result by the more general lower bound of $n^{o(\mathsf{tw}(G)+\sqrt{k})}$, assuming ETH. It implies the lower bound of $L^{o(\mathsf{tw}(G)+\sqrt{k})}$ since $L \leq n$. Note that a mere lower bound of $L^{o(\mathsf{tw}(G)+\sqrt{k})}$ would not exclude an $n^{o(\mathsf{tw}(G))}$-algorithm.

In addition, we completely resolve the complexity of δ-dispersion, by showing NP-hardness for irrational δ (see Section 8). We also study the parameterized complexity when parameterized by the solution size k. The problem is W[1]-hard when $\delta > 2$, and FPT otherwise. Thus, there is a sharp threshold at $\delta = 2$ where the complexity jumps from FPT to W[1]-hard (see Section 9).

We mark statements whose proof can be found in the full version of the paper (see [9]) with "($\star$)".

2 Preliminaries

We use the word *vertex* in the graph-theoretic sense, while we use the word *point* to denote the elements of the geometric structure $P(G)$. As an input for δ-dispersion, we consider graphs G that are undirected, connected, and without loops and isolated vertices.

For an edge $\{u, v\} \in E(G)$ and a real number $\lambda \in [0, 1]$, let $p(u, v, \lambda) \in P(G)$ be the point on edge $\{u, v\}$ that has distance λ from u. Note that $p(u, v, 0) = u$, $p(u, v, 1) = v$ and $p(u, v, \lambda) = p(v, u, 1 - \lambda)$. Further, we use $d(p, q)$ for the length of a shortest path between points $p, q \in P(G)$.

For a subset of vertices $V' \subseteq V(G)$ or a subset of edges $E' \subseteq E(G)$, we denote by $G[V']$ and $G[E']$ the subgraph induced by V' and E', respectively. The neighborhood of a vertex u is $N(u) := \{v \in V(G) \mid \{u, v\} \in E(G)\}$. We use n as the number of vertices of G, when G is clear from the context.

For a graph G and integer $c \geq 1$, let the *c-subdivision of G* be the graph G where every edge is replaced by a path of length c.

▶ **Lemma 1** ([8])**.** *Let G be a graph, let $c \geq 1$ be an integer, and let G' be the c-subdivision of G. Then δ-disp$(G) = (c\delta)$-disp(G').*

For integers a and b, we denote the rational number $\frac{a}{b}$ as *b-simple.* A set $S \subseteq P(G)$ is b-simple, if for every point $p(u, v, \lambda)$ in S the edge position λ is *b-simple.*

▶ **Lemma 2** ([8])**.** *Let $\delta = \frac{a}{b}$ with integers a and b, and let G be a graph. Then, there exists an optimal δ-dispersed set $S^\star$ that is $2b$-simple.*

For an introduction into parameterized algorithms, we refer to [4]. We study of the complexity of Dispersion with the natural parameter solution size k, as well as its dependency on structural measures on the input graph. Besides treewidth $\mathsf{tw}(G)$ and pathwidth $\mathsf{pw}(G)$, we also study the parameters "feedback vertex set size" $\mathsf{fvs}(G)$ and treedepth $\mathsf{td}(G)$.

A graph has a feedback vertex set $W \subseteq V(G)$ if G after removing W contains no cycle. The "feedback vertex set size" is the size of a smallest feedback vertex set of G.

The treedepth of a connected graph G can be defined as follows. If G is disconnected, it is the maximum treedepth of its components; If G consists of a single vertex, then $\mathsf{td}(G) = 1$; And else it is one plus the minimum over all $u \in V(G)$ of the treedepth of G without vertex u.

We provide lower bounds for the time-complexity assuming the Exponential Time Hypothesis (ETH): There is no $2^{o(N)}$-time algorithm for 3-SAT with N variables and $\mathcal{O}(N)$ clauses [11]. For more details on ETH, we refer to [4].

3 Translating δ-Dispersion

There is an intriguing relation of the optimal solution for distance δ and $\frac{\delta}{2\delta+1}$ for the similar problem δ-covering [10]. We may analogously expect that an optimal solution for δ-dispersion translates to an optimal solution for $\frac{\delta}{\delta+1}$-dispersion; i.e., that an optimal δ-dispersed set corresponds to an optimal $\frac{\delta}{\delta+1}$-dispersed set of the same size plus one extra point for every edge.

This is not true for $\delta = 3+\varepsilon$ for any $\varepsilon > 0$: Consider a triangle, where a $(3+\varepsilon)$-dispersed set S contains at most one point p. Since $\frac{\delta}{\delta+1} > \frac{3}{4}$, a $\frac{\delta}{\delta+1}$-dispersed set however contains at most $3 < |S| + 3$ points.

Causing trouble is a non-trivial closed walk containing p of length less than δ. The translating lemma may only apply to a variation of dispersion that is sensitive to such walks, a variant which we call *auto-dispersion.* A δ-dispersed set $S \subseteq P(G)$ is δ*-auto-dispersed* if additionally for every point $p \in S$ there is no walk from p to p of length $< \delta$ that is locally-injective. A walk is *locally-injective* if, when interpreted as a continuous mapping $f : [0,1] \to P(G)$ from $f(0) = p$ to $f(1) = p$, has for every pre-image $c \in (0,1)$ a positive range $\varepsilon > 0$ such that f restricted to the interval $(c-\varepsilon, c+\varepsilon)$ is injective.

▶ **Lemma 3.** ($\star$) *Let G be a graph and $\delta > 0$. Then δ-auto-disp$(G) = \frac{\delta}{\delta+1}$-auto-disp$(G) + |E(G)|$.*

Fortunately, this translation lemma is still useful for ordinary δ-dispersion. We have δ-auto-disp$(G) = \delta$-disp(G) for $\delta \leq 3$, since there is no such locally-injective walk of length < 3. The threshold of 3 is tight according to the above example with graph K_3.

▶ **Corollary 4.** *Let G be a graph and $\delta \in (0,3]$. Then δ-disp$(G) = \frac{\delta}{\delta+1}$-disp$(G) + |E(G)|$.*

4 Dispersion and Independent Set

To solve Dispersion we can borrow from algorithmic results from a generalized independent set problem. A classical independent set is a set of *vertices* where each two elements have to be at least 2 apart from each other (when we consider that the edges have unit length). In a 2-dispersed set also each two elements need to be at least 2 apart from each other, though the set contains a set of *points* of the graph.

To generalize the independent set problem, we may ask that the vertices are not 2 apart but some integer d apart from each other. Such a generalization for independent set is called a *distance-d independent* set or *d-scattered set.* They have been studied by Eto et al. [5] and Katsikarelis et al. [13].

Let $\alpha_d(G)$ be the maximum size of a distance-d independent set, for a graph G and integer d. We relate δ-dispersion to α_d. We consider the *c-subdivision of a graph G*, denoted as G_c, which is the graph G where every edge is replaced by a path of length c, for some integer $c \geq 1$.

▶ **Lemma 5.** *Consider integers a, b and a $2b$-subdivision G_{2b} of a graph G. Then $\frac{a}{b}$-disp$(G) = \alpha_{2a}(G_{2b})$.*

Proof. Consider the b-subdivision G_b of G. Then G_{2b} is a 2-subdivision of G_b. We know that $\frac{a}{b}$-disp$(G) = 2a$-disp(G_{2b}) from Lemma 1. Hence it remains to show $2a$-disp$(G_{2b}) = \alpha_{2a}(G_{2b})$.

Clearly, a distance-$2a$ independent set $I \subseteq V(G)$ is also a $2a$-dispersed set. For the reverse direction, assume there is a $2a$-dispersed set I_{2a} of G_{2b}. Then I_{2a} corresponds to an a-dispersed set I of G_b of same size, according to Lemma 1. Since a is integer, we may assume that S contains only half-integral points, hence points with edge position from $\{0, \frac{1}{2}, 1\}$, according to Lemma 2. Let G_{2b} result from G_b by replacing each edge $\{u, v\}$ by a path $uw_{u,v}v$. Then let $I \subseteq V(G_{2b})$ consist of vertex $u \in V(G)$ with a point in S and every $w_{u,v}$ for every point $p(u, v, \frac{1}{2}) \in S$. Then I is a distance-$2a$ independent set of G_{2b} of size $|I| = |S|$. ◀

Thus to solve Dispersion for $\delta = \frac{a}{b}$ we can use algorithms for distance-d independent set. For rationals $\frac{a}{b}$ with small values of a and b this possibly leads to efficient algorithms. For example, a distance-d independent set on graphs width treewidth $\mathsf{tw}(G)$ (and a given tree decomposition) can be found in time $d^{\mathsf{tw}(G)} n^{\mathcal{O}(1)}$, see [13]. "Simply" subdivide the edges of the input graph sufficiently often, which does not increase the treewidth of the considered graph. To find a $\frac{a}{b}$-dispersed set in a graph G, we can search for distance $2a$ independent set the $2b$-subdivision of G.

▶ **Corollary 6.** *There is an algorithm that, given a rational distance $\frac{a}{b} > 0$ and a graph G, a tree decomposition of width $\mathsf{tw}(G)$, computes a maximum $\frac{a}{b}$-dispersed set S in time $(2a)^{\mathsf{tw}(G)}(bn)^{\mathcal{O}(1)}$.*

However, in general this constitutes a possibly exponential increase of the input size. While in the input of $\frac{a}{b}$-dispersion encodes a and b in binary, the subdivided graph essentially encodes b in unary. Further, if δ is irrational, we do not have a suitable subdivision at all.

5 Rounding the Distance

For a given graph G and distance δ, we state a rational $\delta^\star \geq \delta$ such that δ-disp(G) = $\delta^\star$-disp(G). Our proof is constructive. We give a procedure that efficiently transforms a δ-dispersed set into a $\delta^\star$-dispersed set. The guaranteed rational $\delta^\star$ has a numerator bounded by the longest path in G (or just n as an upper bound thereof). It is independent of the precise structure of the given graph.

To give some intuition: Generally there is some leeway for δ. For example, in a star $K_{1,k}$, $k \geq 1$ for every $\delta \in (1, 2]$ the optimal solution puts a point on every leaf yielding a δ-dispersed set of size k. Hence for instance $\frac{3}{2}$-disp$(K_{1,k})$ = 2-disp$(K_{1,k})$. However, for $\delta > 2$ only one point can be placed, such that 2-disp$(K_{1,k}) \neq (2 + \varepsilon)$-disp$(K_{1,k})$ for every $\varepsilon > 0$.

So what $\delta^\star$ can be guaranteed such that δ-disp(G) = $\delta^\star$-disp(G)? An illustrative example is a path of length 6. Then $\frac{15}{11}$-disp$(G) = 5 = \frac{3}{2}$-disp(G). For $\delta = \frac{15}{11}$ tightly packing 5 points allows to have a space of size $\frac{6}{11}$ at either end of the path, not enough to place another point. However, placing 5 points in distance $\delta = \frac{3}{2}$ allows no leeway; δ is (already) a divisor of 6, the length of the considered path. Distance $\delta^\star$ relies on L, the length of the longest (not-induced) path in G. We have to take into account that δ might divide any path of length $\leq L$. Our $\delta^\star$ is the smallest rational $\frac{a^\star}{b^\star}$ where the numerator $a^\star \leq 2L$. In other words, the inverse of $\delta^\star$ is the next smaller rational number of the inverse of δ in the Farey sequence of order $2L$.

▶ **Theorem 7.** *Let $\delta \in \mathbb{R}^+$. Let L be an upper bound on the length of paths in G. Let $\delta^\star = \frac{a^\star}{b^\star} \geq \delta$ minimal with $a^\star \leq 2L$ and $b^\star \in \mathbb{N}$. Then δ-disp(G) = $\delta^\star$-disp(G).*

Clearly, a $\delta^\star$-dispersed set $S^\star$, is also δ-dispersed, since $\delta^\star \geq \delta$. We have to show the reverse direction. Consider a δ-dispersed set S (of size $|S| \geq 2$) of a connected graph G that is not $\delta^\star$-dispersed, hence δ is irrational or is equal to $\frac{a}{b}$ for some co-prime a, b with $a > 2L$.

In the following we develop our rounding procedure that shows the reverse direction. Our presentation aims to be accessible by starting from the core algorithmic idea from which we unravel all involved technical concepts piece by piece. The detailed proofs are placed in the appendix.

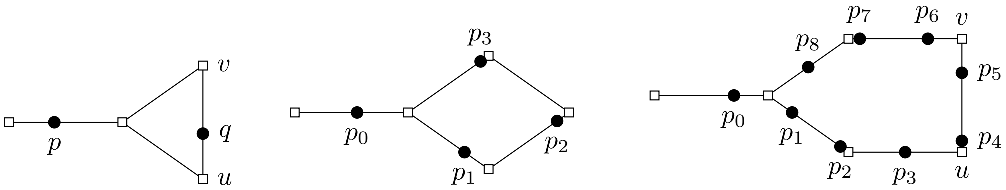


Figure 1 (left) Consider critical points $\{p,q\}$ (points depicted as black dots; vertices as white squares). If we move q away from p by $\varepsilon \geq 0$, their distance increases by ε until q reaches the half-integral point $p(u,v,\frac{1}{2})$. (middle) Let $\{p_i,p_{i-1}\}$ be critical for $i \geq 1$. Consider moving $p_i, i \geq 1$ by $i\varepsilon$ away from p_{i-1}. Once p_2 becomes half-integral, points $\{p_3,p_0\}$ become also critical, hence we cannot continue to move points in the same way. This happens when a point in S becomes half-integral or … (right) … a point half-way between two points in S becomes half-integral, as in this example between p_4,p_5. We say p_4,p_5 *witness* the *pivot* $p(u,v,\frac{1}{2})$.

5.1 Overview

Our rounding procedure repeatedly applies a pushing algorithm to the current point set S. We show that each such step strictly decreases a polynomially bounded potential $\Phi : P(G) \to \mathbb{N}$.

▶ **Theorem 8.** *Suppose that there is an algorithm, that given a δ-dispersed set S with $\delta < \delta^\star$ computes an $\varepsilon > 0$ and a $(\delta+\varepsilon)$-dispersed set S_ε of size $|S_\varepsilon| = |S|$ that satisfies $\Phi(S) > \Phi(S_\varepsilon)$ for some polynomially bounded potential $\Phi : P(G) \to \mathbb{N}$. Then Theorem 7 follows.*

Proof. Let S be a δ-dispersed set. Apply the assumed algorithm to obtain a $\varepsilon > 0$ and a $(\delta+\varepsilon)$-dispersed set S_ε of size $|S_\varepsilon| = |S|$. If $\delta = \delta^\star$, we reached our goal. Else we apply the assumed algorithm again. Since the potential $\Phi : P(G) \to \mathbb{N}$ decreases for S_ε compared to S and Φ is polynomially bounded, we have to reach $\delta^\star$ in polynomial many steps. ◀

In the remainder of this section we will develop such an algorithm. It pushes the points of point set S away from each other such that their pairwise distance increases from "at least δ" to "at least $\delta+\varepsilon$". We choose $\varepsilon \geq 0$ as large as possible limited by some events. Either we already reach $\delta+\varepsilon = \delta^\star$, hence we reached our goal, or at least one of three events occurs. We will specify these events in the course of this section. These events mean that one pushing step, i.e., one step for Theorem 8 terminated. All the following preparations for such a pushing step start anew.

We make sure that our potential $\Phi : P(G) \to \mathbb{N}$ decreases when an event occurs. Each of the three events has a corresponding partial potential $\Phi_1(S)$, $\Phi_2(S)$ and $\Phi_3(S)$. They define the overall potential as $\Phi(S) := \Phi_1(S) + \Phi_2(S) + \Phi_3(S)$. Each part never increases. Whenever event i occurs, $\Phi_i(S)$ strictly decreases.

We denote a pair of points $\{p,q\}$ from our given point set S as δ-*critical*, if they have distance exactly δ. Hence the critical pairs of points are exactly those that we need to push away from each other. At the same time we make sure that, once $\{p,q\}$ are δ-critical, they never turn uncritical again, i.e., they are $(\delta+\varepsilon)$-critical in the next step. An uncritical pair of points $\{p,q\}$ might become critical, hence we have to take care of $\{p,q\}$ in future steps. This constitutes our first event. The corresponding partial potential is $\Phi_1(S)$, the number of uncritical pairs of points $\{p,q\}$.

(Event 1) A δ-uncritical pair of points $\{p,q\}$ becomes $(\delta+\varepsilon)$-critical.

$$\Phi_1(S) \;:=\; \left|\left\{\{p,q\} \in \tbinom{S}{2} \mid \{p,q\} \text{ are not } \delta\text{-critical}\right\}\right| \;\leq\; |S|^2$$

5.2 Coordination of Movement

We need to coordinate the movement of all critical pairs of points. To this end, we will fix some set of *root points* R. Our movement will be locally prescribed for sequences of points $p_0, p_1, \ldots, p_s$ that originate in $p_0 \in R$ and where each $\{p_0, p_1\}, \ldots, \{p_{s-1}, p_s\}$ is critical. The overall movement will be uniquely defined by movement defined for these sequences.

For now, consider such a sequence of points $p_0, p_1, p_2, \ldots$. Our idea is to do not move p_0, to move p_1 by distance ε away from p_0, point p_2 by distance 2ε away from p_1 and so on. We have to stop pushing in this way as soon as one of the points, say p_i, becomes half-integral, i.e., p_i is moved onto a vertex or the midpoint of an edge. See Figure 1 for examples. This constitutes the second event.

(Event 2) A non-half-integral $p \in S$ becomes half-integral.

$$\Phi_2(S) \;\coloneqq\; \big|\{p \in S \mid p \text{ is not half-integral}\}\big| \;\le\; |S|.$$

The next pushing step will choose p_i as one of the root point R and will move the points away from p_i instead of p_0. Very similarly, we stop when a point $r \in P(G)$ that is "half-way" between two points p_i, p_{i-1} becomes half-integral. Formally, we denote such a point r as an (S, δ)-*pivot*, or simply a pivot, if it is half-integral and there is a (critical) pair of points $\{p, q\} \in \binom{S}{2}$, the witnesses, that have equal distances to r, which means $d(p, r) = d(q, r) = \frac{\delta}{2}$. Let $\mathsf{pivots}(S, \delta)$ be the set of (S, δ)-pivots, and let $W(S, \delta) \subseteq \binom{S}{2}$ be the family of pairs of points from S, that witness some (S, δ)-pivot. This leads to the third and final event.

(Event 3) A non-pivot point $r \in P(G)$ becomes a pivot.

$$\Phi_3(S) \;\coloneqq\; \big|\{r \in P(G) \mid r \text{ is half-integral}\} \setminus \mathsf{pivots}(S, \delta)\big| \;\le\; |V(G)|^2.$$

Hence a root point R may not only be a point $p \in S$ but also come from the set of pivots. We will later properly define R as a superset of half-integral points $p_i \in S$ and the (S, δ)-pivots.

We use an *auxiliary graph* G_S for the current δ-dispersed set S. Its vertex set is $S \cup \mathsf{pivots}(S, \delta)$. Essentially we make all pairs of critical $\{p, q\}$ adjacent unless they witness a pivot; If they do witness a pivot, we make them adjacent to the pivot:

- For $\{p, q\} \in W(S, \delta)$ and for every pivot $r \in \mathsf{pivots}(S, \delta)$ they witness, add edges $\{p, r\}, \{r, q\}$; and
- for every critical pair of points $\{p, q\} \in \binom{S}{2} \setminus W(S, \delta)$ add edge $\{p, q\}$.

Note that, for every edge $\{r, p\}$ with $r \in \mathsf{pivots}(S, \delta)$, there is at least one other edge $\{r, q\}$ such that p, q witness r as a pivot.

Now we define the sequence of points which serve as the structure to state the movement. A path $P = (p_0, p_1, \ldots, p_s)$ in the auxiliary graph G_S of length $s \ge 1$ is a *spine* if $p_1, \ldots, p_s$ are not half-integral. Note that any sub-sequence $(p_0, \ldots, p_i)$ for $1 \le i \le s$ is also a spine.

5.3 Velocities

We assign velocities vel_P to the points $p_0, \ldots, p_s$ of a spine P that specify their movement speed. The point p_i for $i \in \{1, \ldots, s\}$ is moved by $\mathsf{vel}(p_i)\varepsilon$. Thus setting $\mathsf{vel}(p_1) = 1$ makes the δ-critical $\{p_0, p_1\}$ become $(\delta + \varepsilon)$-critical, as desired. Setting $\mathsf{vel}(p_i) = i$ for $i \ge 1$, however, can make consecutive points $\{p_{i-1}, p_i\}$ uncritical. Figure 2 provides an example. To see this, fix some shortest p_{i-1}, p_i-path P_i and some shortest p_i, p_{i+1}-path P_{i+1}. The paths P_i and P_{i+1} can have a trivial intersection of only $\{p_i\}$ or their intersection may contain more than one point. We denote this bit of information as $\mathsf{flip}_P(p_i) \in \{-1, 1\}$. We set $\mathsf{flip}_P(p_i) = 1$

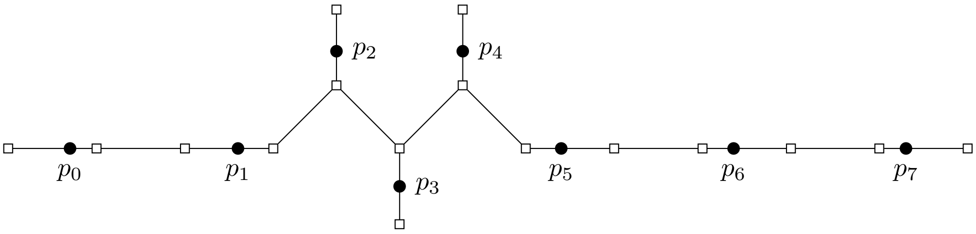


Figure 2 A spine $P = (p_0, \dots, p_7)$. The shortest path between p_0, p_1 and the shortest path between p_1, p_2 have the trivial intersection of $\{p_1\}$, hence $\mathsf{flip}_P(p_1) = 1$. In turn, the shortest path between p_1, p_2 and the shortest path between p_2, p_3 have a non-trivial intersection, hence $\mathsf{flip}_P(p_1) = -1$. Also $\mathsf{flip}_P(p_3) = \mathsf{flip}_P(p_4) = -1$ while the other values are positive. Consequently $(\mathsf{sgn}_P(p_1), \dots, \mathsf{sgn}_P(p_7)) = (1, 1, -1, 1, -1, -1, -1)$. Thus $(\mathsf{vel}_P(p_1), \dots, \mathsf{vel}_P(p_7)) = (1, 2, 1, 2, 1, 0, -1)$. In particular, $\mathsf{sgn}_P(p_5)$ is negative such that it is moved towards p_4. In turn, $\mathsf{sgn}_P(p_7)$ and $\mathsf{vel}_P(p_7)$ are negative such that p_7 has a net movement away from p_0. Under this movement all $\{p_0, p_1\}, \dots, \{p_6, p_7\}$ remain critical.

if and only if the path P_i and P_{i-1} have a trivial intersection. (The definition of flip_P is independent on the exact considered shortest paths and we will define it properly in the next subsection.)

The easy case is when P_i and P_{i+1} have a trivial intersection, i.e., $\mathsf{flip}_P(p_i) = 1$. Then we increase the velocity of the next point p_{i+1}. The first time we encounter the other case, that $\mathsf{flip}_P(p_i) = -1$, we decrease the velocity of the next point p_{i+1}. Further we move p_{i+1} *towards* and not away of p_i. Hence we also specify a sgn of the velocity that records whether a point p_{i+1} is pushed towards or away from its predecessor p_i. All these changes are relative to whether the movement of predecessor p_i is away from p_{i-1}, i.e., whether $\mathsf{sgn}(p_i)$ is positive. For example, the second time we encounter a point p_j with $\mathsf{flip}_P(p_j) = -1$, point p_{j+1} is again moved away from its predecessor.

This leads to the following definition of vel_P and sgn_P for a spine P. We define its half-integral *velocities* $\mathsf{vel}_P : \{p_0, \dots, p_s\} \to \{\frac{z}{2} \mid z \in \mathbb{Z}\}$ depending on *signs* $\mathsf{sgn}_P : \{p_1, \dots, p_s\} \to \{-1, 1\}$, which in turn depend on flip_P. We may drop the subscript P, if it is clear from the context. Let $\mathsf{vel}(p_0) = 0$. Let $\mathsf{vel}(p_1) = \frac{1}{2}$, if $p_0 \in \mathsf{pivots}(S, \delta)$, and let $\mathsf{vel}(p_1) = 1$, if $p_0 \in S$. For $i \geq 1$, let

$$\mathsf{vel}(p_{i+1}) \;\coloneqq\; \mathsf{vel}(p_i) + \mathsf{sgn}(p_{i+1}).$$

Thus $\mathsf{sgn} \in \{-1, 1\}$ indicates whether the velocity increases or decreases. The current sgn is unchanged unless flip is negative. Let $\mathsf{sgn}(p_1) = 1$. For $2 \leq i \leq s$, let

$$\mathsf{sgn}(p_i) \;\coloneqq\; \mathsf{flip}(p_{i-1})\,\mathsf{sgn}(p_{i-1}) \;=\; \prod_{0<j<i} \mathsf{flip}(p_j).$$

The movement step of a point p_i in a spine $P = (p_0, \dots, p_i)$ is now as follows. We push the point p_i by the (possibly negative) distance $\mathsf{sgn}_P(p_i)\,\mathsf{vel}_P(p_i)\varepsilon$ away from its predecessor p_{i-1}. In other words, the point $p_i = p(u, v, \lambda)$ is replaced by the point $p(u, v, \lambda + \mathsf{sgn}_P(p_i)\,\mathsf{vel}_P(p_i)\varepsilon)$ assuming that vertex u compared to v is in some sense closer to the predecessor point p_{i-1}. We make this notion formal in the next subsection.

5.4 Directions

We formalize the notion the direction of a point p towards another point q. The *direction* $\mathsf{dir}(p \to q) \in \{u, v\}$ for distinct points $p = p(u, v, \lambda) \in P(G)$ and $q \in P(G)$ is defined as follows:

- For points $p = p(u, v, \lambda_p)$ and $q = p(u, v, \lambda_q)$ on a common edge $\{u, v\} \in E(G)$ with $\lambda_p < \lambda_q$, let $\mathsf{dir}(p \to q) = v$. Let $\overline{\mathsf{dir}}(p \to q) = u$.
- For points $p = p(u_p, v_p, \lambda_p)$ and $q = p(u_q, v_q, \lambda_q)$ on distinct edges $\{u_p, v_p\} \neq \{u_q, v_q\}$, let $\mathsf{dir}(p \to q)$ be the unique vertex of $\{u_p, v_p\}$ that is contained in *every* shortest path between p and q, if such a vertex exists. If $\mathsf{dir}(p \to q)$ is defined, let $\overline{\mathsf{dir}}(p \to q)$ be the unique vertex in $\{u_p, v_p\} \setminus \{\mathsf{dir}(p \to q)\}$.

▶ **Lemma 9.** ($\star$) *For distinct points $p, q \in V(G_S)$, $\mathsf{dir}(p \to q)$ is well-defined, unless p is half-integral.*

Hence we can properly define $\mathsf{flip}(p_i)$ of a point p_i of a spine $(p_0, \ldots, p_s)$ with $1 \leq i \leq s-1$, since p_i with $i \geq 1$ is non-half-integral. Let

$$\mathsf{flip}(p_i) := \begin{cases} 1, & \mathsf{dir}(p_i \to p_{i-1}) \neq \mathsf{dir}(p_i \to p_{i+1}), \\ -1, & \text{else.} \end{cases}$$

Further, for a non-half-integral point $p = p(u, v, \lambda)$ we have $\{u, v\} = \{\mathsf{dir}(p \to q), \overline{\mathsf{dir}}(p \to q)\}$. By symmetry assume that $\overline{\mathsf{dir}}(p \to q) = u$. We can equivalently specify point p as $p(\overline{\mathsf{dir}}(p \to q), \mathsf{dir}(p \to q), \lambda)$. Conveniently, we may write $p = p(\cdot, \overline{\mathsf{dir}}(p \to q), \lambda)$ since the missing entry is clear from the context. Doing so, the edge position λ measures a part of the length of any shortest p, q-path, specifically the part using the edge of p (assuming q is on another edge).

Now we can also properly define one pushing step for a point p_i of a spine $P = (p_0, \ldots, p_s)$ and for $\varepsilon > 0$. Let λ_i be such that $p_i = p(\cdot, \overline{\mathsf{dir}}(p_i \to p_{i-1}), \lambda_i)$. Then the new point is

$$(p_i)_{P,\varepsilon} := p\big(\cdot,\ \overline{\mathsf{dir}}(p_i \to p_{i-1}),\ \lambda_i + \mathsf{sgn}_P(p_i)\,\mathsf{vel}_P(p_i)\varepsilon\big).$$

▶ **Lemma 10.** ($\star$) *For a spine $P = (p_0, \ldots, p_s)$ and $i \in \{0, \ldots, s-1\}$, points $(p_i)_{P,\varepsilon}, (p_{i+1})_{P,\varepsilon}$ are $(\delta + \varepsilon)$-critical for the maximal $\varepsilon \leq \delta^\star - \delta$ that is limited by the Events 1,2,3.*

5.5 Root Points

We formally define the set of root points R. Let R_0 be the set of half-integral points in G_S. There may be some components of the auxiliary graph G_S without a point in R_0. Let R result from R_0 by adding exactly one point from every component that has no point in R_0.

We consider only spines $P = (p_0, \ldots, p_i)$ where $p_0 \in R$. Clearly every point G_S is part of at least one spine and hence has some movement prescribed. We also claim that the prescribed movement is uniquely defined. In other words, there are no spines $P = (p_0, \ldots, p_i)$ and $Q = (q_0, \ldots, q_j)$ with $p_0, q_0 \in R$ that terminate at the same point $p_i = q_j$ and contradict in their prescribed movement for $p_i = q_j$.

▶ **Lemma 11.** ($\star$) *Let $\delta < \delta^\star$. Consider spines $P = (p_0, \ldots, p_i)$ and $Q = (q_0, \ldots, q_j)$ with $p_0, q_0 \in R$ and $p_i = q_j$. Then (1) $\mathsf{vel}_P(p_i) = \mathsf{vel}_Q(q_j)$; and (2) $\overline{\mathsf{dir}}(p_i \to p_{i-1}) = \overline{\mathsf{dir}}(q_j \to q_{j-1})$ if and only if $\mathsf{sgn}_P(p_i) = \mathsf{sgn}_Q(q_j)$.*

Therefore we can uniquely define the moved version of a point p as $p_\varepsilon := (p)_{P,\varepsilon}$ where we may choose P to be an arbitrary spine starting in R and containing p. This defines the set of pushed points $S_\varepsilon := \{\, p_\varepsilon \mid p \in S \,\}$.

For the proof of Lemma 11, we use that $\delta^\star = \frac{a^\star}{b^\star} \geq \delta$ is minimal with $a^\star \leq 2L$. Since spines P, Q meet at the point $p_i = q_j$ their roots p_0, q_0 must be from the same component in the auxiliary graph G_S; in other words p_0, q_0 are both half-integral or are the same point.

Our proof by contradiction considers these two options and whether P, Q reach p_i and q_j from the same vertex relatively to if they agree on the sgn of $p_i = q_j$. An example is that $p_0 = q_0$ and P, Q reach $p_i = q_j$ from the same vertex (formally with $\overline{\mathsf{dir}}(p_i \to p_{i-1}) \neq \overline{\mathsf{dir}}(q_j \to q_{j-1})$) while they agree on the sgn (formally $\mathsf{sgn}_P(p_i) = \mathsf{sgn}_Q(q_j)$). Then we can glue P and Q together forming a walk starting in $p_0 = q_0$ and returning to $p_0 = q_0$. This walk then has half-integral length at most $2L$ but is made up of hops of length δ. That implies the contradiction that already $\delta = \delta^\star$.

5.6 Summery

With the previous observations we can assemble the algorithm for Theorem 8. Our potential counts how many elements can still trigger Events 1,2,3. That is

$$\Phi(S) \;:=\; \Phi_1(S) + \Phi_2(S) + \Phi_3(S) \;\leq\; 2|S|^2 + |V(G)|^2.$$

We define $\varepsilon^\star \geq 0$ as the maximal $\varepsilon \leq \delta^\star - \delta$ limited by the Events 1,2,3. We claim that $\varepsilon^\star \geq 0$ is defined. This is due to that the above events depend on continuous functions in ε, which are the distance of p_ε to its closest half-integral point, and the distance between points p_ε and q_ε for $p, q \in S$.

To show termination, we prove that no such element can trigger its according event more than once. Lemma 10 already implies that a δ-critical pair of points $\{p, q\}$ stays $(\delta + \varepsilon^\star)$-critical. It remains to show the following monotonicities:

▶ **Lemma 12.** ($\star$) *Let S be a δ-dispersed set for $\delta < \delta^\star$ and $\varepsilon^\star$ defined as above. Then:*

- (1) *$S_{\varepsilon^\star}$ is a $(\delta + \varepsilon^\star)$-dispersed set of size $|S|$.*
- (2) *If $\{p, q\} \in \binom{S}{2}$ is δ-critical, then $\{p_{\varepsilon^\star}, q_{\varepsilon^\star}\}$ is $(\delta + \varepsilon^\star)$-critical.*
- (3) *If $r \in \mathsf{pivots}(S, \delta)$, then $r \in \mathsf{pivots}(S_{\varepsilon^\star}, \delta + \varepsilon^\star)$.*

Now we have all the tools to show Theorem 8. Determine $\varepsilon^\star$ and the $(\delta + \varepsilon^\star)$-dispersed set $S_{\varepsilon^\star}$ as defined before. The resulting set $S_{\varepsilon^\star}$ is a $(\delta + \varepsilon^\star)$-dispersed set of the same size, according to Lemma 12. If $\delta + \varepsilon^\star = \delta^\star$ then $S_{\varepsilon^\star}$ is already the desired $\delta^\star$-dispersed set. Else one of the Events 1,2,3 occurred. We observe that the potential strictly decreases, that is $\Phi(S_{\varepsilon^\star}) < \Phi(S)$. Because of the monotonicities of Lemma 10 and Lemma 12 the partial potentials Φ_1, Φ_2 and Φ_3 do not increase. If Event 1 occurs then Φ_1 strictly decreases. If Event 2 occurs then Φ_2 strictly decreases. If Event 3 occurs then Φ_3 strictly decreases. All in all at least one part strictly decreases and so does Φ. This completes the proof of Theorem 8 and hence of Theorem 7.

6 Algorithmic Implications

Based on the rounding procedure from Section 5, the translation result from Section 3 and connections to distance-d independent set we derive a number of algorithmic results.

▶ **Theorem 13.** *There is an algorithm that, given distance $\delta \geq 0$, a graph G, a tree decomposition and an upper bound $L \in \mathbb{N}$ on the length of the longest path in G, computes a maximum δ-dispersed set S in time $(2L)^{\mathsf{tw}(G)} n^{\mathcal{O}(1)}$.*

Proof. According to Theorem 7, we may consider the rounded up distance, that is a rational $\frac{a}{b} \geq \delta$ with $a \leq 2L$, instead of δ. Notice that $\frac{a}{b}$ is polynomial time computable. As long as $\frac{a}{b} \leq \frac{3}{4}$, we may repeatedly apply Corollary 4 such that eventually we obtain that $\frac{3}{4}b < a \leq 2L$. Let G_{2b} be a $2b$-subdivision of G. Observe that $\mathsf{tw}(G) = \mathsf{tw}(G_{2b})$ and the number of vertices increases only by a factor of $\mathcal{O}(n^2L)$. According to Lemma 5 $\frac{a}{b}$-disp$(G) = \alpha_{2a}(G_{2b})$. Thus, to answer the original δ-Dispersion-instance we may find a maximum distance-$2a$ independent set in G_{2b}, which is possible in time $(2a)^{\mathsf{tw}(G)}n^{\mathcal{O}(1)}$, according to [12]. ◀

This result immediately yields parameterized complexity results for the parameters treedepth and treewidth. Regarding the treewidth, note that n is an upper bound on L. Thus the above algorithm is an XP-algorithm for the parameter treewidth. When a treewidth decomposition is given, Dispersion can be solved in time $2n^{\mathsf{tw}(G)}n^{\mathcal{O}(1)}$.

▶ **Corollary 14.** *Dispersion can be solved in time $2n^{\mathsf{tw}(G)}n^{\mathcal{O}(1)}$, assuming a tree decomposition is given.*

Similarly we obtain an FPT algorithm for treedepth $\mathsf{td}(G)$ of the input graph. The treedepth $\mathsf{td}(G)$ implies a bound on L, which is $L \leq 2^{\mathsf{td}(G)}$. Since also $\mathsf{td}(G) \geq \mathsf{tw}(G)$, we obtain an $2^{\mathcal{O}(\mathsf{td}(G)^2)}n^{\mathcal{O}(1)}$-time algorithm, assuming a treedepth decomposition is given.

▶ **Corollary 15.** *Dispersion can be solved in time $2^{\mathcal{O}(\mathsf{td}(G)^2)}n^{\mathcal{O}(1)}$, assuming a treedepth decomposition is given.*

7 Parameterized Hardness Results

We complement the positive results by hardness results. These results borrow ideas from hardness-reductions for the similar problem Distance Independent Set (DIS), see Section 4.

A natural generalization of treedepth is the maximum diameter of graph G, which is the maximum distance between any vertices $u, v \in V(G)$ (since we only consider connected graphs G). We show NP-hardness for graphs of any diameter ≥ 3 even for chordal graphs by a reduction from Independent Set, similarly as NP-hardness for DIS is shown by Eto et al. [5]. Our reduction also shows W[1]-hardness with respect to the solution size k.

▶ **Lemma 16.** ($\star$) *For every $\delta > 3$, δ-Dispersion is NP-complete and W[1]-hard with parameter solution size, even for connected chordal graphs of diameter $\leq \lceil\delta\rceil$.*

Another direct generalization of treedepth is pathwidth of the input graph G. We show W[1]-hardness even for the combined parameters pathwidth and solution size $\mathsf{pw}(G) + k$. With the same reduction also W[1]-hardness for the combined parameters "feedback vertex set size" $\mathsf{fvs}(G)$ and solution size k follows. We can essentially use the same reduction as used by Katsikarelis et al. to show W[1]-hardness of DIS when parameterized by $\mathsf{fvs}(G) + k$ by reducing from Multi-Colored-Independent-Set [12].

▶ **Theorem 17.** ($\star$) *Dispersion is W[1]-hard parameterized by $\mathsf{pw}(G)+k$. Further, there is no $n^{o(\sqrt{\mathsf{pw}(G)}+\sqrt{k})}$-time algorithm unless ETH fails. Dispersion is W[1]-hard parameterized by $\mathsf{fvs}(G) + k$. Further, there is no $n^{o(\mathsf{fvs}(G)+\sqrt{k})}$-time algorithm unless ETH fails.*

Since $\mathsf{fvs}(G)$ is a linear upper bound for the treewidth of G, we also obtain: Dispersion is W[1]-hard parameterized by $\mathsf{tw}(G) + k$. Further, there is no $n^{o(\mathsf{tw}(G)+\sqrt{k})}$-time algorithm unless ETH fails. Similarly as in [12] we obtain a lower bound for treedepth.

▶ **Theorem 18.** ($\star$) *Assuming ETH, there is no $2^{o(\mathsf{td}(G)^2)}$-time algorithm for Dispersion.*

8 NP-hardness for Irrational Distance

We show NP-hardness of δ-Dispersion for every irrational distance $\delta > 0$. Thus together with earlier results [8] the complexity for every real $\delta > 0$ is resolved: For rational distance $\delta = \frac{a}{b}$ where $a \in \{1, 2\}$ the problem is polynomial time solvable, while it is NP-complete for every other distance $\delta > 0$.

▶ **Theorem 19.** *For every irrational $\delta > 0$, δ-Dispersion is NP-complete.*

The key step is a reduction from Independent Set which shows NP-hardness not only for a single distance δ but for the whole interval $\delta \in (2, 3]$.

Construction. Given a graph G and integer $k \in \mathbb{N}$, we construct an input for δ-Dispersion consisting of a graph G' and integer $k' = k$ as follows. For every vertex $u \in V(G)$ introduce vertices u_1, u_2 and edge $\{u_1, u_2\}$. For every edge $\{u, v\} \in E(G)$ introduce edges $\{u_i, v_j\}$ for every $i, j \in \{1, 2\}$.

▶ **Lemma 20.** *For every $\delta \in (2, 3]$, δ-Dispersion is NP-hard and W[1]-hard when parameterized by solution size.*

Proof. Clearly, this construction is polynomial time computable. Further, the reduction is parameter preserving such that W[1]-hardness of Independent Set implies W[1]-hardness of Dispersion, assuming correctness of the reduction.

Hence, it remains to show the correctness, that G has an independent set of size k if and only if G' has a δ-dispersed set of size k.

($\Rightarrow$) Let $I \subseteq V(G)$ be an independent set of graph G. We define $S := \{p(u_1, u_2, \frac{1}{2}) \mid u \in I\} \subseteq P(G)$, which has size $|S| = |I|$. We claim that S is δ-dispersed in G' for $\delta \in (2, 3]$. Since any vertices $u, v \in V(G)$ have distance at least 2 in G, their corresponding points $p(u_1, u_2, \frac{1}{2})$ and $p(v_1, v_2, \frac{1}{2})$ have distance at least 3 in $P(G)$. Thus they are δ-dispersed for $\delta \in (2, 3]$.

($\Leftarrow$) Let $S \subseteq P(G)$ be a δ-dispersed set for some $\delta \in (2, 3]$. We define the *ball* B_u for $u \in V(G)$ as the points in $P(G)$ with distance at most $\frac{1}{2}$ to u_1 or u_2, which is $B_u := \{p(u_i, v, \lambda) \mid i \in \{1, 2\}, \{u_i, v\} \in E(G'), \lambda \in [0, \frac{1}{2}]\}$. Every ball B_u for $u \in V(G)$ contains at most one point from S since points $p, q \in B_u$ can be at most $2 < \delta$ apart. Every union $B_u \cup B_v$ for adjacent $\{u, v\} \in E(G)$ contains at most one point from S since points $p, q \in B_u \cup B_v$ can also be at most $2 < \delta$ apart.

Now we define an independent set $I \subseteq V(G)$. Add vertex $u \in V(G)$ for every point $p \in S \cap B_u$ except when $p \in B_u \cap B_v$ for some $v \in P(G)$. If $p \in S \cap B_u \cap B_v$, add either the point u or v to I. Then $|I| = |S|$ since the union of B_u for $u \in V(G)$ is the whole point space $P(G)$. Further, no adjacent vertices u, v are in I since $B_u \cup B_v$ contain at most one point from S. Thus $I \subseteq V(G)$ is an independent set of size $|S|$. ◀

Because $\delta \le 3$ we may apply Lemma 3 to obtain NP-hardness for δ in the interval $(\frac{2}{2x+1}, \frac{3}{3x+1}]$ for every integer $x \ge 0$. Applying Lemma 1 yields NP-hardness for δ in the interval $(\frac{2c}{2x+1}, \frac{3c}{3x+1}]$ for every integer $c \ge 1$.

Now, consider any irrational distance $\delta > 0$. Consider $F := \{c\delta^{-1} - \lfloor c\delta^{-1} \rfloor \mid c \ge 1\}$, the set of fractional parts of multiples of δ^{-1}. Since δ^{-1} is irrational, F is a dense subset of the interval $[0, 1]$. Let integer $c \ge 1$ be such that $\frac{1}{3} \le c\delta^{-1} - \lfloor c\delta^{-1} \rfloor < \frac{1}{2}$. Thus there is a non-negative x such that $x + \frac{1}{3} \le c\delta^{-1} < x + \frac{1}{2}$. This implies that $\frac{2c}{2x+1} < \delta \le \frac{3c}{3x+1}$ and hence NP-hardness for δ-dispersion. This finishes the proof of Theorem 19.

9 Parameter Solution Size

δ-Dispersion parameterized by the solution size k is W[1]-hard when $\delta > 2$: When $\delta \in (2, 3]$ Lemma 20 shows W[1]-hardness, while for $\delta > 3$ Lemma 16 implies W[1]-hardness even when the input graph is chordal. It remains to consider $\delta \leq 2$. Observe that for $\delta \leq 2$, every graph G satisfies $\delta\text{-disp}(G) \geq \nu(G)$ [8], where $\nu(G)$ is the maximum matching size of G. Thus, a win-win situation occurs. Determine $\nu(G)$ in polynomial time. If $k \leq \nu(G)$, we may immediately answer "yes". Otherwise $k > \nu(G) \geq \frac{\mathsf{vc}(G)}{2}$, where $\mathsf{vc}(G)$ is the minimum size of a vertex cover in G. The size of a vertex cover upper bounds the treedepth. A treedepth decomposition of size $\mathsf{td}(G)$ is computable in FPT-time [15]. Thus we may apply the FPT algorithm for parameter treedepth from Theorem 13.

▶ **Theorem 21.** *δ-Dispersion parameterized by solution size k is FPT if $\delta \leq 2$; and $W[1]$-hard if $\delta > 2$.*

References

1 Shimon Abravaya and Michael Segal. Maximizing the number of obnoxious facilities to locate within a bounded region. *Comput. Oper. Res.*, 37(1):163–171, 2010. doi:10.1016/j.cor.2009.04.004.

2 Boaz Ben-Moshe, Matthew J. Katz, and Michael Segal. Obnoxious facility location: complete service with minimal harm. *International Journal of Computational Geometry and Applications*, 10(6):581–592, 2000.

3 Richard L. Church and Robert S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12:107–118, 1978.

4 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.

5 Hiroshi Eto, Fengrui Guo, and Eiji Miyano. Distance-d independent set problems for bipartite and chordal graphs. *J. Comb. Optim.*, 27(1):88–99, 2014. doi:10.1007/s10878-012-9594-4.

6 Alan J. Goldman and Perino M. Dearing. Concepts of optimal location for partially noxious facilities. *ORSA Bulletin*, 23:B–31, 1975.

7 Alexander Grigoriev, Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger. Dispersing obnoxious facilities on a graph. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 33:1–33:11, 2019. doi:10.4230/LIPIcs.STACS.2019.33.

8 Alexander Grigoriev, Tim A Hartmann, Stefan Lendl, and Gerhard J Woeginger. Dispersing obnoxious facilities on a graph. *Algorithmica*, 83(6):1734–1749, 2021.

9 Tim A. Hartmann and Stefan Lendl. Dispersing obnoxious facilities on graphs by rounding distances, 2022. doi:10.48550/ARXIV.2206.11337.

10 Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger. Continuous facility location on graphs. *Math. Program.*, 192(1):207–227, 2022. doi:10.1007/s10107-021-01646-x.

11 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

12 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 292–305, 2018. doi:10.1007/978-3-030-00256-5_24.

13 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.

14 Matthew J. Katz, Klara Kedem, and Michael Segal. Improved algorithms for placing undesirable facilities. *Comput. Oper. Res.*, 29(13):1859–1872, 2002. doi:10.1016/S0305-0548(01)00063-6.

15 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 931–942, 2014. doi:10.1007/978-3-662-43948-7_77.

16 Michael Segal. Placing an obnoxious facility in geometric networks. *Nordic Journal of Computing*, 10:224–237, 2003.

17 Arie Tamir. On the solution value of the continuous p-center location problem on a graph. *Math. Oper. Res.*, 12(2):340–349, 1987. doi:10.1287/moor.12.2.340.

18 Arie Tamir. Obnoxious facility location on graphs. *SIAM J. Discret. Math.*, 4(4):550–567, 1991. doi:10.1137/0404048.

19 Martijn van Ee. Approximability of the dispersed p→-neighbor k-supplier problem. *Discret. Appl. Math.*, 289:219–229, 2021. doi:10.1016/j.dam.2020.10.007.

On the Binary and Boolean Rank of Regular Matrices

Ishay Haviv
The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Michal Parnas
The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Abstract

A $0,1$ matrix is said to be regular if all of its rows and columns have the same number of ones. We prove that for infinitely many integers k, there exists a square regular $0,1$ matrix with binary rank k, such that the Boolean rank of its complement is $k^{\widetilde{\Omega}(\log k)}$. Equivalently, the ones in the matrix can be partitioned into k combinatorial rectangles, whereas the number of rectangles needed for any cover of its zeros is $k^{\widetilde{\Omega}(\log k)}$. This settles, in a strong form, a question of Pullman (Linear Algebra Appl., 1988) and a conjecture of Hefner, Henson, Lundgren, and Maybee (Congr. Numer., 1990). The result can be viewed as a regular analogue of a recent result of Balodis, Ben-David, Göös, Jain, and Kothari (FOCS, 2021), motivated by the clique vs. independent set problem in communication complexity and by the (disproved) Alon-Saks-Seymour conjecture in graph theory. As an application of the produced regular matrices, we obtain regular counterexamples to the Alon-Saks-Seymour conjecture and prove that for infinitely many integers k, there exists a regular graph with biclique partition number k and chromatic number $k^{\widetilde{\Omega}(\log k)}$.

2012 ACM Subject Classification Theory of computation → Communication complexity; Mathematics of computing → Graph coloring

Keywords and phrases Binary rank, Boolean rank, Regular matrices, Non-deterministic communication complexity, Biclique partition number, Chromatic number

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.56

Related Version *Full Version*: `http://arxiv.org/abs/2203.13073`

Funding *Ishay Haviv*: Research supported in part by the Israel Science Foundation (grant No. 1218/20).

Acknowledgements We thank the anonymous reviewers for their helpful and constructive comments.

1 Introduction

For a $0,1$ matrix M of dimensions $n \times m$, consider the following three notions of rank.

- The (standard) *rank* of M over $\mathbb{R}$, denoted by $\mathrm{rank}_{\mathbb{R}}(M)$, is the minimal k for which there exist real matrices A and B of dimensions $n \times k$ and $k \times m$ respectively, such that $M = A \cdot B$ where the operations are over $\mathbb{R}$.
- The *binary rank* of M, denoted by $\mathrm{rank}_{\mathrm{bin}}(M)$, is the minimal k for which there exist $0,1$ matrices A and B of dimensions $n \times k$ and $k \times m$ respectively, such that $M = A \cdot B$ where the operations are over $\mathbb{R}$. Equivalently, $\mathrm{rank}_{\mathrm{bin}}(M)$ is the smallest number of monochromatic combinatorial rectangles in a *partition* of the ones in M.
- The *Boolean rank* of M, denoted by $\mathrm{rank}_{\mathbb{B}}(M)$, is the minimal k for which there exist $0,1$ matrices A and B of dimensions $n \times k$ and $k \times m$ respectively, such that $M = A \cdot B$ where the operations are under Boolean arithmetic (namely, $0+x = x+0 = x$, $1+1 = 1 \cdot 1 = 1$, and $x \cdot 0 = 0 \cdot x = 0$). Equivalently, $\mathrm{rank}_{\mathbb{B}}(M)$ is the smallest number of monochromatic combinatorial rectangles in a *cover* of the ones in M.

Note that every $0,1$ matrix M satisfies $\mathrm{rank}_{\mathrm{bin}}(M) \geq \mathrm{rank}_{\mathbb{R}}(M)$ and $\mathrm{rank}_{\mathrm{bin}}(M) \geq \mathrm{rank}_{\mathbb{B}}(M)$.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 56; pp. 56:1–56:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The above notions of rank play a central role in the area of communication complexity, introduced in 1979 by Yao [31]. In the communication problem associated with a $0,1$ matrix M of dimensions $n \times m$, one player holds a row index $i \in [n]$ and another player holds a column index $j \in [m]$, and their goal is to decide whether $M_{i,j} = 1$ while minimizing the worst-case number of communicated bits. For the deterministic setting, the well-known log-rank conjecture of Lovász and Saks [24] suggests that the communication complexity of the problem is polynomially related to $\log_2 \text{rank}_{\mathbb{R}}(M)$ (see, e.g., [25]). For the non-deterministic setting, it is not difficult to see that the minimum number of bits that should be communicated is precisely $\lceil \log_2 \text{rank}_{\mathbb{B}}(M) \rceil$. For the *unambiguous* non-deterministic setting, where each input is required to have at most one accepting computation, the minimum number of bits that should be communicated is precisely $\lceil \log_2 \text{rank}_{\text{bin}}(M) \rceil$.

For a $0,1$ matrix M, let $\overline{M}$ denote the complement matrix obtained from M by replacing the ones by zeros and the zeros by ones. A result of Yannakakis [30] implies that every $0,1$ matrix M with $\text{rank}_{\text{bin}}(M) = k$ satisfies

$$\text{rank}_{\mathbb{B}}(\overline{M}) \leq \text{rank}_{\text{bin}}(\overline{M}) \leq k^{O(\log k)}. \tag{1}$$

The challenge of determining the largest possible value of $\text{rank}_{\mathbb{B}}(\overline{M})$ for a $0,1$ matrix M with $\text{rank}_{\text{bin}}(M) = k$ has attracted intensive attention in the literature, mostly with the equivalent formulation of the clique vs. independent set problem introduced in [30] (see [20, Chapter 4.4]). The first non-trivial bound was given by Huang and Sudakov [19] who provided, building on a construction of Razborov [28], a family of such matrices M satisfying $\text{rank}_{\mathbb{B}}(\overline{M}) \geq \Omega(k^{6/5})$ (see [11] for extended constructions). The constant $6/5$ in the exponent was improved to $3/2$ by Amano [1] and then to 2 by Shigeta and Amano [29]. The first super-polynomial separation was obtained by Göös [13], who provided a family of such matrices M satisfying $\text{rank}_{\mathbb{B}}(\overline{M}) \geq k^{\Omega(\log^{0.128} k)}$. This was then improved in a work of Ben-David, Hatami, and Tal [3] to $\text{rank}_{\mathbb{B}}(\overline{M}) \geq k^{\Omega(\log^{0.22} k)}$. In a recent breakthrough, it was shown by Balodis, Ben-David, Göös, Jain, and Kothari [2] that the bound can be further improved to $\text{rank}_{\mathbb{B}}(\overline{M}) \geq k^{\widetilde{\Omega}(\log k)}$, which matches the upper bound in (1) up to $\log \log k$ factors hidden in the $\widetilde{\Omega}$ notation. Note that the result of [2] strengthens an earlier result of Göös, Pitassi, and Watson [15], who provided a near optimal separation between the binary rank of a $0,1$ matrix and the deterministic communication complexity of the problem associated with it.

Interestingly, the above problem is closely related to a graph-theoretic problem proposed by Alon, Saks, and Seymour in 1991 (see [21]). For a graph G, let $\chi(G)$ denote its chromatic number, and let $\text{bp}(G)$ denote its biclique partition number, that is, the smallest number of edge-disjoint bicliques (i.e., complete bipartite graphs) needed for a partition of the edge set of G. A classic result of Graham and Pollak [16] asserts that the complete graph K_n on n vertices satisfies $\text{bp}(K_n) = n - 1$. Inspired by this result, Alon, Saks, and Seymour conjectured that every graph G satisfies $\text{bp}(G) \geq \chi(G) - 1$. The conjecture was disproved by Huang and Sudakov in [19], where it was shown that for infinitely many integers k there exists a graph G satisfying $\text{bp}(G) = k$ and $\chi(G) \geq \Omega(k^{6/5})$. These graphs were used there to derive the aforementioned separation between $\text{rank}_{\text{bin}}(M)$ and $\text{rank}_{\mathbb{B}}(\overline{M})$ for $0,1$ matrices M (see [19, Section 4]). In a work of Bousquet, Lagoutte, and Thomassé [6], the two problems were shown to be essentially equivalent, allowing the authors of [2] to derive, for infinitely many integers k, the existence of a graph G satisfying $\text{bp}(G) = k$ and $\chi(G) \geq k^{\widetilde{\Omega}(\log k)}$. As in the matrix setting, the gap is optimal up to $\log \log k$ factors in the exponent.

A $0,1$ matrix M is said to be *d-regular* if every row and every column in M has precisely d ones. In 1986, Brualdi, Manber, and Ross [7] proved that for every d-regular $0,1$ matrix M of dimensions $n \times n$ where $0 < d < n$, the rank of M over the reals is equal to that of its complement, that is, $\text{rank}_{\mathbb{R}}(M) = \text{rank}_{\mathbb{R}}(\overline{M})$. Following their work, Pullman [27] asked in 1988 whether every such matrix M satisfies $\text{rank}_{\text{bin}}(M) = \text{rank}_{\text{bin}}(\overline{M})$. In 1990, Hefner, Henson, Lundgren, and Maybee [18] conjectured that the answer to this question is negative (see [18, Conjecture 3.2]). The question was asked again in 1995 in a survey by Monson, Pullman, and Rees [26] (see [26, Open problem 7.1]).[1] Note that for the Boolean rank, such a statement does not hold in general. For example, the 1-regular identity matrix I_n satisfies $\text{rank}_{\mathbb{B}}(I_n) = n$ and yet $\text{rank}_{\mathbb{B}}(\overline{I_n}) = (1 + o(1)) \cdot \log_2 n$ (see [12]).

1.1 Our Contribution

The current work aims to determine the largest possible gap between the binary rank of *regular* 0, 1 matrices and the Boolean rank of their complement. Our main result is the following.

▶ **Theorem 1.** *For infinitely many integers k, there exists a square regular 0, 1 matrix M satisfying $\text{rank}_{\text{bin}}(M) = k$ and $\text{rank}_{\mathbb{B}}(\overline{M}) \geq k^{\widetilde{\Omega}(\log k)}$.*

Theorem 1 can be viewed as a regular analogue of the aforementioned result of Balodis et al. [2], showing that their near optimal separation between $\text{rank}_{\text{bin}}(M)$ and $\text{rank}_{\mathbb{B}}(\overline{M})$ can also be attained by regular matrices M. Since every 0, 1 matrix M satisfies $\text{rank}_{\text{bin}}(\overline{M}) \geq \text{rank}_{\mathbb{B}}(\overline{M})$, Theorem 1 settles, in a strong form, the question of Pullman asked in [27, 26] (and the variants of the question mentioned there) and confirms the conjecture of Hefner et al. [18]. We remark that regular matrices M with $\text{rank}_{\mathbb{B}}(\overline{M})$ larger than $\text{rank}_{\text{bin}}(M)$ can also be derived from [19] (see Section 1.2 for details). While these matrices are sufficient to answer the original question of [27, 26], they only achieve a polynomial gap between the quantities.

The proof of Theorem 1 relies on a modification of the construction of [2] to the regular setting. It involves an extension of the query-to-communication lifting theorem in non-deterministic communication complexity proved by Göös, Lovett, Meka, Watson, and Zuckerman [14], as well as a two-source extractor studied by Bouda, Pivoluska, and Plesch [4] and by Kothari, Meka, and Raghavendra [22]. For an overview of the proof, see Section 1.2.

As alluded to before, matrices M with $\text{rank}_{\text{bin}}(M)$ much smaller than $\text{rank}_{\mathbb{B}}(\overline{M})$ are known to imply graphs G with $\text{bp}(G)$ much smaller than $\chi(G)$, and thus yield counterexamples to the Alon-Saks-Seymour conjecture (see [6]). Although the conjecture is false in general, it is of interest to identify classes of graphs that satisfy a polynomial version of the conjecture. In particular, it was asked in [2] whether the chromatic number of perfect graphs is polynomially upper bounded in terms of their biclique partition number (see [30] for a related question; see also [23, 5, 10]). As an application of Theorem 1, we show that this is not the case for the class of regular graphs. Namely, we show that the near optimal separation achieved in [2] between the biclique partition number and the chromatic number can also be attained by regular graphs.

▶ **Theorem 2.** *For infinitely many integers k, there exists a simple regular graph G satisfying $\text{bp}(G) = k$ and $\chi(G) \geq k^{\widetilde{\Omega}(\log k)}$.*

1.2 Overview of Proofs

Our goal is to obtain regular 0, 1 matrices M for which the binary rank of M is much smaller than the Boolean rank of $\overline{M}$. We first observe that a polynomial gap between the two quantities, for a regular matrix, can be derived from a construction of Huang and

[1] The question of [27, 18, 26] was originally formulated using the notion of *non-negative integer rank*, which coincides with the binary rank for 0, 1 matrices (see, e.g., [17, Lemma 2.1]).

Sudakov [19]. Indeed, it can be verified that the (simple) graphs G given in [19], which satisfy $\mathrm{bp}(G) = k$ and $\chi(G) \geq \Omega(k^{6/5})$, are regular, hence their adjacency matrices are regular as well. The following simple claim implies that these adjacency matrices achieve a polynomial gap between the binary rank and the Boolean rank of the complement.

▷ Claim 3. For every simple graph G, the adjacency matrix M of G satisfies that $\mathrm{rank_{bin}}(M) \leq 2 \cdot \mathrm{bp}(G)$ and $\mathrm{rank}_{\mathbb{B}}(\overline{M}) \geq \chi(G)$.

Proof. For a simple graph G on the vertex set $[n]$, put $k = \mathrm{bp}(G)$, and consider the k bipartitions $(A_1, B_1), \ldots, (A_k, B_k)$ of the k edge-disjoint bicliques that form a partition of the edge set of G. Observe that for every $i \in [k]$, the sets $A_i \times B_i$ and $B_i \times A_i$ form combinatorial rectangles of ones in the adjacency matrix M of G, and that these $2k$ rectangles form a partition of the ones in M, hence $\mathrm{rank_{bin}}(M) \leq 2 \cdot k$.

Next, put $m = \mathrm{rank}_{\mathbb{B}}(\overline{M})$, and let $A_1 \times B_1, \ldots, A_m \times B_m$ be m combinatorial rectangles that form a cover of the ones in $\overline{M}$, i.e., the zeros in M. For every $i \in [m]$, let C_i denote the set of elements $j \in [n]$ satisfying $(j, j) \in A_i \times B_i$. Since G is simple, the elements on the diagonal of M are all zeros, hence the sets C_i for $i \in [m]$ cover all vertices of G. Since $A_i \times B_i$ is a rectangle of zeros in M, it also follows that C_i is an independent set in G. This implies that $m \geq \chi(G)$, and we are done. ◁

The matrices M that are known to achieve super-polynomial separations between $\mathrm{rank_{bin}}(M)$ and $\mathrm{rank}_{\mathbb{B}}(\overline{M})$, however, are apparently far from being regular [13, 3, 2]. Their constructions rely on a powerful technique, known as *query-to-communication lifting*, that enables to deduce separation results in communication complexity from separation results in the more approachable area of query complexity. The proofs of the separation results of [13, 3, 2] involve two main steps, as described below.

In the first step, one provides a family of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ with a large gap between two certain measures of Boolean functions, namely, the unambiguous 1-certificate complexity of f and the 0-certificate complexity of f (see Section 2.3). These measures can be viewed as query complexity analogues of the binary rank of a matrix and the Boolean rank of its complement. It is shown in [2] that the gap between the two measures can be nearly quadratic.

In the second step, the separation is "lifted" from query complexity to communication complexity. This is done by considering, for some gadget function $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$, the communication problem in which two players get inputs from $\{0,1\}^{\ell \cdot n}$ and aim to determine the value of the composed function $f \circ g^n : \{0,1\}^{\ell \cdot n} \times \{0,1\}^{\ell \cdot n} \to \{0,1\}$, defined by $(f \circ g^n)(x, y) = f(g(x_1, y_1), g(x_2, y_2), \ldots, g(x_n, y_n))$ for all $x, y \in \{0,1\}^{\ell \cdot n}$. Here, the vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are viewed as concatenations of n blocks of size ℓ. Query-to-communication lifting results typically show that for some gadget g, a gap between certain query complexity measures of f implies a gap between the suitable communication complexity measures of the composed function $f \circ g^n$. For the non-deterministic setting, it is shown in [14] that if the gadget g is the inner product function on vectors of length $\ell = \Theta(\log n)$, then a gap between the unambiguous 1-certificate complexity and the 0-certificate complexity for f implies a gap between the unambiguous non-deterministic communication complexity and the co-non-deterministic communication complexity for $f \circ g^n$ (see also [13, Appendix A]). The analysis uses the fact that the inner product function forms a two-source extractor, as shown by Chor and Goldreich [9].

Let M denote the matrix associated with the communication problem of $f \circ g^n$ for the function f constructed in [2] and the inner product function g. The lifting result of [14] implies that M attains a near optimal separation between $\mathrm{rank_{bin}}(M)$ and $\mathrm{rank}_{\mathbb{B}}(\overline{M})$. However, it can be seen that the matrix M is not regular at all. For example, the row and the column of M that correspond to the all-zero vector consist of only ones or only zeros, depending on the value of f on the all-zero vector.

We turn to describe how we obtain regular matrices M with a similar gap between $\mathrm{rank}_{\mathrm{bin}}(M)$ and $\mathrm{rank}_{\mathbb{B}}(\overline{M})$. We first observe that to construct a regular matrix M, it suffices to replace the inner product function in the above construction by a different gadget function g. Specifically, it turns out that if g is unbiased in a strong sense, namely, it is unbiased even while fixing one of its two inputs, then the matrix M associated with $f \circ g^n$ is regular for any function f (see Section 4.1). Hence, to obtain the desired separation on regular matrices, we provide an extension of the query-to-communication lifting theorem of [14] which allows the gadget function g to be not only the inner product function but any low-discrepancy function. We note that such an extension was speculated already in [14, Remark 1] and was actually established for the deterministic and probabilistic settings in a recent work of Chattopadhyay, Filmus, Koroth, Meir, and Pitassi [8]. Building on the approach of [14] and on tools supplied in [8], we prove that such an extension holds for the non-deterministic setting as well (for a precise statement, see Theorem 7). We proceed by showing that a slight variant g of the inner product function, studied in [4] and in [22], is unbiased in the required sense and has low discrepancy. Then, to prove Theorem 1, we apply our generalized query-to-communication lifting theorem to the family of functions f provided in [2] with this gadget g.

Let us mention that our generalized lifting theorem is not essential for the proof of Theorem 1. It turns out that the matrix M obtained using the aforementioned gadget function g has a sub-matrix that corresponds to a composition with the standard inner product function, hence the lower bound on $\mathrm{rank}_{\mathbb{B}}(\overline{M})$ can also be derived from the lifting result of [14]. Yet, the generality of our lifting theorem can be used to obtain a separation between $\mathrm{rank}_{\mathrm{bin}}(M)$ and $\mathrm{rank}_{\mathbb{B}}(\overline{M})$ using various other gadget functions, and we believe that it might find additional applications.

We finally use the regular matrices given in Theorem 1 to provide regular counterexamples for the Alon-Saks-Seymour conjecture and to prove Theorem 2. It is shown in [6] that a matrix M with $\mathrm{rank}_{\mathrm{bin}}(M)$ much smaller than $\mathrm{rank}_{\mathbb{B}}(\overline{M})$ can be transformed into a graph G with $\mathrm{bp}(G)$ much smaller than $\chi(G)$. This transformation, however, does not preserve the regularity. In fact, a natural attempt to produce a regular graph G from a regular matrix M using the approach of [6] results in a graph that is not even simple (because it has loops). Moreover, certain steps of the argument of [6] identify subgraphs of this graph G with a biclique partition number much smaller than the chromatic number, but those subgraphs are not necessarily regular even if G is. We overcome these difficulties by combining the approach of [6] with a couple of additional ideas, and show that any square regular matrix M with a large gap between $\mathrm{rank}_{\mathrm{bin}}(M)$ and $\mathrm{rank}_{\mathbb{B}}(\overline{M})$ can be transformed into a simple regular graph G with a similar gap between $\mathrm{bp}(G)$ and $\chi(G)$ (see Theorem 12).

1.3 Outline

The rest of the paper is organized as follows. In Section 2, we collect several definitions and results needed throughout the paper. In Section 3, we present our generalized query-to-communication lifting theorem in non-deterministic communication complexity. Its proof can be found in the full version of the paper. In Section 4, we present and analyze a certain gadget function, and combine it with the lifting theorem to prove Theorem 1. Finally, in Section 5, we obtain regular graphs that form counterexamples to the Alon-Saks-Seymour conjecture and confirm Theorem 2.

2 Preliminaries

2.1 Non-deterministic Communication Complexity

Let Λ be a finite set, and let $F : \Lambda \times \Lambda \to \{0,1\}$ be a function. In the communication problem associated with F, one player holds an input $x \in \Lambda$ and another player holds an input $y \in \Lambda$, and their goal is to decide whether $F(x,y) = 1$ by a communication protocol that minimizes the worst-case number of communicated bits. The $0,1$ matrix M associated with the function F is the matrix whose rows and columns are indexed by Λ, defined by $M_{x,y} = F(x,y)$ for all $x, y \in \Lambda$. Consider the following three non-deterministic communication complexity measures of a function F.

- The *non-deterministic communication complexity* of F, denoted by $\mathsf{NP}^{\mathsf{cc}}(F)$, is the smallest possible number of communicated bits in a non-deterministic communication protocol for F, that is, a protocol satisfying that $F(x,y) = 1$ if and only if there exists an accepting computation on (x,y). It holds that $\mathsf{NP}^{\mathsf{cc}}(F) = \lceil \log_2 \mathrm{rank}_{\mathbb{B}}(M) \rceil$.
- The *co-non-deterministic communication complexity* of F, denoted by $\mathsf{coNP}^{\mathsf{cc}}(F)$, is the non-deterministic communication complexity of the negation $\neg F$ of F, defined by $(\neg F)(x,y) = 1 - F(x,y)$ for all $x, y \in \Lambda$. It thus holds that $\mathsf{coNP}^{\mathsf{cc}}(F) = \lceil \log_2 \mathrm{rank}_{\mathbb{B}}(\overline{M}) \rceil$.
- A non-deterministic protocol is called *unambiguous* if it satisfies that each input has at most one accepting computation. The smallest possible number of communicated bits in such a protocol for F is referred to as the *unambiguous non-deterministic communication complexity* of F and is denoted by $\mathsf{UP}^{\mathsf{cc}}(F)$. It holds that $\mathsf{UP}^{\mathsf{cc}}(F) = \lceil \log_2 \mathrm{rank}_{\mathrm{bin}}(M) \rceil$.

2.2 Composed Functions

For integers n and ℓ, let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be two functions. The function $g^n : \{0,1\}^{\ell \cdot n} \times \{0,1\}^{\ell \cdot n} \to \{0,1\}^n$ is defined by

$$g^n(x,y) = (g(x_1,y_1), g(x_2,y_2), \ldots, g(x_n,y_n))$$

for all $x, y \in \{0,1\}^{\ell \cdot n}$, where the vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are viewed as concatenations of n blocks of size ℓ. The *composed function* $f \circ g^n : \{0,1\}^{\ell \cdot n} \times \{0,1\}^{\ell \cdot n} \to \{0,1\}$ is defined by $(f \circ g^n)(x,y) = f(g^n(x,y))$. For a set $I \subseteq [n]$ and a vector $x \in \{0,1\}^{\ell \cdot n}$, we let $x_I \in \{0,1\}^{\ell \cdot |I|}$ denote the projection of x to the blocks whose indices are in I. Note that when $I = \{i\}$ for some $i \in [n]$, we have $x_i = x_I$. For vectors $x, y \in \{0,1\}^{\ell \cdot n}$, we let $g^I(x_I, y_I)$ denote the projection of $g^n(x,y)$ to the indices of I.

2.3 Certificate Complexity

An n-variate k-*DNF* formula φ is a Boolean formula on n variables that can be written as a disjunction $\varphi = c_1 \vee \cdots \vee c_m$, where every c_i is a conjunction of at most k literals. The formula φ is said to be *unambiguous* if for every input $x \in \{0,1\}^n$ there is at most one $i \in [m]$ that satisfies $c_i(x) = 1$. For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, consider the following query complexity measures.

- The 1-*certificate complexity* of f, denoted by $\mathsf{C}_1(f)$, is the smallest integer k for which f can be written as a k-DNF formula.
- The 0-*certificate complexity* of f, denoted by $\mathsf{C}_0(f)$, is $\mathsf{C}_1(\neg f)$, where $\neg f$ is the negation of f. Equivalently, $\mathsf{C}_0(f)$ is the smallest integer k for which f can be written as a k-CNF formula.
- The *unambiguous* 1-*certificate complexity* of f, denoted by $\mathsf{UC}_1(f)$, is the smallest integer k for which f can be written as an unambiguous k-DNF formula.

We need the following result that was proved in [2].

▶ **Theorem 4** ([2]). *For infinitely many integers r, there exists a Boolean function $f : \{0,1\}^n \to \{0,1\}$ satisfying $\mathsf{UC}_1(f) = r$ and $\mathsf{C}_0(f) \geq \widetilde{\Omega}(r^2)$ where $r = n^{\Omega(1)}$.*

2.4 Discrepancy

▶ **Definition 5** (Discrepancy). *Let Λ be a finite set, let $g : \Lambda \times \Lambda \to \{0,1\}$ be a function, and let X, Y be independent random variables that are uniformly distributed over Λ. The* discrepancy *of g with respect to a combinatorial rectangle $R \subseteq \Lambda \times \Lambda$ is denoted by $\mathrm{disc}_R(g)$ and is defined by*

$$\mathrm{disc}_R(g) = \Big|\Pr\left[g(X,Y) = 0 \text{ and } (X,Y) \in R\right] - \Pr\left[g(X,Y) = 1 \text{ and } (X,Y) \in R\right]\Big|.$$

The discrepancy *of g, denoted by $\mathrm{disc}(g)$, is defined as the maximum of $\mathrm{disc}_R(g)$ over all combinatorial rectangles $R \subseteq \Lambda \times \Lambda$.*

3 Lifting from Certificate to Communication Complexity

In this section, we present our extension of the query-to-communication lifting theorem in non-deterministic communication complexity to general low-discrepancy functions. We start with a simple upper bound on the unambiguous non-deterministic communication complexity of a composed function.

▶ **Lemma 6.** *For all functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$, it holds that $\mathsf{UP}^{\mathsf{cc}}(f \circ g^n) \leq O\big(\mathsf{UC}_1(f) \cdot \max(\log_2 n, \ell)\big)$.*

Proof. Put $k = \mathsf{UC}_1(f)$. Then, the function f can be written as an unambiguous n-variate k-DNF formula $\varphi = c_1 \vee \cdots \vee c_m$ where $m \leq (2n)^k$. Consider the following non-deterministic protocol for the communication problem associated with the function $f \circ g^n$. Let $x, y \in \{0,1\}^{\ell \cdot n}$ be the inputs of the players. The first player selects non-deterministically an index $i \in [m]$ and sends it to the other player. Let $I \subseteq [n]$ denote the set of indices of the variables that appear in the clause c_i, and note that $|I| \leq k$. Then, the first player sends the projection x_I of x to the blocks of I, and similarly, the second player sends the projection y_I of y to the blocks of I. The players accept if and only if $c_i(g^I(x_I, y_I)) = 1$.

Observe that $(f \circ g^n)(x,y) = 1$ if and only if the protocol has an accepting computation on the inputs x, y. Observe further that the fact that φ is unambiguous implies that the protocol is unambiguous as well. Finally, the number of bits communicated by the protocol is $O(\log_2 m + k \cdot \ell) \leq O\big(k \cdot \max(\log_2 n, \ell)\big)$, completing the proof. ◀

We turn to state a lower bound on the co-non-deterministic communication complexity of composed functions $f \circ g^n$ for low-discrepancy functions g. Its proof can be found in the full version of the paper.

▶ **Theorem 7.** *For every $\eta > 0$ there exists $c > 0$ for which the following holds. Let ℓ and n be integers such that $\ell \geq c \cdot \log_2 n$, and let $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be a function satisfying $\mathrm{disc}(g) \leq 2^{-\eta \cdot \ell}$. Then, for every function $f : \{0,1\}^n \to \{0,1\}$, it holds that*

$$\mathsf{coNP}^{\mathsf{cc}}(f \circ g^n) \geq \Omega\big(\eta \cdot \mathsf{C}_0(f) \cdot \ell\big).$$

4 The Binary and Boolean Rank of Regular Matrices

In what follows we consider the notion of strongly unbiased functions and show that compositions with such functions are associated with regular matrices. We then present a strongly unbiased function and analyze its discrepancy. Equipped with this function, we apply the lifting theorem from the previous section to prove Theorem 1.

4.1 Strongly Unbiased Functions

Consider the following definition.

▶ **Definition 8.** *Let ℓ be an integer. We call a function $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$* strongly unbiased *if for every vector $x \in \{0,1\}^\ell$, the number of vectors $y \in \{0,1\}^\ell$ satisfying $g(x,y) = 1$ is $2^{\ell-1}$, and for every vector $y \in \{0,1\}^\ell$, the number of vectors $x \in \{0,1\}^\ell$ satisfying $g(x,y) = 1$ is $2^{\ell-1}$. Equivalently, g is strongly unbiased if the matrix associated with g is $2^{\ell-1}$-regular.*

The following lemma shows that compositions with strongly unbiased functions are associated with regular matrices.

▶ **Lemma 9.** *For all functions $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ and $f : \{0,1\}^n \to \{0,1\}$, if g is strongly unbiased then the matrix associated with the composed function $f \circ g^n$ is regular.*

Proof. Let $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be a strongly unbiased function, let $f : \{0,1\}^n \to \{0,1\}$ be a function, and let M be the matrix of dimensions $2^{\ell \cdot n} \times 2^{\ell \cdot n}$ associated with the composed function $f \circ g^n$. Since g is strongly unbiased, it follows that for every vector $x \in \{0,1\}^{\ell \cdot n}$ and for every vector $a \in \{0,1\}^n$, precisely 2^{-n} fraction of the vectors $y \in \{0,1\}^{\ell \cdot n}$ satisfy $g^n(x,y) = a$. This implies that the row of the matrix M that corresponds to a vector $x \in \{0,1\}^{\ell \cdot n}$ consists of the evaluations of the function f on all vectors $a \in \{0,1\}^n$, where each such evaluation appears exactly $2^{-n} \cdot 2^{\ell \cdot n} = 2^{(\ell-1)n}$ times. In particular, the number of ones in this row is $2^{(\ell-1)n} \cdot |f^{-1}(1)|$. Since this number is independent of x, it follows that this is the number of ones in each row of the matrix M. By symmetry, this is also the number of ones in each column of M, implying that the matrix M is regular. ◀

4.2 The Gadget Function

For an integer $\ell \geq 1$, define the function $g_\ell : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ by

$$g_\ell(x,y) = x_1 + y_1 + \sum_{i=2}^{\ell} x_i \cdot y_i \pmod 2$$

for all $x, y \in \{0,1\}^\ell$. We first observe that g_ℓ is strongly unbiased.

▶ **Lemma 10.** *For every integer $\ell \geq 1$, the function g_ℓ is strongly unbiased.*

Proof. Consider the function g_ℓ for an integer $\ell \geq 1$. By definition, for every $x \in \{0,1\}^\ell$, it holds that for every $y' \in \{0,1\}^{\ell-1}$ exactly one of the two vectors $y \in \{0,1\}^\ell$ with suffix y' satisfies $g(x,y) = 1$. This implies that for every $x \in \{0,1\}^\ell$ precisely $2^{\ell-1}$ of the vectors $y \in \{0,1\}^\ell$ satisfy $g(x,y) = 1$. By symmetry, we also have that for every $y \in \{0,1\}^\ell$ precisely $2^{\ell-1}$ of the vectors $x \in \{0,1\}^\ell$ satisfy $g(x,y) = 1$, so we are done. ◀

The following lemma claims that the functions g_ℓ have low discrepancy. We note that this can be directly derived from a bound on the discrepancy of the inner product function. Yet, we present in the full version of the paper a bound with a somewhat better multiplicative constant, borrowing an argument of Bouda, Pivoluska, and Plesch [4].

▶ **Lemma 11.** *For every integer $\ell \geq 1$, the discrepancy of the function g_ℓ satisfies*

$$\mathrm{disc}(g_\ell) \leq 2^{-(\ell+1)/2}.$$

4.3 Proof of Theorem 1

We are ready to put everything together and to complete the proof of Theorem 1.

Proof of Theorem 1. By Theorem 4, for infinitely many integers r, there exists a Boolean function $f : \{0,1\}^n \to \{0,1\}$ satisfying $\mathsf{UC}_1(f) = r$ and $\mathsf{C}_0(f) \geq \widetilde{\Omega}(r^2)$ where $r = n^{\Omega(1)}$. For an integer ℓ, consider the function $g_\ell : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ defined in Section 4.2. By Lemma 11, it holds that $\mathrm{disc}(g_\ell) \leq 2^{-\eta \cdot \ell}$ for $\eta = 1/2$. Theorem 7 yields that there exists a constant c, such that for $\ell = \lceil c \cdot \log_2 n \rceil$, the composed function $f \circ g_\ell^n$ satisfies

$$\mathsf{coNP}^{\mathsf{cc}}(f \circ g_\ell^n) \geq \Omega(\mathsf{C}_0(f) \cdot \ell) \geq \widetilde{\Omega}(r^2). \tag{2}$$

By Lemma 6, it further holds that

$$\mathsf{UP}^{\mathsf{cc}}(f \circ g_\ell^n) \leq O(\mathsf{UC}_1(f) \cdot \ell) \leq \widetilde{O}(r), \tag{3}$$

where for the second inequality we have used our choice of ℓ and the fact that $r = n^{\Omega(1)}$.

To complete the proof, let M be the square $2^{\ell \cdot n} \times 2^{\ell \cdot n}$ matrix associated with the composed function $f \circ g_\ell^n$. By Lemma 10, the function g_ℓ is strongly unbiased, hence by Lemma 9, the matrix M is regular. Recalling that $\mathsf{UP}^{\mathsf{cc}}(f \circ g_\ell^n) = \lceil \log_2 \mathrm{rank}_{\mathrm{bin}}(M) \rceil$, it follows from (3) that $\mathrm{rank}_{\mathrm{bin}}(M) \leq 2^{\widetilde{O}(r)}$. Put $k = \mathrm{rank}_{\mathrm{bin}}(M)$, and combine (2) with the fact that $\mathsf{coNP}^{\mathsf{cc}}(f \circ g_\ell^n) = \lceil \log_2 \mathrm{rank}_{\mathbb{B}}(\overline{M}) \rceil$ to obtain that $\mathrm{rank}_{\mathbb{B}}(\overline{M}) \geq 2^{\widetilde{\Omega}(r^2)} \geq k^{\widetilde{\Omega}(\log k)}$. ◀

5 The Alon-Saks-Seymour Conjecture and Regular Graphs

In this section, we prove the following theorem.

▶ **Theorem 12.** *For every square regular $0,1$ matrix M, there exists a simple regular graph G satisfying $\mathrm{bp}(G) \leq 33 \cdot \mathrm{rank}_{\mathrm{bin}}(M)^2$ and $\chi(G) \geq \mathrm{rank}_{\mathbb{B}}(\overline{M})^{1/3}$.*

Applying Theorem 12 to the matrices given by Theorem 1 yields regular graphs that form counterexamples to the Alon-Saks-Seymour conjecture with a near optimal gap between the biclique partition number and the chromatic number. This confirms Theorem 2.

5.1 Biclique Covering

We start with some definitions that will be used throughout the proof of Theorem 12. All graphs considered here are undirected. They do not contain parallel edges but they may have loops. As usual, a graph is said to be *simple* if it contains no loops and no parallel edges. For a graph $G = (V, E)$, a *biclique of* G is a complete bipartite subgraph of G, that is, a pair (A, B) of sets $A, B \subseteq V$ where every vertex of A is adjacent in G to every vertex of B. For adjacent vertices x, y of G such that $x \in A$ and $y \in B$, we say that the biclique (A, B) covers the oriented edge (x, y). Note that although the edges of G are undirected, a biclique of G covers edges of G with some orientation.

For an integer t, a *t-biclique covering* of G is a collection of bicliques of G that cover every edge of G at least once and at most t times. The minimum size of such a covering is called the *t-biclique covering number of* G and is denoted by $\mathrm{bp}_t(G)$. For $t = 1$, a 1-biclique covering is also called a *biclique partition*, and we write $\mathrm{bp}(G) = \mathrm{bp}_1(G)$.

We need the following result of Bousquet, Lagoutte, and Thomassé [6].

▷ Claim 13 ([6, Claim 28]). Let $H = (V, E)$ be a simple graph, and let $\mathcal{C}$ be a t-biclique covering of size k of H. Let $E' \subseteq E$ be the set of edges of H that are covered by $\mathcal{C}$ exactly t times. Then, the graph $H' = (V, E')$ satisfies $\mathrm{bp}(H') \leq (2k)^t$.

5.2 From Regular Matrices to Regular Graphs

We are ready to prove Theorem 12.

Proof of Theorem 12. Let M be an $n \times n$ regular $0, 1$ matrix, and let d denote the number of ones in each row and each column of M. Put $k = \mathrm{rank}_{\mathrm{bin}}(M)$ and $m = \mathrm{rank}_{\mathbb{B}}(\overline{M})$.

We first define a graph $H = (V, E)$ on the vertex set $V = [n] \times [n]$ in which every two (not necessarily distinct) vertices $(i_1, j_1), (i_2, j_2) \in V$ are adjacent if $M_{i_1,j_2} = 1$ or $M_{i_2,j_1} = 1$. Define $V_0 = \{(i, j) \in V \mid M_{i,j} = 0\}$ and $V_1 = \{(i, j) \in V \mid M_{i,j} = 1\}$. Note that $V = V_0 \cup V_1$, and notice that the vertices of H that have loops are precisely the vertices of V_1.

Let $H_0 = H[V_0]$ denote the subgraph of H induced on the vertices of V_0. Clearly, H_0 is a simple graph. The following lemma relates its chromatic number to the Boolean rank of $\overline{M}$.

▶ **Lemma 14.** *The graph H_0 satisfies $\chi(H_0) \geq m$.*

Proof. Put $r = \chi(H_0)$. Then, there exists a partition of V_0 into r independent sets $I_1, \ldots, I_r$ of H_0. For each $t \in [r]$, let A_t be the set of elements $i \in [n]$ for which there exists some $j \in [n]$ such that $(i, j) \in I_t$, and let B_t be the set of elements $j \in [n]$ for which there exists some $i \in [n]$ such that $(i, j) \in I_t$. Since I_t is an independent set in H_0, it follows that every pair $(i, j) \in A_t \times B_t$ satisfies $M_{i,j} = 0$. This implies that $A_t \times B_t$ is a combinatorial rectangle of zeros in the matrix M. Since the r given independent sets cover the entire set V_0, it follows that for every pair $(i, j) \in V_0$ there exists some $t \in [r]$ such that $(i, j) \in I_t$, and this t satisfies $(i, j) \in A_t \times B_t$. This shows that the rectangles $A_t \times B_t$ with $t \in [r]$ form a cover of the zeros of M, hence $r \geq \mathrm{rank}_{\mathbb{B}}(\overline{M}) = m$, as required. ◀

The next lemma provides a 2-biclique covering of H.

▶ **Lemma 15.** *There exists a 2-biclique covering $\mathcal{C}$ of H such that*

1. $|\mathcal{C}| = k$,
2. *for every adjacent distinct vertices $(i_1, j_1), (i_2, j_2)$ of H, if both $M_{i_1,j_2} = 1$ and $M_{i_2,j_1} = 1$ hold, then the edge that connects them is covered by $\mathcal{C}$ twice in the two opposite orientations, and if only one of them holds, then it is covered by $\mathcal{C}$ once, and*
3. *every loop of H is covered by $\mathcal{C}$ once.*

Proof. By $k = \mathrm{rank}_{\mathrm{bin}}(M)$, there exists a collection of k combinatorial rectangles $A_t \times B_t$ of ones, $t \in [k]$, that forms a partition of the ones of the matrix M. We define $\mathcal{C}$ as the collection of all bicliques of the form $C_t = (A_t \times [n], [n] \times B_t)$ for $t \in [k]$.

Let $(i_1, j_1), (i_2, j_2)$ be two (not necessarily distinct) vertices of H. If $M_{i_1,j_2} = 1$ then there exists a unique $t \in [k]$ such that $(i_1, j_2) \in A_t \times B_t$. This implies that the oriented edge $((i_1, j_1), (i_2, j_2))$ is covered by the biclique C_t and is not covered by any other biclique of $\mathcal{C}$. If, however, it holds that $M_{i_1,j_2} = 0$, then no $t \in [k]$ satisfies $(i_1, j_2) \in A_t \times B_t$, hence the oriented edge $((i_1, j_1), (i_2, j_2))$ is not covered by any biclique of $\mathcal{C}$.

We turn to show that $\mathcal{C}$ is a 2-biclique covering of H that satisfies the assertion of the lemma. By definition, we have $|\mathcal{C}| = k$, as required for Item 1. Let $(i_1, j_1), (i_2, j_2)$ be two distinct vertices of H. If the vertices are adjacent then $M_{i_1,j_2} = 1$ or $M_{i_2,j_1} = 1$. The above discussion implies that if both the conditions hold then the edge that connects them is covered twice in the two opposite orientations, whereas if only one of the conditions holds, then the edge is covered once, as required for Item 2. For a vertex (i, j) that has a loop, it holds that $M_{i,j} = 1$, hence the oriented edge $((i, j), (i, j))$ is covered once by $\mathcal{C}$, as required for Item 3. On the other hand, if the vertices $(i_1, j_1), (i_2, j_2)$ are not adjacent then $M_{i_1,j_2} = 0$ and $M_{i_2,j_1} = 0$, hence no oriented edge between them is covered by $\mathcal{C}$. It thus follows that $\mathcal{C}$ is a 2-biclique covering of H, and we are done. ◀

Let $\mathcal{C}$ be the 2-biclique covering of H given by Lemma 15. Consider the two subgraphs of H_0 defined by $H_0^{(1)} = (V_0, E_1)$ and $H_0^{(2)} = (V_0, E_2)$, where E_t is the set of edges of H_0 that are covered by $\mathcal{C}$ exactly t times for $t \in [2]$. Since the edge set of H_0 is $E_1 \cup E_2$, it follows that

$$\chi(H_0) \leq \chi(H_0^{(1)}) \cdot \chi(H_0^{(2)}). \tag{4}$$

To obtain the desired simple regular graph, we proceed by considering the following two cases according to the chromatic number of $H_0^{(2)}$.

Case 1. Suppose first that $\chi(H_0^{(2)}) \geq m^{1/3}$. Let $\mathcal{C}'$ be the collection of bicliques of H obtained from $\mathcal{C}$ by replacing every biclique $(A, B) \in \mathcal{C}$ by the three bicliques $(A \cap B, A \cap B)$, $(A \cap B, B \setminus A)$, and $(A \setminus B, B)$, where bicliques with an empty part can be avoided. Observe that these three bicliques cover precisely the same edges covered by (A, B) with the same multiplicities and orientations. Therefore, $\mathcal{C}'$ is a 2-biclique covering of H of size $|\mathcal{C}'| \leq 3k$ which satisfies Items 2 and 3 of Lemma 15. It further satisfies that each of its bicliques has either equal or disjoint parts. We let $\mathcal{C}'' \subseteq \mathcal{C}'$ denote the collection of bicliques of $\mathcal{C}'$ with equal parts. It clearly holds that $|\mathcal{C}''| \leq k$ and $|\mathcal{C}' \setminus \mathcal{C}''| \leq 2k$.

Every biclique of $\mathcal{C}''$ has the form (A, A) for some set $A \subseteq V$. For every $x \in A$, it covers a loop of x as an oriented edge (x, x), and for every distinct $x, y \in A$, it covers the edge that connects x and y in the two opposite orientations, namely, as (x, y) and as (y, x). This implies that all the vertices that appear in the bicliques of $\mathcal{C}''$ have loops in H and thus belong to V_1. Since the parts of the bicliques of $\mathcal{C}' \setminus \mathcal{C}''$ are disjoint, it follows that the bicliques of $\mathcal{C}''$ cover all the loops of H. Since $\mathcal{C}'$ is a 2-biclique covering of H that covers the loops once, it follows that no edge is covered by both $\mathcal{C}''$ and $\mathcal{C}' \setminus \mathcal{C}''$.

Let F be the graph obtained from H by removing the edges of the bicliques of $\mathcal{C}''$. Since the bicliques of $\mathcal{C}''$ cover all the loops of H, it follows that the graph F is simple. The collection $\mathcal{C}' \setminus \mathcal{C}''$ forms a 2-biclique covering of F, hence $\mathrm{bp}_2(F) \leq 2k$. Let $F^{(2)}$ denote the subgraph of F on V that includes all the edges that are covered by $\mathcal{C}' \setminus \mathcal{C}''$ twice. Since the bicliques of $\mathcal{C}''$ involve only vertices of V_1, it follows that $F^{(2)}$ has an induced subgraph isomorphic to $H_0^{(2)}$, implying that $\chi(F^{(2)}) \geq \chi(H_0^{(2)}) \geq m^{1/3}$.

Now, let G be the graph that contains two disjoint copies of $F^{(2)}$, with additional edges between the two copies according to the bicliques of $\mathcal{C}''$. More precisely, G is the graph on the vertex set $V \times [2]$ in which two vertices (x, b) and (y, b) for $b \in [2]$ are adjacent if x and y are adjacent in $F^{(2)}$, and two vertices $(x, 1)$ and $(y, 2)$ are adjacent if (x, y) is an oriented edge covered by the bicliques of $\mathcal{C}''$. The graph G is simple, because $F^{(2)}$ is simple and because no oriented edge is covered twice by $\mathcal{C}''$. We claim that G satisfies the assertion of the theorem.

Firstly, G has an induced subgraph isomorphic to $F^{(2)}$, hence it follows that $\chi(G) \geq \chi(F^{(2)}) \geq m^{1/3}$.

Secondly, we claim that $\mathrm{bp}(G) \leq 33 \cdot k^2$. To see this, use Claim 13 and $\mathrm{bp}_2(F) \leq 2k$ to obtain that $\mathrm{bp}(F^{(2)}) \leq (4k)^2$, that is, at most $(4k)^2$ bicliques are needed for a partition of the edges of each copy of $F^{(2)}$ in G. Consider further the bicliques $(A \times \{1\}, A \times \{2\})$ for $(A, A) \in \mathcal{C}''$, which form a partition with size at most k of the edges of G between the vertices of $V \times \{1\}$ and those of $V \times \{2\}$. It follows that $\mathrm{bp}(G) \leq 2 \cdot (4k)^2 + k \leq 33 \cdot k^2$.

Finally, we claim that G is regular with degree d^2. To see this, consider an arbitrary vertex $(i_1, j_1, b) \in V \times [2]$ in G. This vertex is adjacent to the vertices (i_2, j_2, b) for which the pairs (i_1, j_1) and (i_2, j_2) are adjacent in H and the edge that connects them is covered twice by $\mathcal{C}' \setminus \mathcal{C}''$. It is further adjacent to the vertices (i_2, j_2, b') with $b' \neq b$ for which the pairs (i_1, j_1) and (i_2, j_2) are adjacent in H and the edge that connects them is covered by $\mathcal{C}''$ (twice if they are distinct, and once otherwise). Since $\mathcal{C}'$ satisfies Items 2 and 3 of Lemma 15, it follows that the degree of (i_1, j_1, b) in G is precisely the number of pairs $(i_2, j_2) \in V$ satisfying $M_{i_1,j_2} = 1$ and $M_{i_2,j_1} = 1$. By the d-regularity of M, the latter is equal to d^2, so we are done.

Case 2. Suppose next that $\chi(H_0^{(2)}) < m^{1/3}$. We start by proving that there exists an independent set $S \subseteq V_0$ in the graph $H_0^{(2)}$ for which

$$\chi(H_0^{(1)}[S]) \geq m^{1/3}. \tag{5}$$

Indeed, the assumption implies that there exists a proper coloring of $H_0^{(2)}$ with fewer than $m^{1/3}$ colors. If the induced subgraph of $H_0^{(1)}$ on every color class of this coloring has chromatic number smaller than $m^{1/3}$, then one can obtain a proper coloring of $H_0^{(1)}$ whose number of colors is smaller than $m^{1/3} \cdot m^{1/3} = m^{2/3}$, which implies using (4) that $\chi(H_0) < m^{2/3} \cdot m^{1/3} = m$, in contradiction to Lemma 14. This implies that some color class $S \subseteq V_0$ of the coloring of $H_0^{(2)}$ satisfies (5).

Now, consider the 3-partite graph G' whose vertex set consists of three copies of V that are connected by three copies of the bicliques of $\mathcal{C}$ oriented in a cyclic manner. More precisely, the vertex set of G' is $V \times [3]$ and its edges are those of the bicliques $(A \times \{1\}, B \times \{2\})$, $(A \times \{2\}, B \times \{3\})$, and $(A \times \{3\}, B \times \{1\})$ for all $(A, B) \in \mathcal{C}$. By Lemma 15, no oriented edge of the bicliques of $\mathcal{C}$ is covered twice. It thus follows that G' is a simple graph and that each of its edges is covered by the above bicliques exactly once. By $|\mathcal{C}| = k$, it follows that $\mathrm{bp}(G') \leq 3k$. Further, Items 2 and 3 of Lemma 15 imply that the degree of every vertex $(i_1, j_1, b) \in V \times [3]$ of G' is precisely the sum of the number of pairs $(i_2, j_2) \in V$ satisfying $M_{i_1,j_2} = 1$ and the number of pairs $(i_2, j_2) \in V$ satisfying $M_{i_2,j_1} = 1$. Since the matrix M is d-regular, it follows that the graph G' is regular with degree $2nd$.

We next define a graph G as follows. The graph G is obtained from G' by removing all the edges whose both endpoints are in $S \times [3]$ and by adding the edges of the induced subgraph $H[S]$ of H on S to each of the three copies of S in G (i.e., $S \times \{b\}$ for $b \in [3]$). Since G' is a simple graph, using the fact that S is a subset of V_0, it follows that G is a simple graph as well. We claim that G satisfies the assertion of the theorem.

Firstly, since S is an independent set in $H_0^{(2)}$, the subgraph of G induced on every copy of S is isomorphic to $H_0^{(1)}[S]$. It thus follows from (5) that $\chi(G) \geq \chi(H_0^{(1)}[S]) \geq m^{1/3}$.

Secondly, we claim that $\mathrm{bp}(G) \leq 9k$. To see this, recall that $\mathrm{bp}(G') \leq 3k$, and consider some biclique partition with size at most $3k$ of the edges of G'. Replace each biclique $(A \times \{b\}, B \times \{b'\})$ of this partition, where $b \neq b'$, by the two bicliques

$$((A \setminus S) \times \{b\}, B \times \{b'\}) \ \text{ and } \ ((A \cap S) \times \{b\}, (B \setminus S) \times \{b'\}).$$

This gives us a biclique partition with size at most $6k$ of all the edges of G' but those spanned by the vertices of $S \times [3]$. It remains to cover the edges of the three copies of $H[S]$ in G. Since S is an independent set in $H_0^{(2)}$, each edge of $H[S]$ is covered by $\mathcal{C}$ exactly once, so by restricting the bicliques of $\mathcal{C}$ to the vertices of S, we get a biclique partition of $H[S]$ with size at most k. This gives us a biclique partition with size at most k of the edges of $G[V \times \{b\}]$ for each $b \in [3]$, implying that $\mathrm{bp}(G) \leq 6k + 3k = 9k$.

Finally, we claim that G is regular. To see this, recall that G' is regular and that G is obtained from G' by replacing the edges between the different copies of S by the corresponding edges inside the copies of S. Since those edges are covered exactly once by $\mathcal{C}$, this does not change the degrees of the vertices, yielding that the graph G is regular as well, and we are done. ◀

References

1 Kazuyuki Amano. Some improved bounds on communication complexity via new decomposition of cliques. *Discret. Appl. Math.*, 166:249–254, 2014.

2 Kaspars Balodis, Shalev Ben-David, Mika Göös, Siddhartha Jain, and Robin Kothari. Unambiguous DNFs and Alon-Saks-Seymour. In *IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS'21)*, pages 116–124. IEEE, 2021.

3 Shalev Ben-David, Pooya Hatami, and Avishay Tal. Low-sensitivity functions from unambiguous certificates. In *8th Innovations in Theoretical Computer Science Conference (ITCS'17)*, pages 28:1–28:23, 2017.

4 Jan Bouda, Matej Pivoluska, and Martin Plesch. Improving the Hadamard extractor. *Theor. Comput. Sci.*, 459:69–76, 2012.

5 Nicolas Bousquet, Aurélie Lagoutte, Frédéric Maffray, and Lucas Pastor. Decomposition techniques applied to the clique-stable set separation problem. *Discret. Math.*, 341(5):1492–1501, 2018.

6 Nicolas Bousquet, Aurélie Lagoutte, and Stéphan Thomassé. Clique versus independent set. *European J. Combinatorics*, 40:73–92, 2014.

7 Richard A. Brualdi, Rachel Manber, and Jeffrey A. Ross. On the minimum rank of regular classes of matrices of zeros and ones. *J. Combin. Theory Ser. A*, 41(1):32–49, 1986.

8 Arkadev Chattopadhyay, Yuval Filmus, Sajin Koroth, Or Meir, and Toniann Pitassi. Query-to-communication lifting using low-discrepancy gadgets. *SIAM J. Comput.*, 50(1):171–210, 2021. Preliminary version in ICALP'19.

9 Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988. Preliminary version in FOCS'85.

10 Maria Chudnovsky and Paul Seymour. Subdivided claws and the clique-stable set separation property. In *2019–20 MATRIX Annals*, volume 4, pages 483–487. Springer, 2021.

11 Sebastian M. Cioabă and Michael Tait. More counterexamples to the Alon-Saks-Seymour and rank-coloring conjectures. *Electron. J. Combinatorics*, 18(1), 2011.

12 D. de Caen, David A. Gregory, and Norman J. Pullman. The Boolean rank of zero-one matrices. *Proc. of the 3rd Caribbean Conference on Combinatorics and Computing*, pages 169–173, 1981.

13 Mika Göös. Lower bounds for clique vs. independent set. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS'15)*, pages 1066–1076, 2015.

14 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM J. Comput.*, 45(5):1835–1869, 2016. Preliminary version in STOC'15.

15 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM J. Comput.*, 47(6):2435–2450, 2018. Preliminary version in FOCS'15.

16 Ronald L. Graham and Henry O. Pollak. On the addressing problem for loop switching. *Bell Syst. Tech. J.*, 50(8):2495–2519, 1971.

17 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.

18 Kim A. S. Hefner, Teresa D. Henson, J. Richard Lundgren, and John S. Maybee. Biclique coverings of bigraphs and digraphs and minimum semiring ranks of $\{0,1\}$-matrices. *Congr. Numer.*, 71:115–122, 1990.

19 Hao Huang and Benny Sudakov. A counterexample to the Alon-Saks-Seymour conjecture and related problems. *Combinatorica*, 32(2):205–219, 2012.

20 Stasys Jukna. *Boolean Function Complexity – Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012.

21 Jeff Kahn. Recent results on some not-so-recent hypergraph matching and covering problems. In *Extremal Problems for Finite Sets*, pages 305–353. Bolyai Soc. Math. Stud., 1994.

22 Pravesh K. Kothari, Raghu Meka, and Prasad Raghavendra. Approximating rectangles by juntas and weakly-exponential lower bounds for LP relaxations of CSPs. In *Proc. of the 49th Annual ACM Symposium on Theory of Computing (STOC'17)*, pages 590–603, 2017.

23 Aurélie Lagoutte and Théophile Trunck. Clique-stable set separation in perfect graphs with no balanced skew-partitions. *Discret. Math.*, 339(6):1809–1825, 2016.

24 László Lovász and Michael E. Saks. Lattices, Möbius functions and communication complexity. In *IEEE 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 81–90, 1988.

25 Shachar Lovett. Recent advances on the log-rank conjecture in communication complexity. *Bull. EATCS*, 112, 2014.

26 Sylvia D. Monson, Norman J. Pullman, and Rolf Rees. A survey of clique and biclique coverings and factorizations of $(0,1)$-matrices. *Bull. Inst. Combin. Appl.*, 14:17–86, 1995.

27 Norman J. Pullman. Ranks of binary matrices with constant line sums. *Linear Algebra Appl.*, 104:193–197, 1988.

28 Alexander A. Razborov. The gap between the chromatic number of a graph and the rank of its adjacency matrix is superlinear. *Discret. Math.*, 108(1–3):393–396, 1992.

29 Manami Shigeta and Kazuyuki Amano. Ordered biclique partitions and communication complexity problems. *Discret. Appl. Math.*, 184:248–252, 2015.

30 Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991. Preliminary version in STOC'88.

31 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing. In *Proc. of the 11th Annual ACM Symposium on Theory of Computing (STOC'79)*, pages 209–213, 1979.

Gaps, Ambiguity, and Establishing Complexity-Class Containments via Iterative Constant-Setting

Lane A. Hemaspaandra
Department of Computer Science, University of Rochester, Rochester, NY, USA

Mandar Juvekar
Department of Computer Science, University of Rochester, Rochester, NY, USA

Arian Nadjimzadah
Department of Computer Science, University of Rochester, Rochester, NY, USA

Patrick A. Phillips
Riverside Research, Arlington, VA, USA

Abstract

Cai and Hemachandra used iterative constant-setting to prove that Few $\subseteq \oplus$P (and thus that FewP $\subseteq \oplus$P). In this paper, we note that there is a tension between the nondeterministic ambiguity of the class one is seeking to capture, and the density (or, to be more precise, the needed "nongappy"-ness) of the easy-to-find "targets" used in iterative constant-setting. In particular, we show that even less restrictive gap-size upper bounds regarding the targets allow one to capture ambiguity-limited classes. Through a flexible, metatheorem-based approach, we do so for a wide range of classes including the logarithmic-ambiguity version of Valiant's unambiguous nondeterminism class UP. Our work lowers the bar for what advances regarding the existence of infinite, P-printable sets of primes would suffice to show that restricted counting classes based on the primes have the power to accept superconstant-ambiguity analogues of UP. As an application of our work, we prove that the Lenstra–Pomerance–Wagstaff Conjecture implies that all $\mathcal{O}(\log\log n)$-ambiguity NP sets are in the restricted counting class $\mathrm{RC}_{\mathrm{PRIMES}}$.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases structural complexity theory, computational complexity theory, ambiguity-limited NP, counting classes, P-printable sets

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.57

Funding Work supported in part by NSF Grant CCF-2006496.

Acknowledgements We thank B. Carleton, H. Welles, and the anonymous referees.

1 Introduction

We show that every NP set of low ambiguity belongs to broad collections of restricted counting classes.

We now describe the two types of complexity classes just mentioned. For any set $S \subseteq \mathbb{N}^+$, the restricted counting class RC_S [7] is defined by $\mathrm{RC}_S = \{L \mid (\exists f \in \#\mathrm{P})(\forall x \in \Sigma^*)[(x \notin L \implies f(x) = 0) \wedge (x \in L \implies f(x) \in S)]\}$. That is, a set L is in RC_S exactly if there is a nondeterministic polynomial-time Turing machine (NPTM) that on each string not in L has zero accepting paths and on each string in L has a number of accepting paths that belongs to the set S. For example, though this is an extreme case, NP $= \mathrm{RC}_{\mathbb{N}^+}$.

In the 1970s, Valiant started the study of ambiguity-limited versions of NP by introducing the class UP [36], unambiguous polynomial time, which in the above notation is simply $\mathrm{RC}_{\{1\}}$. (The ambiguity (limit) of an NPTM refers to an upper bound on how many *accepting*

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 57; pp. 57:1–57:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of containment results. (Theorem 4.19 also gives a slightly stronger form of the 2^n-nongappiness result than the version stated here.)

If $T \subseteq \mathbb{N}^+$ X, then Y

X	Y	Reference
has an $(n + \mathcal{O}(1))$-nongappy, P-printable subset	$\mathrm{FewP} \subseteq \mathrm{RC}_T$	[7]
has an $\mathcal{O}(n)$-nongappy, P-printable subset	$\mathrm{UP}_{\leq \mathcal{O}(\log n)} \subseteq \mathrm{RC}_T$	Thm. 4.10
has an $\mathcal{O}(n \log n)$-nongappy, P-printable subset	$\mathrm{UP}_{\leq \mathcal{O}(\sqrt{\log n})} \subseteq \mathrm{RC}_T$	Thm. 4.19
for some real number $k > 1$ has an n^k-nongappy, P-printable subset	$\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{\log \log n}{2 \log k}} \subseteq \mathrm{RC}_T$	Thm. 4.13
has an $n^{\log n}$-nongappy, P-printable subset	$\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{1}{2} \log \log \log n} \subseteq \mathrm{RC}_T$	Thm. 4.19
has an $n^{(\log n)^{\mathcal{O}(1)}}$-nongappy, P-printable subset	$\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{1}{3} \log \log \log \log n} \subseteq \mathrm{RC}_T$	Thm. 4.19
has a 2^n-nongappy, P-printable subset S	$\mathrm{UP}_{\leq \max(1, \lfloor \frac{\log^*(n) - \log^*(\log^*(n)+1) - 1}{\lambda} \rfloor)}$ $\subseteq \mathrm{RC}_T$, where $\lambda = 4 + \min_{s \in S, \lvert s \rvert \geq 2}(\lvert s \rvert)$	Thm. 4.19
is infinite	$\mathrm{UP}_{\leq \mathcal{O}(1)} \subseteq \mathrm{RC}_T$	Cor. 4.4

paths it has as a function of the input's length. An NP language falls within a given level of ambiguity if it is accepted by some NPTM that happens to satisfy that ambiguity limit.) More generally, for each function $f : \mathbb{N} \to \mathbb{N}^+$ or $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$, $\mathrm{UP}_{\leq f(n)}$ denotes the class of languages L for which there is an NPTM N such that, for each x, if $x \notin L$ then N on input x has no accepting paths, and if $x \in L$ then $1 \leq \#\mathrm{acc}_N(x) \leq \lfloor f(|x|) \rfloor$ (where $\#\mathrm{acc}_N(x)$ denotes the number of accepting computation paths of N on input x). (Since, for all N and x, $\#\mathrm{acc}_N(x) \in \mathbb{N}$, the class $\mathrm{UP}_{\leq f(n)}$ just defined would be unchanged if $\lfloor f(|x|) \rfloor$ were replaced by $f(|x|)$.) Ambiguity-limited nondeterministic classes whose ambiguity limits range from completely unambiguous ($\mathrm{UP}_{\leq 1}$, i.e., UP) to polynomial ambiguity (Allender and Rubinstein's class FewP [3]) have been defined and studied.

In this paper, we show that many ambiguity-limited counting classes – including ones based on types of logarithmic ambiguity, loglog ambiguity, logloglog ambiguity, and loglogloglog ambiguity – are contained in various collections of restricted counting classes. We do so primarily through two general theorems (Theorems 4.7 and 4.12) that help make clear how, as the size of the "holes" allowed in the sets underpinning the restricted counting classes becomes smaller (i.e., as the sets become more "nongappy"), one can handle more ambiguity. Table 1 summarizes our results about the containment of ambiguity-limited counting classes in restricted counting classes.

Only for polynomial ambiguity was a result of this sort previously known. In particular, Beigel, Gill, and Hertrampf [5], strengthening Cai and Hemachandra's result $\mathrm{FewP} \subseteq \oplus\mathrm{P}$ [13], proved that $\mathrm{FewP} \subseteq \mathrm{RC}_{\{1,3,5,\ldots\}}$, and Borchert, Hemaspaandra, and Rothe [7] noted that $\mathrm{FewP} \subseteq \mathrm{RC}_T$ for each nonempty set $T \subseteq \mathbb{N}^+$ that has an easily presented (formally, P-printable [25], whose definition will be given in Section 2) subset V that is $(n+\mathcal{O}(1))$-nongappy (i.e., for some k the set V never has more than k adjacent, empty lengths; that is, for each collection of $k+1$ adjacent lengths, V will always contain at least one string whose length is one of those $k+1$ lengths).

Our proof approach in the present paper connects somewhat interestingly to the history just mentioned. We will describe in Section 4 the approach that we will call *the iterative constant-setting technique*. However, briefly put, that refers to a process of sequentially setting a series of constants – first c_0, then c_1, then c_2, ..., and then c_m – in such a way that, for each $0 \leq j \leq m$, the summation $\sum_{0 \leq \ell \leq j} c_\ell \binom{j}{\ell}$ falls in a certain "yes" or "no" target set, as required by the needs of the setting. For RC_S classes, the "no" target set will be $\{0\}$ and the "yes" target set will be S. In this paper, we will typically put sets into restricted counting classes by building Turing machines that guess (for each $0 \leq \ell \leq j$) cardinality-ℓ sets of accepting paths of another NPTM and then amplify each such successful accepting-path-set guess by – via splitting/cloning of the path – creating from it c_ℓ accepting paths.

A technically novel aspect of the proofs of the two main theorems (Theorems 4.7 and 4.12, each in effect a metatheorem) is that those proofs each provide, in a unified way for a broad class of functions, an analysis of value-growth in the context of iterated functions.

Cai and Hemachandra's [13] result FewP $\subseteq \oplus$P was proven (as was an even more general result about a class known as "Few") by the iterative constant-setting technique. Beigel, Gill, and Hertrampf [5], while generously noting that "this result can also be obtained by a close inspection of Cai and Hemachandra's proof," proved the far stronger result FewP $\subseteq \mathrm{RC}_{\{1,3,5,\ldots\}}$ simply and directly rather than by iterative constant-setting. Borchert, Hemaspaandra, and Rothe's [7] even more general result, noted above for its proof, resurrected the iterative constant-setting technique, using it to understand one particular level of ambiguity. This present paper is, in effect, an immersion into the far richer world of possibilities that the iterative constant-setting technique can offer, if one puts in the work to analyze and bound the growth rates of certain constants central to the method. In particular, as noted above we use the iterative constant-setting method to obtain a broad range of results (see Table 1) regarding how ambiguity-limited nondeterminism is not more powerful than appropriately nongappy restricted counting classes.

Each of our results has immediate consequences regarding the power of the primes as a restricted-counting acceptance type. Borchert, Hemaspaandra, and Rothe's result implies that if the set of primes has an $(n + \mathcal{O}(1))$-nongappy, P-printable subset, then FewP $\subseteq \mathrm{RC}_{\mathrm{PRIMES}}$. However, it is a long-open research issue whether there exists *any* infinite, P-printable subset of the primes, much less an $(n + \mathcal{O}(1))$-nongappy one. Our results lower the bar on what one must assume about how nongappy hypothetical infinite, P-printable subsets of the primes are in order to imply that some superconstant-ambiguity-limited nondeterministic version of NP is contained in $\mathrm{RC}_{\mathrm{PRIMES}}$. We prove that even infinite, P-printable sets of primes with merely exponential upper bounds on the size of their gaps would yield such a result. We also prove – by exploring the relationship between density and nongappiness – that the Lenstra–Pomerance–Wagstaff Conjecture [35, 38] (regarding the asymptotic density of the Mersenne primes) implies that $\mathrm{UP}_{\leq \mathcal{O}(\log\log n)} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$. The Lenstra–Pomerance–Wagstaff Conjecture is characterized in Wikipedia [41] as being "widely accepted," the fact that it disagrees with a different conjecture (Gillies' Conjecture [22]) notwithstanding.

Additional results, discussions and comments, and the omitted proofs of Theorems 4.3, 4.12, 4.13, 4.14, and 4.19, Propositions 2.5, 4.9, and 4.17, and Corollary 4.15 can be found our full technical report version [26].

2 Definitions

$\mathbb{N} = \{0, 1, 2, \ldots\}$. $\mathbb{N}^+ = \{1, 2, \ldots\}$. Each positive natural number, other than 1, is prime or composite. A prime number is a number that has no positive divisors other than 1 and itself. PRIMES $= \{i \in \mathbb{N} \mid i \text{ is a prime}\} = \{2, 3, 5, 7, 11, \ldots\}$. A composite number is one that has at least one positive divisor other than 1 and itself; COMPOSITES $= \{i \in \mathbb{N} \mid i \text{ is a composite number}\} = \{4, 6, 8, 9, 10, 12, \ldots\}$. $\mathbb{R}$ is the set of all real numbers,

$\mathbb{R}^+ = \{x \in \mathbb{R} \mid x > 0\}$, and $\mathbb{R}^{\geq 1} = \{x \in \mathbb{R} \mid x \geq 1\}$. All logs in this paper (thus those involved in log, loglog, logloglog, loglogloglog, and $\log^{[i]}$, and also those called within the definitions of $\log^*$ and our new $\log^{\circledast}$) are base 2. Also, each call of the log function in this paper, $\log(\cdot)$, is implicitly a shorthand for $\log(\max(1, \cdot))$. We do this so that formulas such as $\log\log\log(\cdot)$ do not cause domain problems on small inputs. (Admittedly, this is also distorting log in the domain-valid open interval (0,1). However, that interval never comes into play in our paper except incidentally when iterated logs drop something into it, and also in the definitions of $\log^*$ and $\log^{\circledast}$ but in those two cases – see the discussion in Footnotes 2 and 8 of [26] – the max happens not to change what those evaluate to on (0,1).)

As mentioned earlier, for any NPTM N and any string x, $\#\mathrm{acc}_N(x)$ will denote the number of accepting computation paths of N on input x. #P [37] is the counting version of NP: $\#\mathrm{P} = \{f : \Sigma^* \to \mathbb{N} \mid (\exists \text{ NPTM } N)(\forall x \in \Sigma^*)[\#\mathrm{acc}_N(x) = f(x)]\}$. $\oplus$P ("Parity P") is the class of sets L such that there is a function $f \in \#\mathrm{P}$ such that, for each string x, it holds that $x \in L \iff f(x) \equiv 1 \pmod 2$ [34, 23].

We will use $\mathcal{O}$ in its standard sense, namely, if f and g are functions (from whose domain negative numbers are typically excluded), then we say $f(n) = \mathcal{O}(g(n))$ exactly if there exist positive integers c and n_0 such that $(\forall n \geq n_0)[f(n) \leq cg(n)]$. We sometimes will also, interchangeably, speak of or write a $\mathcal{O}$ expression as representing a set of functions (e.g., writing $f(n) \in \mathcal{O}(g(n))$) [10, 11], which in fact is what the "big O" notation truly represents.

The notions RC_S, UP, and $\mathrm{UP}_{\leq f(n)}$ are as defined in Section 1. For each $k \geq 1$, Watanabe [39] implicitly and Beigel [4] explicitly studied the constant-ambiguity classes $\mathrm{RC}_{\{1,2,3,\ldots,k\}}$ which, following the notation of Lange and Rossmanith [32], we will usually denote $\mathrm{UP}_{\leq k}$. We extend the definition of $\mathrm{UP}_{\leq f(n)}$ to classes of functions as follows. For classes $\mathcal{F}$ of functions mapping $\mathbb{N}$ to $\mathbb{N}^+$ or $\mathbb{N}$ to $\mathbb{R}^{\geq 1}$, we define $\mathrm{UP}_{\leq \mathcal{F}} = \bigcup_{f \in \mathcal{F}} \mathrm{UP}_{\leq f(n)}$. We mention that the class $\mathrm{UP}_{\leq \mathcal{O}(1)}$ is easily seen to be equal to $\bigcup_{k \in \mathbb{N}^+} \mathrm{UP}_{\leq k}$, which is a good thing since that latter definition of the notion is how $\mathrm{UP}_{\leq \mathcal{O}(1)}$ was defined in the literature more than a quarter of a century ago [29]. $\mathrm{UP}_{\leq \mathcal{O}(1)}$ can be (informally) described as the class of all sets acceptable by NPTMs with constant-bounded ambiguity. Other related classes will also be of interest to us. For example, $\mathrm{UP}_{\leq \mathcal{O}(\log n)}$ captures the class of all sets acceptable by NPTMs with logarithmically-bounded ambiguity. Allender and Rubinstein [3] introduced and studied FewP, the polynomial-ambiguity NP languages, which can be defined by $\mathrm{FewP} = \{L \mid (\exists \text{ polynomial } f)[L \in \mathrm{UP}_{\leq f(n)}]\}$.

The $\mathrm{UP}_{\leq f(n)}$ classes, which will be central to this paper's study, capture ambiguity-bounded versions of NP. They are also motivated by the fact that they completely characterize the existence of ambiguity-bounded (complexity-theoretic) one-way functions.[1]

▶ **Proposition 2.1.** *Let f be any function mapping from $\mathbb{N}$ to $\mathbb{N}^+$. $\mathrm{P} \neq \mathrm{UP}_{\leq f(n)}$ if and only if there exists an $f(n)$-to-one one-way function.*

[1] A (possibly nontotal) function g is said to be a one-way function exactly if (a) g is polynomial-time computable, (b) g is honest (i.e., there exists a polynomial q such that, for each y in the range of g, there exists a string x such that $g(x) = y$ and $|x| \leq q(|y|)$; simply put, each string y mapped to by g is mapped to by some string x that is not much longer than y), and (c) g is not polynomial-time invertible (i.e., there exists no (possibly nontotal) polynomial-time function h such that for each y in the range of g, it holds that $h(y)$ is defined and $g(h(y))$ is defined and $g(h(y)) = y$) [24]. For each $f : \mathbb{N} \to \mathbb{N}^+$ and each (possibly nontotal) function $g : \Sigma^* \to \Sigma^*$, we say that g is $f(n)$-to-one exactly if, for each $y \in \Sigma^*$, $\|\{x \mid g(x) = y\}\| \leq f(|y|)$. When g is a one-way function, the function f is sometimes referred to as an ambiguity limit on the function g, and the special case of $f(n) = 1$ is the case of unambiguous one-way functions. (This is a different notion of ambiguity than that used for NPTMs, though Proposition 2.1 shows that the notions are closely connected.)

That claim holds even if f is not nondecreasing, and holds even if f is not a computable function. To the best of our knowledge, Proposition 2.1 has not been stated before for the generic case of any function $f : \mathbb{N} \to \mathbb{N}^+$. However, many concrete special cases are well known, and the proposition follows from the same argument as is used for those (see for example [27, Proof of Theorem 2.5] for a tutorial presentation of that type of argument). In particular, the proposition's special cases are known already for UP (due to [24, 30]), $\mathrm{UP}_{\leq k}$ (for each $k \in \mathbb{N}^+$) and $\mathrm{UP}_{\leq \mathcal{O}(1)}$ (in [29, 6]), FewP (in [3]), and (since the following is another name for NP) $\mathrm{UP}_{\leq 2^{n^{\mathcal{O}(1)}}}$ (folklore, see [27, Theorem 2.5, Part 1]). The proposition holds not just for single functions f, but also for classes that are collections of functions, e.g., $\mathrm{UP}_{\leq \mathcal{O}(\log n)}$.

For any function f, we use $f^{[n]}$ to denote function iteration: $f^{[0]}(\alpha) = \alpha$ and inductively, for each $n \in \mathbb{N}$, $f^{[n+1]}(\alpha) = f(f^{[n]}(\alpha))$. For each real number $\alpha \geq 0$, $\log^*(\alpha)$ ("(base 2) log star of α") is the smallest natural number k such that $\log^{[k]}(\alpha) \leq 1$. Although the logarithm of 0 is not defined, note that $\log^*(0)$ is well-defined, namely it is 0 since $\log^{[0]}(0) = 0$.

A set L is said to be P-printable [25] exactly if there is a deterministic polynomial-time Turing machine such that, for each $n \in \mathbb{N}$, the machine when given as input the string 1^n prints (in some natural coding, such as printing each of the strings of L in lexicographical order, inserting the character # after each) exactly the set of all strings in L of length less than or equal to n.

Notions of whether a set has large empty expanses between one element and the next will be central to our work in this paper. Borchert, Hemaspaandra, and Rothe [7] defined and used such a notion, in a way that is tightly connected to our work. We present here the notion they called "nongappy," but here, we will call it "nongappy$_{\text{value}}$" to distinguish their value-centered definition from the length-centered definitions that will be our norm in this paper.

▶ **Definition 2.2** ([7]). *A set $S \subseteq \mathbb{N}^+$ is said to be nongappy$_{value}$ if $S \neq \emptyset$ and $(\exists k > 0)(\forall m \in S)(\exists m' \in S)[m' > m \wedge m'/m \leq k]$.*

This says that the gaps between one element of the set and the next greater one are, as to the *values* of the numbers, bounded by a multiplicative constant. Note that, if we view the natural numbers as naturally coded in binary, that is equivalent to saying that the gaps between one element of the set and the next greater one are, as to the *lengths* of the two strings, bounded by an additive constant. That is, a nonempty set $S \subseteq \mathbb{N}^+$ is said to be nongappy$_{\text{value}}$ by this definition if the gaps in the lengths of elements of S are bounded by an additive constant, and thus we have the following result that clearly holds.

▶ **Proposition 2.3.** *A set $S \subseteq \mathbb{N}^+$ is nongappy$_{value}$ if and only if $S \neq \emptyset$ and $(\exists k > 0)(\forall m \in S)(\exists m' \in S)[m' > m \wedge |m'| \leq |m| + k]$.*

In Section 4 we define other notions of nongappiness that allow larger gaps than the above does. We will always focus on lengths, and so we will consistently use the term "nongappy" in our definitions to speak of gaps quantified in terms of the *lengths* of the strings involved. We now introduce a new notation for the notion nongappy$_{\text{value}}$, and show that our definition does in fact refer to the same notion as that of Borchert, Hemaspaandra, and Rothe.

▶ **Definition 2.4.** *A set $S \subseteq \mathbb{N}^+$ is $(n + \mathcal{O}(1))$-nongappy if $S \neq \emptyset$ and $(\exists f \in \mathcal{O}(1))(\forall m \in S)(\exists m' \in S)[m' > m \wedge |m'| \leq |m| + f(|m|)]$.*

While at first glance this might seem to be different from Borchert, Hemaspaandra, and Rothe's definition, it is easy to see that both definitions are equivalent.

▶ **Proposition 2.5.** *A set S is $(n + \mathcal{O}(1))$-nongappy if and only if it is nongappy$_{value}$.*

3 Related Work

The most closely related work has already largely been covered in the nonappendix part of the paper, but we now briefly mention that work and its relationship to this paper. In particular, the most closely related papers are the work of Cai and Hemachandra [13], Hemaspaandra and Rothe [28], and Borchert, Hemaspaandra, and Rothe [7], which introduced and studied the iterative constant-setting technique as a tool for exploring containments of counting classes. The former two (and also the important related work of Borchert and Stephan [8]) differ from the present paper in that they are not about restricted counting classes, and unlike the present paper, Borchert, Hemaspaandra, and Rothe's paper, as to containment of ambiguity-limited classes, addresses only FewP. (It is known that FewP is contained in the class known as SPP and is indeed so-called SPP-low [31, 17, 18], however that does not make our containments in restricted counting classes uninteresting, as it seems unlikely that SPP is contained in *any* restricted counting class, since SPP's "no" case involves potentially exponential numbers of accepting paths, not zero such paths.) The interesting, recent paper of Cox and Pay [16] draws on the result of Borchert, Hemaspaandra, and Rothe [7] that appears as our Theorem 4.1 to establish that FewP $\subseteq \mathrm{RC}_{\{2^t-1 \,|\, t\in\mathbb{N}^+\}}$ (note that the right-hand side is the restricted counting class defined by the Mersenne numbers), a result that itself implies FewP $\subseteq \mathrm{RC}_{\{1,3,5,\ldots\}}$.

"RC" (restricted counting) classes [7] are central to this paper. The literature's earlier "CP" classes [12] might at first seem similar, but they don't restrict rejection to the case of having zero accepting paths. Leaf languages [9], a different framework, do have flexibility to express "RC" classes, and so are an alternate notation one could use, though in some sense they would be overkill as a framework here due to their extreme descriptive power. The class $\mathrm{RC}_{\{1,3,5,\ldots\}}$ first appeared in the literature under the name $\mathrm{ModZ_2P}$ [5]. Ambiguity-limited classes are also quite central to this paper, and among those we study (see Section 2) are ones defined, or given their notation that we use, in the following papers: [36, 4, 39, 3, 32].

P-printability is due to Hartmanis and Yesha [25]. Allender [2] established a sufficient condition, which we will discuss later, for the existence of infinite, P-printable subsets of the primes. As discussed in the text right after Corollary 4.2 and in Footnote 2, none of the results of Ford, Maynard, Tao, and others [20, 33, 19] about "infinitely often" lower bounds on gaps in the primes, nor any possible future bounds, can possibly be strong enough to be the sole obstacle to a FewP $\subseteq \mathrm{RC}_{\mathrm{PRIMES}}$ construction.

4 Gaps, Ambiguity, and Iterative Constant-Setting

What is the power of NPTMs whose number of accepting paths is 0 for each string not in the set and is a prime for each string in the set? In particular, does that class, $\mathrm{RC}_{\mathrm{PRIMES}}$, contain FewP or, for that matter, any interesting ambiguity-limited nondeterministic class? That is the question that motivated this work.

Why might one hope that $\mathrm{RC}_{\mathrm{PRIMES}}$ might contain some ambiguity-limited classes? Well, we clearly have that NP $\subseteq \mathrm{RC}_{\mathrm{COMPOSITES}}$, so having the composites as our acceptance targets allows us to capture all of NP. Why? For any NP machine N, we can make a new machine N' that mimics N, except it clones each accepting path into four accepting paths, and so when N has zero accepting paths N' has zero accepting paths, and when N has at least one accepting path N' has a composite number of accepting paths.

On the other hand, why might one suspect that interesting ambiguity-limited nondeterministic classes such as FewP might *not* be contained in $\mathrm{RC}_{\mathrm{PRIMES}}$? Well, it is not even clear that FewP is contained in the class of sets that are accepted by NPTMs that accept via having a prime number of accepting paths, and reject by having a nonprime number of accepting paths (rather than being restricted to rejecting only by having zero accepting paths, as is $\mathrm{RC}_{\mathrm{PRIMES}}$). That is, even a seemingly vastly more flexible counting class does not seem to in any obvious way contain FewP.

This led us to revisit the issue of identifying the sets $S \subseteq \mathbb{N}^+$ that satisfy $\mathrm{FewP} \subseteq \mathrm{RC}_S$, studied previously by, for example, Borchert, Hemaspaandra, and Rothe [7] and Cox and Pay [16]. In particular, Borchert, Hemaspaandra, and Rothe showed, by the iterative constant-setting technique, the following theorem. From it, we immediately have Cor. 4.2.

▶ **Theorem 4.1** ([7, Theorem 3.4]). *If $T \subseteq \mathbb{N}^+$ has an $(n + \mathcal{O}(1))$-nongappy, P-printable subset, then* $\mathrm{FewP} \subseteq \mathrm{RC}_T$.

▶ **Corollary 4.2.** *If* PRIMES *contains an $(n + \mathcal{O}(1))$-nongappy, P-printable subset, then* $\mathrm{FewP} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$.

Does PRIMES contain an $(n+\mathcal{O}(1))$-nongappy, P-printable subset? The Bertrand–Chebyshev Theorem [15] states that for each natural number $k > 3$, there exists a prime p such that $k < p < 2k - 2$. Thus PRIMES clearly has an $(n + \mathcal{O}(1))$-nongappy subset.[2] Indeed, since – with p_i denoting the ith prime – $(\forall \epsilon > 0)(\exists N)(\forall n > N)[p_{n+1} - p_n < \epsilon p_n]$ [40], it holds that represented in binary there are primes at all but a finite number of bit-lengths. Unfortunately, to the best of our knowledge it remains an open research issue whether there exists *any* infinite, P-printable subset of the primes, much less one that in addition is $(n + \mathcal{O}(1))$-nongappy. In fact, the best sufficient condition we know of for the existence of an infinite, P-printable set of primes is a relatively strong hypothesis of Allender [2, Corollary 32 and the comment following it] about the probabilistic complexity class R [21] and the existence of secure extenders. However, that result does not promise that the infinite, P-printable set of primes is $(n + \mathcal{O}(1))$-nongappy – not even now, when it is known that primality is not merely in the class R but even is in the class P [1].

So the natural question to ask is: Can we at least lower the bar for what strength of advance – regarding the existence of P-printable sets of primes and the nongappiness of such sets – would suffice to allow $\mathrm{RC}_{\mathrm{PRIMES}}$ to contain some interesting ambiguity-limited class?

In particular, the notion of nongappiness used in Theorem 4.1 above means that our length gaps between adjacent elements of our P-printable set must be bounded by an additive constant. Can we weaken that to allow larger gaps, e.g., gaps of multiplicative constants, and still have containment for some interesting ambiguity-limited class?

We show that the answer is yes. More generally, we show that there is a tension and trade-off between gaps and ambiguity. As we increase the size of gaps we are willing to tolerate, we can prove containment results for restrictive counting classes, but of increasingly small levels of ambiguity. On the other hand, as we lower the size of the gaps we are willing to tolerate, we increase the amount of ambiguity we can handle.

[2] We mention in passing that it follows from the fact that PRIMES clearly *does* have an $(n + \mathcal{O}(1))$-nongappy subset that none of the powerful results by Ford, Maynard, Tao, and others [20, 33, 19] about "infinitely often" lower bounds for gaps in the primes, or in fact any results purely about lower bounds on gaps in the primes, can possibly prevent there from being a set of primes whose gaps are small enough that the set could, if sufficiently accessible, be used in a Cai–Hemachandra-type iterative constant-setting construction seeking to show that $\mathrm{FewP} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$. (In fact – keeping in mind that the difference between the value of a number and its coded length is exponential – the best such gaps known are almost exponentially too weak to preclude a Cai–Hemachandra-type iterative constant-setting construction.) Rather, the only obstacle will be the issue of whether there is such a set that in addition is computationally easily accessible/thin-able, i.e., whether there is such an $(n + \mathcal{O}(1))$-nongappy subset of the primes that is P-printable.

It is easy to see that the case of constant-ambiguity nondeterminism is so extreme that the iterative constant-setting method works for all infinite sets regardless of how nongappy they are. (It is even true that the containment $\mathrm{UP}_{\leq k} \subseteq \mathrm{RC}_T$ holds for some finite sets T, such as $\{1, 2, 3, \ldots, k\}$; but our point here is that it holds for *all* infinite sets $T \subseteq \mathbb{N}^+$.)

▶ **Theorem 4.3.** *For each infinite set $T \subseteq \mathbb{N}^+$ and for each natural $k \geq 1$, $\mathrm{UP}_{\leq k} \subseteq \mathrm{RC}_T$.*

Theorem 4.3 should be compared with the discussion by Hemaspaandra and Rothe [28, p. 210] of an NP-many-one-hardness result of Borchert and Stephan [8] and a $\mathrm{UP}_{\leq k}$-1-truth-table-hardness result. In particular, both those results are in the *un*restricted setting, and so neither implies Theorem 4.3. The proof of Theorem 4.3 can be found as Appendix A of our [26]. However, we recommend that the reader read it, if at all, only after reading the proof of Theorem 4.7, whose proof also uses (and within this paper, is the key presentation of) iterative constant-setting, and is a more interesting use of that approach.

▶ **Corollary 4.4.** *For each infinite set $T \subseteq \mathbb{N}^+$, $\mathrm{UP}_{\leq \mathcal{O}(1)} \subseteq \mathrm{RC}_T$.*

▶ **Corollary 4.5.** $\mathrm{UP}_{\leq \mathcal{O}(1)} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$.

So constant-ambiguity nondeterminism can be done by the restrictive counting class based on the primes. However, what we are truly interested in is whether we can achieve a containment for superconstant levels of ambiguity. We in fact can do so, and we now present such results for a range of cases between constant ambiguity ($\mathrm{UP}_{\leq \mathcal{O}(1)}$) and polynomial ambiguity (FewP). We first define a broader notion of nongappiness.

▶ **Definition 4.6.** *Let F be any function mapping $\mathbb{R}^+$ to $\mathbb{R}^+$. A set $S \subseteq \mathbb{N}^+$ is F-nongappy if $S \neq \emptyset$ and $(\forall m \in S)(\exists m' \in S)[m' > m \wedge |m'| \leq F(|m|)]$.*[3]

This definition sets F's domain and codomain to include real numbers, despite the fact that the underlying F-nongappy set S is of the type $S \subseteq \mathbb{N}^+$. The codomain is set to include real numbers because many notions of nongappiness we examine rely on non-integer values. Since we are often iterating functions, we thus set F's domain to be real numbers as well. Doing so does not cause problems as to computability because F is a function that is never actually computed by the Turing machines in our proofs; it is merely one that is mathematically reasoned about in the analysis of the nongappiness of sets underpinning restricted counting classes.

The following theorem generalizes the iterative constant-setting technique that Borchert, Hemaspaandra, and Rothe used to prove Theorem 4.1.

▶ **Theorem 4.7.** *Let F be a function mapping from $\mathbb{R}^+$ to $\mathbb{R}^+$ and let n_0 be a positive natural number such that F restricted to the domain $\{t \in \mathbb{R}^+ \mid t \geq n_0\}$ is nondecreasing and for all $t \geq n_0$ we have (a) $F(t) \geq t + 2$ and (b) $(\forall c \in \mathbb{N}^+)[cF(t) \geq F(ct)]$. Let j be a function, mapping from $\mathbb{N}$ to $\mathbb{N}^+$, that is at most polynomial in the value of its input and is computable in time polynomial in the value of its input. Suppose $T \subseteq \mathbb{N}^+$ has an F-nongappy, P-printable subset S. Let $\lambda = 4 + |s|$ where s is the smallest element of S with $|s| \geq n_0$. If for some $\beta \in \mathbb{N}^+$, $F^{[j(n)]}(\lambda) = \mathcal{O}(n^\beta)$, then $\mathrm{UP}_{\leq j(n)} \subseteq \mathrm{RC}_T$.*

[3] In two later definitions, 4.8 and 4.18, we apply Definition 4.6 to classes of functions. In each case, we will directly define that, but in fact will do so as the natural lifting (namely, saying a set is $\mathcal{F}$-nongappy exactly if there is an $F \in \mathcal{F}$ such that the set is F-nongappy). The reason we do not directly define lifting as applying to all classes $\mathcal{F}$ is in small part that we need it only in those two definitions, and in large part because doing so could cause confusion, since an earlier definition (Def. 2.4) that is connecting to earlier work is using as a syntactic notation an expression that itself would be caught up by such a lifting (though the definition given in Def. 2.4 is consistent with the lifting reading, give or take the fact that we've now broadened our focus to the reals rather than the naturals).

This theorem has a nice interpretation: a sufficient condition for an ambiguity-limited class $\mathrm{UP}_{\leq j(n)}$ to be contained in a particular restricted counting class is for there to be at least $j(n)$ elements that are reachable in polynomial time in an F-nongappy subset of the set that defines the counting class, assuming that the nongappiness of the counting class and the ambiguity of the $\mathrm{UP}_{\leq j(n)}$ class satisfy the above conditions.

Proof of Theorem 4.7. Let F, j, n_0, T, and S be as per the theorem statement. Suppose $(\exists \beta' \in \mathbb{N}^+)[F^{[j(n)]}(\lambda) = \mathcal{O}(n^{\beta'})]$, and fix a value $\beta \in \mathbb{N}^+$ such that $F^{[j(n)]}(\lambda) = \mathcal{O}(n^{\beta})$.

We start our proof by defining three sequences of constants that will be central in our iterative constant-setting argument, and giving bounds on their growth. Set c_1 to be the least element of S with $|c_1| \geq n_0$. For $n \in \{2, 3, \ldots\}$, given $c_1, c_2, \ldots, c_{n-1}$, we set

$$b_n = \sum_{1 \leq \ell \leq n-1} c_\ell \binom{n}{\ell}. \tag{1}$$

With b_n set, we define a_n to be the least element of S such that $a_n \geq b_n$. Finally, we set $c_n = a_n - b_n$. We now show that $\max_{1 \leq \ell \leq j(n)} |a_\ell|$ and $\max_{1 \leq \ell \leq j(n)} |c_\ell|$ are both at most polynomial in n. Take any $i \in \{2, 3, \ldots\}$. By the construction above and since S is F-nongappy, we have $|c_i| \leq |a_i| \leq F(|b_i|)$. Using our definition of b_i from Eq. 1 we get $b_i = \sum_{1 \leq k \leq i-1} c_k \binom{i}{k} \leq (i-1)(\max_{1 \leq k \leq i-1} c_k)\binom{i}{\lceil \frac{i}{2} \rceil} \leq (\max_{1 \leq k \leq i-1} c_k)(2^{2i})$. Thus we can bound the length of b_i by $|b_i| \leq 2i + \max_{1 \leq k \leq i-1} |c_k| \leq 2i + \max_{1 \leq k \leq i} |c_k|$. Since this is true for all $i \in \{2, 3, \ldots\}$, it follows that if $\max_{1 \leq \ell \leq j(n)} |c_\ell|$ is at most polynomial in n, then $\max_{1 \leq \ell \leq j(n)} |b_\ell|$ is at most polynomial in n, and since for all i, $a_i = b_i + c_i$, $\max_{1 \leq \ell \leq j(n)} |a_\ell|$ is at most polynomial in n. We now show that $\max_{1 \leq \ell \leq j(n)} |c_\ell|$ is in fact polynomial in n.

Let $n \in \{2, 3, \ldots\}$ be arbitrary. For each $i \in \{2, 3, \ldots, j(n)\}$, we have that $|b_i| \geq |c_1| \geq n_0$. Since F restricted to $\{t \in \mathbb{R}^+ \mid t \geq n_0\}$ is nondecreasing,

$$|c_i| \leq F(|b_i|) \leq F(2i + \max_{1 \leq k \leq i-1} |c_k|). \tag{2}$$

Since Eq. 2 holds for $2 \leq i \leq j(n)$ we can repeatedly apply it inside the max to get

$$|c_i| \leq F(2i + F(2(i-1) + F(\cdots 2 \cdot 4 + F(2 \cdot 3 + F(2 \cdot 2 + |c_1|)) \cdots))). \tag{3}$$

Recall that $\lambda = 4 + |c_1|$. From condition (a) of the theorem statement and since $|c_1| \geq n_0$, we have $F(\lambda) \geq 2 + \lambda = 2 + 4 + |c_1| \geq 6$, and thus $|c_i| \leq F(2i + F(2(i-1) + F(\cdots 2 \cdot 4 + F(2F(\lambda)) \cdots)))$. Since it follows from our theorem's assumptions that $(\forall t \geq \lambda)(\forall c \in \mathbb{N}^+)[cF(t) \geq F(ct)]$, we have $|c_i| \leq F(2i + F(2(i-1) + F(\cdots 2 \cdot 4 + 2F(F(\lambda)) \cdots)))$. Continuing to use the inequalities $(\forall k \geq 3)[2 \cdot k \leq F^{[k-2]}(\lambda)]$ and $(\forall t \geq \lambda)(\forall c \in \mathbb{N}^+)[cF(t) \geq F(ct)]$ we get $|c_i| \leq (i-1)(F^{[i-1]}(\lambda))$. Since $(\forall t \geq \lambda)[F(t) \geq t]$ and $i \leq j(n)$, we have that $|c_i| \leq (i-1)(F^{[i-1]}(\lambda)) \leq j(n)F^{[j(n)]}(\lambda)$. Since this bound holds for all $i \in \{2, 3, \ldots, j(n)\}$, it follows that $\max_{2 \leq \ell \leq n} |c_\ell| \leq j(n)F^{[j(n)]}(\lambda)$, and thus $\max_{1 \leq \ell \leq n} |c_\ell| \leq j(n)F^{[j(n)]}(\lambda) + |c_1|$. By supposition, $F^{[j(n)]}(\lambda) = \mathcal{O}(n^{\beta})$. Also, from our theorem's assumptions, $j(n)$ is polynomial in the value n, which means we can find some β'' such that $j(n) = \mathcal{O}(n^{\beta''})$. Hence we have $j(n)F^{[j(n)]}(\lambda) = \mathcal{O}(n^{\beta+\beta''})$. Since $|c_1|$ is a finite constant, this means $j(n)F^{[j(n)]}(\lambda) + |c_1|$ is polynomially bounded, and so $\max_{1 \leq \ell \leq j(n)} |c_\ell|$ is at most polynomial in n. By the argument in the preceding paragraph, $\max_{1 \leq \ell \leq j(n)} |a_\ell|$ is at most polynomial in n.

We now show that $\mathrm{UP}_{\leq j(n)} \subseteq \mathrm{RC}_T$. Let L be in $\mathrm{UP}_{\leq j(n)}$, witnessed by an NPTM $\hat{N}$. To show $L \in \mathrm{RC}_T$ we describe an NPTM N that, on each input x, has 0 accepting paths if $x \notin L$, and has $\#\mathrm{acc}_N(x) \in T$ if $x \in L$. On input x, our machine N computes $j(|x|)$ and then computes the constants $c_1, c_2, \ldots, c_{j(|x|)}$ as described above. Then N nondeterministically guesses an integer $i \in \{1, 2, \ldots, j(|x|)\}$, and nondeterministically guesses a cardinality-i set of paths of $\hat{N}(x)$. If all the paths guessed in a cardinality-i set are accepting paths, then N branches into c_i accepting paths; otherwise, that branch of N rejects. If $\hat{N}(x)$ has fewer than i paths, then the subtree of N that guessed i will have 0 accepting paths, since we cannot guess i distinct paths of $\hat{N}(x)$. We claim that N shows $L \in \mathrm{RC}_T$.

Consider any input x. If $x \notin L$, then clearly for all $i \in \{1, 2, \ldots, j(|x|)\}$ each cardinality-i set of paths of $\hat{N}$ guessed will have at least one rejecting path, and so N will have no accepting path. Suppose $x \in L$. Then $\hat{N}$ must have some number of accepting paths k. Since $\hat{N}$ witnesses $L \in \mathrm{UP}_{\leq j(n)}$, we must have $1 \leq k \leq j(|x|)$. Our machine N will have c_1 accepting paths for each accepting path of $\hat{N}$, c_2 additional accepting paths for each pair of accepting paths of $\hat{N}$, c_3 additional accepting paths for each triple of accepting paths of $\hat{N}$, and so on. Of course, for any cardinality-i set where $i > k$, at least one of the paths must be rejecting, and so N will have no accepting paths from guessing each $i > k$. Thus we have $\#\mathrm{acc}_N(x) = \sum_{1 \leq \ell \leq k} c_\ell \binom{k}{\ell}$. If $k = 1$, we have $\#\mathrm{acc}_N(x) = c_1$. If $2 \leq k \leq j(|x|)$, then $\#\mathrm{acc}_N(x) = c_k + \sum_{1 \leq \ell \leq k-1} c_\ell \binom{k}{\ell} = c_k + b_k = a_k$. In either case, $\#\mathrm{acc}_N(x) \in S$, and hence $\#\mathrm{acc}_N(x) \in T$. To complete our proof for $L \in \mathrm{RC}_T$ we need to check that N is an NPTM.

Note that, by assumption, $j(|x|)$ can be computed in time polynomial in $|x|$. Furthermore, the value $j(|x|)$ is at most polynomial in $|x|$, and so N's simulation of each cardinality-i set of paths of $\hat{N}$ can be done in time polynomial in $|x|$. Since S is P-printable and $\max_{1 \leq i \leq j(|x|)} |a_i|$ is at most polynomial in $|x|$, finding the constants a_i can be done in time polynomial in $|x|$. Also, since $\max_{1 \leq i \leq j(|x|)} |c_i|$ is at most polynomial in $|x|$, the addition and multiplication to compute each c_i can be done in time polynomial in $|x|$. All other operations done by N are also polynomial-time, and so N is an NPTM. ◀

It is worth noting that in general iterative constant-setting proofs it is sometimes useful to have a nonzero constant c_0 in order to add a constant number $c_0 \binom{i}{0} = c_0$ of accepting paths. However, when trying to show containment in a restricted counting class (as is the case here), we set $c_0 = 0$ to ensure that $\#\mathrm{acc}_N(x) = 0$ if $x \notin L$, and so we do not even have a c_0 but rather start iterative constant-setting and its sums with the c_1 case (as in Eq. 1).

Theorem 4.7 can be applied to get complexity-class containments. In particular, we now define a notion of nongappiness based on a multiplicative-constant increase in lengths, and we show – as Theorem 4.10 – that this notion of nongappiness allows us to accept all sets of logarithmic ambiguity.

▶ **Definition 4.8.** *A set $S \subseteq \mathbb{N}^+$ is $\mathcal{O}(n)$-nongappy if $S \neq \emptyset$ and $(\exists f \in \mathcal{O}(n))(\forall m \in S)(\exists m' \in S)[m' > m \wedge |m'| \leq f(|m|)]$.*

The following proposition notes that one can view this definition in a form similar to Borchert, Hemaspaandra, and Rothe's definition to see that $\mathcal{O}(n)$-nongappy sets are, as to the increase in the lengths of consecutive elements, bounded by a multiplicative constant. (In terms of values, this means that the gaps between the values of one element of the set and the next are bounded by a polynomial increase.)

▶ **Proposition 4.9.** *A set $S \subseteq \mathbb{N}^+$ is $\mathcal{O}(n)$-nongappy if and only if there exists $k \in \mathbb{N}^+$ such that S is kn-nongappy.*

▶ **Theorem 4.10.** *If $T \subseteq \mathbb{N}^+$ has an $\mathcal{O}(n)$-nongappy, P-printable subset, then $\mathrm{UP}_{\leq \mathcal{O}(\log n)} \subseteq \mathrm{RC}_T$.*

Proof. By the "only if" direction of Proposition 4.9, there exists a $k \in \mathbb{N}^+$ such that T has a kn-nongappy, P-printable subset. We can assume $k \geq 2$ since if a set has a $1n$-nongappy, P-printable subset then it also has a $2n$-nongappy, P-printable subset. Let $F : \mathbb{R}^+ \to \mathbb{R}^+$

be the function $F(t) = kt$. The function F satisfies the conditions from Theorem 4.7 since for all $t \geq 2$, $F(t) = kt \geq t + 2$, $(\forall c)[cF(n) = ckn = F(cn)]$, and F is nondecreasing on $\mathbb{R}^+$. Let $\lambda = 4 + |s|$ where s is the smallest element of the kn-nongappy, P-printable subset of T such that the conditions on F hold for all $t \geq |s|$, i.e., s is the smallest element of the kn-nongappy, P-printable subset of T such that $|s| \geq 2$. For any function $g : \mathbb{N} \to \mathbb{R}^{\geq 1}$ satisfying $g(n) = \mathcal{O}(\log n)$ it is not hard to see (since for each natural n it holds that $\log(n + 2) \geq 1$) that there must exist some $d \in \mathbb{N}^+$ such that $(\forall n \in \mathbb{N}^+)[g(n) \leq d \log(n + 2)]$, and hence $\mathrm{UP}_{\leq g(n)} \subseteq \mathrm{UP}_{\leq d \log(n+2)} = \mathrm{UP}_{\leq \lfloor d \log(n+2) \rfloor}$. Additionally, $j(n) = \lfloor d \log(n + 2) \rfloor$ satisfies the conditions from Theorem 4.7 since $j(n)$ can be computed in time polynomial in n and has value at most polynomial in n. Applying Theorem 4.7, to prove that $\mathrm{UP}_{\leq j(n)} \subseteq \mathrm{RC}_T$ it suffices to show that there is some $\beta \in \mathbb{N}^+$ such that $F^{[j(n)]}(\lambda) = \mathcal{O}(n^\beta)$ where λ is given by the statement of the theorem. So it suffices to show that for some $\beta \in \mathbb{N}^+$ and for all but finitely many n, $F^{[j(n)]}(\lambda) \leq n^\beta$. Note that $F^{[j(n)]}(\lambda) = k^{j(n)}\lambda$. So it is enough to show that for all but finitely many n, $k^{j(n)}\lambda \leq n^\beta$, or (taking logs) equivalently that for all but finitely many n, $\lfloor d \log(n + 2) \rfloor \log k + \log \lambda \leq \beta \log n$. Set β to be the least integer greater than $2d \log k + \log \lambda$. Then for all $n \geq 2$ we have $\beta \log n \geq 2d \log k \log n + \log \lambda \log n \geq d \log k \log(n^2) + \log \lambda \log n \geq d \log k \log(n + 2) + \log \lambda \geq \lfloor d \log(n + 2) \rfloor \log k + \log \lambda$, which is what we needed. Thus for any function $g : \mathbb{N} \to \mathbb{R}^{\geq 1}$ satisfying $g(n) = \mathcal{O}(\log n)$ we have that there exists a function j such that $\mathrm{UP}_{\leq g(n)} \subseteq \mathrm{UP}_{\leq j(n)} \subseteq \mathrm{RC}_T$. ◀

▶ **Corollary 4.11.** *If* PRIMES *has an $\mathcal{O}(n)$-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(\log n)} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$.

In order for the iterative constant-setting approach used in Theorem 4.7 to be applicable, it is clear that we need to consider UP classes that have at most polynomial ambiguity, because otherwise the constructed NPTMs could not guess large enough collections of paths within polynomial time. Since in the statement of Theorem 4.7 we use the function j to denote the ambiguity of a particular UP class, this requires j to be at most polynomial in the value of its input. Furthermore, since our iterative constant-setting requires having a bound on the number of accepting paths the UP machine could have had on a particular string, we also need to be able to compute the function j in time polynomial in the value of its input. Thus the limitations on the function j are natural and seem difficult to remove. Theorem 4.7 is flexible enough to, by a proof similar to that of Theorem 4.10, imply Borchert, Hemaspaandra, and Rothe's result stated in Theorem 4.1 where j reaches its polynomial bound. Another limitation of Theorem 4.7 is that it requires that for all t greater than or equal to a fixed constant n_0, $(\forall c \in \mathbb{N}^+)[cF(t) \geq F(ct)]$. It is possible to prove a similar result where for all t greater than or equal to a fixed constant n_0, $(\forall c \in \mathbb{N}^+)[cF(t) \leq F(ct)]$, which we now do as Theorem 4.12.

▶ **Theorem 4.12.** *Let F be a function mapping from $\mathbb{R}^+$ to $\mathbb{R}^+$ and let n_0 be a positive natural number such that F restricted to the domain $\{t \in \mathbb{R}^+ \mid t \geq n_0\}$ is nondecreasing and for all $t \geq n_0$ we have (a) $F(t) \geq t + 2$ and (b) $(\forall c \in \mathbb{N}^+)[cF(t) \leq F(ct)]$. Let j be a function mapping from $\mathbb{N}$ to $\mathbb{N}^+$ that is computable in time polynomial in the value of its input and whose output is at most polynomial in the value of its input. Suppose $T \subseteq \mathbb{N}^+$ has an F-nongappy, P-printable subset S. Let $\lambda = 4 + |s|$ where s is the smallest element of S with $|s| \geq n_0$. If for some β, $F^{[j(n)]}(j(n)\lambda) = \mathcal{O}(n^\beta)$, then $\mathrm{UP}_{\leq j(n)} \subseteq \mathrm{RC}_T$.*

How does this theorem compare with our other metatheorem, Theorem 4.7? Since in both metatheorems F is nondecreasing after a prefix, speaking informally and broadly, the functions F where (after a prefix) $(\forall c \in \mathbb{N}^+)[cF(t) \leq F(ct)]$ holds grow faster than the functions F where (after a prefix) $(\forall c \in \mathbb{N}^+)[cF(t) \geq F(ct)]$ holds. (The examples we give of applying the two theorems reflect this.) So, this second metatheorem is accommodating larger gaps in the sets of integers that define our restricted counting class, but is also assuming a slightly stronger condition for the containment of an ambiguity-limited class to follow. More specifically, since we have the extra factor of $j(n)$ inside of the iterated application of F, we may need even more than $j(|x|)$ elements to be reachable in polynomial time (exactly how many more will depend on the particular function F).

We now discuss some other notions of nongappiness and obtain complexity-class containments regarding them using Theorem 4.12.

▶ **Theorem 4.13.** *If there exists a real number $k > 1$ such that $T \subseteq \mathbb{N}^+$ has an n^k-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(1)+\frac{\log\log n}{2\log k}} \subseteq \mathrm{RC}_T$.

Theorem 4.13 has an interesting consequence when applied to the Mersenne primes. In particular, as we now show, it can be used to prove that the Lenstra–Pomerance–Wagstaff Conjecture implies that the $\mathcal{O}(\log\log n)$-ambiguity sets in NP each belong to $\mathrm{RC}_{\mathrm{PRIMES}}$.

A Mersenne prime is a prime of the form $2^k - 1$. We will use the Mersenne prime counting function $\mu(n)$ to denote the number of Mersenne primes with length less than or equal to n (when represented in binary). The Lenstra–Pomerance–Wagstaff Conjecture [35, 38] (see also [14]) asserts that there are infinitely many Mersenne primes, and that $\mu(n)$ grows asymptotically as $e^\gamma \log n$ where $\gamma \approx 0.577$ is the Euler–Mascheroni constant. (Note: We say that $f(n)$ grows asymptotically as $g(n)$ when $\lim_{n\to\infty} f(n)/g(n) = 1$.) Having infinitely many Mersenne primes immediately yields an infinite, P-printable subset of the primes. In particular, on input 1^n we can print all Mersenne primes of length less than or equal to n in polynomial time by just checking (using a deterministic polynomial-time primality test [1]) each number of the form $2^k - 1$ whose length is less than or equal to n, and if it is prime then printing it. If the Lenstra–Pomerance–Wagstaff Conjecture holds, what can we also say about the gaps in the Mersenne primes? We address that with the following result.

▶ **Theorem 4.14.** *If the Lenstra–Pomerance–Wagstaff Conjecture holds, then for each $\epsilon > 0$ the primes (indeed, even the Mersenne primes) have an $n^{1+\epsilon}$-nongappy, P-printable subset.*

▶ **Corollary 4.15.** *If the Lenstra–Pomerance–Wagstaff Conjecture holds, then* $\mathrm{UP}_{\leq \mathcal{O}(\log\log n)} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$ *(indeed,* $\mathrm{UP}_{\leq \mathcal{O}(\log\log n)} \subseteq \mathrm{RC}_{\mathrm{MersennePRIMES}}$*).*

We will soon turn to discussing more notions of nongappiness and what containment theorems hold regarding them. However, to support one of those notions, we first define a function that will arise naturally in Theorem 4.19.

▶ **Definition 4.16.** *For any $\alpha \in \mathbb{R}$, $\alpha > 0$, $\log^{\circledast}(\alpha)$ is the largest natural number k such that $\log^{[k]}(\alpha) \geq k$. We define $\log^{\circledast}(0)$ to be 0.*

For $\alpha > 1$, taking $k = 0$ satisfies $\log^{[k]}(\alpha) \geq k$. Also, for all $\ell \geq \log^*(\alpha)$, $\log^{[\ell]}(\alpha) \leq \log^{[\log^*(\alpha)]}(\alpha) \leq 1 \leq \ell$, and so no $\ell \geq \log^*(\alpha)$ can be used as the k in the definition above. So there is at least one, but only finitely many k such that $\log^{[k]}(\alpha) \geq k$, which means that $\log^{\circledast}(\alpha)$ is well-defined. Using the def. of $\log^{\circledast}(\alpha)$ and the above, we get $\log^{\circledast}(\alpha) \leq \log^*(\alpha)$ when $\alpha > 1$. For $\alpha \leq 1$, 0 is the only natural number for which the condition from the def. holds, and so $\log^{\circledast}(\alpha) = 0$ if $\alpha \leq 1$. Thus for $\alpha \leq 1$, $\log^{\circledast}(\alpha) = \log^*(\alpha)$. As to the relationship of its values to those of $\log^*$, we have the following proposition.

▶ **Proposition 4.17.** *For all $\alpha \geq 0$, $\log^*(\alpha) - \log^*(\log^*(\alpha) + 1) - 1 \leq \log^{\circledast}(\alpha) \leq \log^*(\alpha)$.*

▶ **Definition 4.18.** *A nonempty set* $S \subseteq \mathbb{N}^+$ *is*

1. $\mathcal{O}(n \log n)$*-nongappy if* $(\exists f \in \mathcal{O}(n \log n))(\forall m \in S)(\exists m' \in S)[m' > m \wedge |m'| \leq f(|m|)]$, *and*
2. $n^{(\log n)^{\mathcal{O}(1)}}$*-nongappy if* $(\exists f \in \mathcal{O}(1))(\forall m \in S)(\exists m' \in S)[m' > m \wedge |m'| \leq |m|^{(\log |m|)^{f(|m|)}}]$.

Definitions of $n^{\log n}$-nongappy and 2^n-nongappy are provided via Definition 4.6, since $n^{\log n}$ and 2^n are each a single function, not a collection of functions. Those two notions, along with the two notions of Definition 4.18, will be the focus of Theorem 4.19. That theorem obtains the containments related to those four notions of nongappiness. As one would expect, as the allowed gaps become larger the corresponding UP classes become more restrictive in their ambiguity bounds. Theorem 4.19 also gives a corollary about primes.

▶ **Theorem 4.19.** *Let* T *be a subset of* $\mathbb{N}^+$.

1. *If* T *has an* $\mathcal{O}(n \log n)$*-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(\sqrt{\log n})} \subseteq \mathrm{RC}_T$.
2. *If* T *has an* $n^{\log n}$*-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{1}{2} \log \log \log n} \subseteq \mathrm{RC}_T$.
3. *If* T *has an* $n^{(\log n)^{\mathcal{O}(1)}}$*-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{1}{3} \log \log \log \log n} \subseteq \mathrm{RC}_T$.
4. *If* T *has a* 2^n*-nongappy, P-printable subset* S*, then* $\mathrm{UP}_{\leq \max(1, \lfloor \frac{\log^{\circledast} n}{\lambda} \rfloor)} \subseteq \mathrm{RC}_T$ *(and so certainly also* $\mathrm{UP}_{\leq \max(1, \lfloor \frac{\log^*(n) - \log^*(\log^*(n)+1)-1}{\lambda} \rfloor)} \subseteq \mathrm{RC}_T$*), where* $\lambda = 4 + \min_{s \in S, |s| \geq 2}(|s|)$.

▶ **Corollary 4.20.**

1. *If* PRIMES *has an* $\mathcal{O}(n \log n)$*-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(\sqrt{\log n})} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$.
2. *If* PRIMES *has an* $n^{\log n}$*-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{1}{2} \log \log \log n} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$.
3. *If* PRIMES *has an* $n^{(\log n)^{\mathcal{O}(1)}}$*-nongappy, P-printable subset, then* $\mathrm{UP}_{\leq \mathcal{O}(1) + \frac{1}{3} \log \log \log \log n} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$.
4. *If* PRIMES *has a* 2^n*-nongappy, P-printable subset* S*, then* $\mathrm{UP}_{\leq \max(1, \lfloor \frac{\log^{\circledast} n}{\lambda} \rfloor)} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$ *(and so certainly also* $\mathrm{UP}_{\leq \max(1, \lfloor \frac{\log^*(n) - \log^*(\log^*(n)+1)-1}{\lambda} \rfloor)} \subseteq \mathrm{RC}_{\mathrm{PRIMES}}$*), where* $\lambda = 4 + \min_{s \in S, |s| \geq 2}(|s|)$.

5 Conclusions and Open Problems

We proved two flexible metatheorems that can be used to obtain containments of ambiguity-limited classes in restricted counting classes, and applied those theorems to prove containments for some of the most natural ambiguity-limited classes. Beyond the containments we derived based on Theorems 4.7 and 4.12, those two metatheorems themselves seem to reflect a trade-off between the ambiguity allowed in an ambiguity-limited class and the smallness of gaps in a set of natural numbers defining a restricted counting class. One open problem is to make explicit, in a smooth and complete fashion, this trade-off between gaps and ambiguity. Another challenge is to capture the relationship between $\log^{\circledast}$ and $\log^*$ more tightly than Proposition 4.17 does (see Section 4 of [26]). Finally, though it would be a major advance since not even any infinite, P-printable subsets of the primes are currently known, in light of Corollaries 4.11 and 4.20, a natural goal would be to prove that the primes have infinite, P-printable subsets that satisfy some, or all, of our nongappiness properties.

References

1 M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

2 E. Allender. Some consequences of the existence of pseudorandom generators. *Journal of Computer and System Sciences*, 39(1):101–124, 1989.

3 E. Allender and R. Rubinstein. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.

4 R. Beigel. On the relativized power of additional accepting paths. In *Proceedings of the 4th Structure in Complexity Theory Conference*, pages 216–224. IEEE Computer Society Press, June 1989.

5 R. Beigel, J. Gill, and U. Hertrampf. Counting classes: Thresholds, parity, mods, and fewness. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, pages 49–57. Springer-Verlag Lecture Notes in Computer Science #415, February 1990.

6 L. Berman. *Polynomial Reducibilities and Complete Sets.* PhD thesis, Cornell University, Ithaca, NY, 1977.

7 B. Borchert, L. Hemaspaandra, and J. Rothe. Restrictive acceptance suffices for equivalence problems. *London Mathematical Society Journal of Computation and Mathematics*, 3:86–95, 2000.

8 B. Borchert and F. Stephan. Looking for an analogue of Rice's Theorem in circuit complexity theory. *Mathematical Logic Quarterly*, 46(4):489–504, 2000.

9 D. Bovet, P. Crescenzi, and R. Silvestri. Complexity classes and sparse oracles. *Journal of Computer and System Sciences*, 50(3):382–390, 1995.

10 G. Brassard. Crusade for a better notation. *SIGACT News*, 17(1):60–64, 1985.

11 G. Brassard and P. Bratley. *Algorithmics: Theory & Practice.* Prentice Hall, 1988.

12 J.-Y. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

13 J.-Y. Cai and L. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23(2):95–106, 1990.

14 C. Caldwell. Heuristics model for the distribution of Mersennes. The PrimePages, 2021. URL verified 2022/6/22. URL: `https://primes.utm.edu/mersenne/heuristic.html`.

15 P. Chebyshev. Mémoire sur les nombres premiers. *Journal de Mathématiques Pures et Appliquées. Série 1*, 17:366–390, 1852.

16 J. Cox and T. Pay. An overview of some semantic and syntactic complexity classes. Technical report, Computing Research Repository, June 2018. `arXiv:1806.03501`.

17 S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

18 S. Fenner, L. Fortnow, and L. Li. Gap-definability as a closure property. *Information and Computation*, 130(1):1–17, 1996.

19 K. Ford, B. Green, S. Konyagin, J. Maynard, and T. Tao. Long gaps between primes. *Journal of the American Mathematical Society*, 31(1):65–105, 2018.

20 K. Ford, B. Green, S. Konyagin, and T. Tao. Large gaps between consecutive prime numbers. *Annals of Mathematics. Second Series*, 183(3):935–974, 2016.

21 J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

22 D. Gillies. Three new Mersenne primes and a statistical theory. *Mathematics of Computation*, 18(85):93–97, 1964. Corrigendum 31(140):1051, 1977.

23 L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43(1):43–58, 1986.

24 J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988. `doi:10.1137/0217018`.

25 J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34(1–2):17–32, 1984.

26 L. Hemaspaandra, M. Juvekar, A. Nadjimzadah, and P. Phillips. Gaps, ambiguity, and establishing complexity-class containments via iterative constant-setting. Technical report, Computing Research Repository, September 2021. Revised, June 2022. `arXiv:2109.147648`.

27 L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Springer-Verlag, 2002.

28 L. Hemaspaandra and J. Rothe. A second step towards complexity-theoretic analogs of Rice's Theorem. *Theoretical Computer Science*, 244(1–2):205–217, 2000.

29 L. Hemaspaandra and M. Zimand. Strong forms of balanced immunity. Technical Report TR-480, Department of Computer Science, University of Rochester, Rochester, NY, December 1993. Revised, May 1994.

30 K. Ko. On some natural complete operators. *Theoretical Computer Science*, 37(1):1–30, 1985.

31 J. Köbler, U. Schöning, S. Toda, and J. Torán. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2):272–286, 1992.

32 K.-J. Lange and P. Rossmanith. Unambiguous polynomial hierarchies and exponential size. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 106–115. IEEE Computer Society Press, June/July 1994.

33 J. Maynard. Large gaps between primes. *Annals of Mathematics, Second Series*, 183(3):915–933, 2016.

34 C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag Lecture Notes in Computer Science #145, January 1983.

35 C. Pomerance. Recent developments in primality testing. *The Mathematical Intelligencer*, 3(3):97–105, 1981.

36 L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.

37 L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

38 S. Wagstaff, Jr. Divisors of Mersenne numbers. *Mathematics of Computation*, 40(161):385–397, 1983.

39 O. Watanabe. On hardness of one-way functions. *Information Processing Letters*, 27(3):151–157, 1988.

40 Wikipedia. Prime gap. en.wikipedia.org/wiki/Prime_gap, 2021. URL verified 2022/6/22.

41 Wikipedia. Gillies' conjecture. en.wikipedia.org/wiki/Gillies%27_conjecture, 2022. URL verified 2022/6/22.

Independent Set Reconfiguration on Directed Graphs

Takehiro Ito
Graduate School of Information Sciences,
Tohoku University, Sendai, Japan

Yuni Iwamasa
Graduate School of Informatics,
Kyoto University, Kyoto, Japan

Yasuaki Kobayashi
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

Yu Nakahata
Division of Information Science,
Nara Institute of Science and Technology, Ikoma, Japan

Yota Otachi
Graduate School of Informatics,
Nagoya University, Nagoya, Japan

Masahiro Takahashi
Graduate School of Informatics,
Kyoto University, Kyoto, Japan

Kunihiro Wasa
Faculty of Science and Engineering,
Hosei University, Tokyo, Japan

Abstract

Directed Token Sliding asks, given a directed graph and two sets of pairwise nonadjacent vertices, whether one can reach from one set to the other by repeatedly applying a local operation that exchanges a vertex in the current set with one of its out-neighbors, while keeping the nonadjacency. It can be seen as a reconfiguration process where a token is placed on each vertex in the current set, and the local operation slides a token along an arc respecting its direction. Previously, such a problem was extensively studied on undirected graphs, where the edges have no directions and thus the local operation is symmetric. Directed Token Sliding is a generalization of its undirected variant since an undirected edge can be simulated by two arcs of opposite directions.

In this paper, we initiate the algorithmic study of Directed Token Sliding. We first observe that the problem is PSPACE-complete even if we forbid parallel arcs in opposite directions and that the problem on directed acyclic graphs is NP-complete and W[1]-hard parameterized by the size of the sets in consideration. We then show our main result: a linear-time algorithm for the problem on directed graphs whose underlying undirected graphs are trees, which are called polytrees. Such a result is also known for the undirected variant of the problem on trees [Demaine et al. TCS 2015], but the techniques used here are quite different because of the asymmetric nature of the directed problem. We present a characterization of yes-instances based on the existence of a certain set of directed paths, and then derive simple equivalent conditions from it by some observations, which yield an efficient algorithm. For the polytree case, we also present a quadratic-time algorithm that outputs, if the input is a yes-instance, one of the shortest reconfiguration sequences.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorial reconfiguration, token sliding, directed graph, independent set, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.58

Related Version *Full Version*: `https://arxiv.org/abs/2203.13435`

Funding *Takehiro Ito*: JSPS KAKENHI Grant Numbers JP18H04091, JP19K11814, JP20H05793
Yuni Iwamasa: JSPS KAKENHI Grant Numbers JP20K23323, JP20H05795, JP22K17854
Yasuaki Kobayashi: JSPS KAKENHI Grant Numbers JP20H00595, JP20H05793, JP20K19742
Yu Nakahata: JSPS KAKENHI Grant Numbers JP19H01103, JP20H05794, JP22H03549, JP22K17851
Yota Otachi: JSPS KAKENHI Grant Numbers JP18H04091, JP20H05793, JP21K11752, JP22H00513
Kunihiro Wasa: JSPS KAKENHI Grant Numbers JP19K20350, JP20H05793, JP22H03549, JP22K17849

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 58; pp. 58:1–58:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In a reconfiguration problem, we are given an instance of a decision problem with two feasible solutions I^{s} and I^{t}. The task is to decide if we can transform I^{s} into I^{t} by repeatedly applying a modification rule while maintaining the feasibility of intermediate solutions. Reconfiguration problems are studied for several problems such as Independent Set [17, 28, 8, 12, 13, 18, 4, 10, 29, 3, 26], Dominating Set [34, 16, 30], Clique [27], Matching [9, 5, 23], and Graph Coloring [7, 32, 6]. (See also surveys [35, 31].)

Among the problems, Independent Set Reconfiguration is one of the most well-studied problems. There are three different modification rules studied in the literature: Token Addition and Removal (TAR) [19, 28], Token Jumping (TJ) [24, 25, 10], and Token Sliding (TS) [17, 8, 12, 13, 18, 4, 29, 3, 26]. TAR allows us to add or remove any vertex in the current independent set as long as the size of the resultant set is at least a given threshold. TJ allows us to exchange any vertex in the set with any vertex outside the set. TS is a restricted version of TJ that requires the exchanged vertices to be adjacent. We call Independent Set Reconfiguration under TS simply Token Sliding. Observe that all these rules are *symmetric*: If one can transform I^{s} into I^{t}, then one also can transform I^{t} into I^{s}. Our question is: What happens if we adopt an *asymmetric* rule?

In this paper, we study a directed variant of Token Sliding, which we call Directed Token Sliding. In this problem, the input graph is directed and we can slide tokens along the arcs only in their directions. Here, we say that a set I of vertices of a directed graph G is an independent set when it is an independent set in the underlying undirected graph of G.[1] If we allow two arcs with the opposite directions, this problem is a generalization of Token Sliding, and thus PSPACE-complete in general.

We show three results for Directed Token Sliding. First, we show that the problem is PSPACE-complete even on oriented graphs, where antiparallel arcs (two parallel arcs with opposite directions) are not allowed. Second, if we restrict input graphs to directed acyclic graphs (DAGs), we prove that the problem is NP-complete. Moreover, the problem is $W[1]$-hard parameterized by the number of tokens. As for our positive and main result, we show that the problem can be solved in linear time on polytrees, that is, digraphs whose underlying graphs are trees. For a polytree, we show that all reconfiguration sequences have the same length. We can also construct one of them in $\mathrm{O}(k|V|)$ time, where k is the size of the input independent sets and V is the set of vertices of the input graph. Note that our algorithm is optimal in the following sense: We can show that there exists an infinite family of instances on directed paths whose reconfiguration sequences have length $\Omega(k|V|)$ (which is easily obtained from lower bound examples for undirected paths given in [12]). For Token Sliding on undirected trees, Demaine et al. [12] showed a linear-time algorithm to check the reconfigurability. They also showed an $\mathrm{O}(|V|^2)$-time algorithm to construct a reconfiguration sequence of length $\mathrm{O}(|V|^2)$ for yes-instances. However, the output sequence is not guaranteed to be shortest. The best known algorithm to output a shortest reconfiguration sequence for undirected trees runs in $\mathrm{O}(|V|^5)$ time [33]. In contrast to the undirected counterpart, our algorithm to construct a (shortest) reconfiguration sequence on polytrees runs in $\mathrm{O}(k|V|)$ time, which is optimal. It should be mentioned that our quadratic-time algorithm does not imply a quadratic-time algorithm for finding a shortest reconfiguration sequence for undirected trees since we do not allow polytrees to have antiparallel arcs.

[1] One may define an independent set of a digraph in an asymmetric way: A subset I of the vertex set V of the directed graph G is an independent set if $u \in I$ implies $v \notin I$ for all $u, v \in V$ such that G has an arc (u, v). However, this definition is the same as ours. (If u and v are adjacent in the underlying undirected graph, at most one of u and v can belong to I.)

Due to the space limitation, several proofs (marked with $\star$) are omitted and can be found in the full version [21].

Related work

Token Sliding on undirected graphs was introduced by Hearn and Demaine [17]. They show that the problem is PSPACE-complete even on planar graphs with maximum degree 3. The problem is also PSPACE-complete on bipartite graphs [29] and split graphs [3]. In contrast to these hardness results, the problem is polynomial-time solvable on cographs [28], claw-free graphs [8], trees [12], cactus graphs [18], interval graphs [4], bipartite permutation graphs, and bipartite distance-hereditary graphs [13].

Interestingly, Independent Set Reconfiguration on bipartite graphs is NP-complete under TAR and TJ [29]. Such graph classes are not known for TS. In this paper, we show that Directed Token Sliding is NP-complete on directed acyclic graphs (DAGs).

Token Sliding is also studied from the viewpoint of fixed-parameter tractability (FPT). When the parameter is the number of tokens, the problem is W[1]-hard both on C_4-free graphs and bipartite graphs, while it becomes FPT on their intersection, C_4-free bipartite graphs [2]. In this paper, we show that Directed Token Sliding is W[1]-hard on DAGs.

Although Token Sliding, as far as we know, has not been studied for digraphs, there are some studies on reconfiguration considering directions of edges. An orientation of a simple undirected graph is an assignment of directions to the edges of the graph. There are several studies for reconfiguring orientations of a graph, such as strong orientations [15, 14, 20], acyclic orientations [14], nondeterministic constraint logic [17], and α-orientations [1]. Very recently, Ito et al. [22] studied reconfiguration of several subgraphs in directed graphs, such as directed trees, directed paths, and directed acyclic subgraphs. Although these reconfiguration problems are defined on directed graphs, the reconfiguration rules are symmetric, meaning that if one solution X can be obtained from another solution Y by applying some reconfiguration rule, Y can be obtained from X by the same rule as well.

2 Preliminaries

In this section, we introduce notations on graphs and define our problem. For a positive integer k, we define $[k] \coloneqq \{1, \dots, k\}$. We also define $[0] \coloneqq \emptyset$.

2.1 Graph notation

Let G be a directed graph (digraph). We denote by $V(G)$ and $A(G)$ the vertex and arc sets of G, respectively. For a digraph G, its *underlying (undirected) graph*, denoted by G^{und}, is an undirected graph obtained by ignoring the directions of arcs in G. For an undirected graph G', we denote by $V(G')$ and $E(G')$ the vertex and edge sets of G', respectively. For a digraph G, a vertex set $S \subseteq V(G)$ is said to be *independent* if it is an independent set in the underlying graph G^{und}, i.e., for all two different vertices $u, v \in S$, we have $\{u, v\} \notin E(G^{\mathrm{und}})$. A digraph G is an *oriented graph* if, for all vertices $u, v \in V(G)$, G contains at most one of the possible arcs (u, v) or (v, u). If G contains no directed cycles, G is *acyclic* and such digraphs are called *directed acyclic graphs (DAGs)*. If G^{und} is a tree, G is a *polytree*. Similarly, if G^{und} is a forest, G is a *polyforest*.

For an arc $e = (u, v) \in A(G)$ of a digraph G, the vertex u is called the *tail* of e, and v called the *head* of e. The both u and v are called the *endpoints* of e. For $e \in A(G)$ and $v \in V(G)$, e is *incident to* v if v is an endpoint of e. If $(u, v) \in A(G)$, u is an *in-neighbor* *of v in G* and v is an *out-neighbor of u in G*. In addition, u is a *neighbor* of v and v is a *neighbor* of u. For a vertex v, we denote by $N^+_G(v)$ and $N^-_G(v)$ the sets of out-neighbors and in-neighbors of v in G, respectively, that is, $N^+_G(v) \coloneqq \{u \in V(G) \mid (v,u) \in A(G)\}$ and $N^-_G(v) \coloneqq \{u \in V(G) \mid (u,v) \in A(G)\}$. We denote by $N_G(v)$ the set of neighbors of v, that is, $N_G(v) \coloneqq N^+_G(v) \cup N^-_G(v)$. In addition, we define $N_G[v] \coloneqq N_G(v) \cup \{v\}$. We also define $\Gamma^+_G(v) \coloneqq \big\{(v,u) \bigm| u \in N^+_G(v)\big\}$, $\Gamma^-_G(v) \coloneqq \big\{(u,v) \bigm| u \in N^-_G(v)\big\}$, and $\Gamma_G(v) \coloneqq \Gamma^+_G(v) \cup \Gamma^-_G(v)$. If no confusion arises, we omit the subscript G from $N_G(v)$, $N^+_G(v)$, $N^-_G(v)$, $N_G[v]$, $\Gamma^+_G(v)$, $\Gamma^-_G(v)$, and $\Gamma_G(v)$. A *directed path* P is a sequence of vertices and arcs $(v_1, e_1, \dots, e_{\ell-1}, v_\ell)$ such that, all the vertices are distinct and, for every $i \in [\ell-1]$, $e_i = (v_i, v_{i+1})$ holds. We call v_1 and v_ℓ the *source* and *sink* of P, and denote by $s(P)$ and $t(P)$, respectively. This P is called a *v_1-v_ℓ (directed) path* or a *(directed) path from v_1 to v_ℓ*. The *length* of P is the number of arcs and we denote it by $|P|$, that is, $|P| = \ell - 1$. For a directed path P with $|P| \geq 2$, we define $s'(P) \coloneqq v_2$ and $t'(P) \coloneqq v_{\ell-1}$. For vertex sets X and Y with the same size k on a digraph, we refer to a set $\mathcal{P} = \{P_1, \dots, P_k\}$ of directed paths as a *directed path matching from X to Y* if $\mathcal{P}$ have distinct sources and distinct sinks, that is, $\{s(P_i) \mid i \in [k]\} = X$ and $\{t(P_i) \mid i \in [k]\} = Y$. Note that two sets X and Y may intersect in this definition.

2.2 Definition of Directed Token Sliding

Let I^{s} and I^{t} be independent sets in a digraph G with $|I^{\mathsf{s}}| = |I^{\mathsf{t}}|$. A sequence $\langle I_0, \dots, I_\ell \rangle$ of independent sets of G is a *reconfiguration sequence from I^{s} to I^{t} in G* if $I_0 = I^{\mathsf{s}}, I_\ell = I^{\mathsf{t}}$, and for all $i \in [\ell]$ we have $I_{i-1} \setminus I_i = \{u\}$ and $I_i \setminus I_{i-1} = \{v\}$ with $(u,v) \in A$. The *length* of $\langle I_0, \dots, I_\ell \rangle$ is ℓ. We say that *I^{s} is reconfigurable into I^{t} in G* and *I^{t} is reconfigurable from I^{s} in G* if there is a reconfiguration sequence from I^{s} to I^{t}.

We study the following problem:

Problem Directed Token Sliding
Instance A triple $(G, I^{\mathsf{s}}, I^{\mathsf{t}})$, where G is a digraph and $I^{\mathsf{s}}, I^{\mathsf{t}} \subseteq V(G)$ are independent sets in G with $|I^{\mathsf{s}}| = |I^{\mathsf{t}}|$.
Question Is I^{s} reconfigurable into I^{t}?

In Directed Token Sliding, the sets I^{s} and I^{t} can be seen as the initial and target positions of "tokens" placed on the vertices in the sets, and then the problem asks whether we can move the tokens from I^{s} to I^{t} by repeatedly sliding tokens along arcs keeping that the tokens are not adjacent. The difference between our problem and Token Sliding is that, in the former, the input graph is directed and we can slide tokens along the arcs only in their directions. However, since we can simulate an undirected edge with the two arcs with the opposite directions, the problem is a generalization of Token Sliding. It immediately follows from the PSPACE-completeness of Token Sliding [17] that Directed Token Sliding is PSPACE-complete in general.

3 Hardness results

In this section, we provide hardness results for Directed Token Sliding. The first hardness result is the PSPACE-completeness of Directed Token Sliding for oriented graphs. This follows from a reduction from Token Sliding on undirected graphs, which is PSPACE-complete [17].

The idea of the proof of Theorem 1 as follows. From an undirected graph G, we construct a directed graph G' by replacing each edge of G with two directed arcs in opposite directions. Then, $(G, I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance of Token Sliding if and only if $(G', I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance of Directed Token Sliding. To make G' being an oriented graph (that is, G' has at most one arc of (u, v) and (v, u) for $u, v \in V(G)$), we replace each vertex v of G with two copies of vertices v_1 and v_2 and each edge $\{u, v\}$ of G with a strongly connected tournament on $\{u_1, u_2, v_1, v_2\}$, instead of the above replacement. This yields an oriented graph G' and we can show that $(G, I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance of Token Sliding if and only if $(G', J^{\mathsf{s}}, J^{\mathsf{t}})$ is a yes-instance of Directed Token Sliding, where $J^{\mathsf{s}} = \{v_1 \mid v \in I^{\mathsf{s}}\}$ and $J^{\mathsf{t}} = \{v_1 \mid v \in I^{\mathsf{t}}\}$.

▶ **Theorem 1** (⋆). *Directed Token Sliding is PSPACE-complete on oriented graphs.*

The second hardness result is the NP-completeness and W[1]-hardness of Directed Token Sliding for DAGs. These results are obtained by reducing Multicolored Independent Set, which is known to be NP-complete and W[1]-hard parameterized by the solution size k [11], to Directed Token Sliding.

▶ **Theorem 2** (⋆). *Directed Token Sliding on DAGs is NP-complete and W[1]-hard parameterized by the number of tokens.*

4 Linear-time algorithm for polytrees

This section is devoted to proving our main result Theorem 3 below, a linear-time algorithm for Directed Token Sliding on polytrees.

▶ **Theorem 3.** *Let $T = (V, A)$ be a polytree. Directed Token Sliding on T can be solved in $\mathrm{O}(|V|)$ time. Moreover, if the answer is affirmative, then all reconfiguration sequences have the same length, and one of them can be constructed in $\mathrm{O}(k|V|)$ time, where k is the size of the input independent set.*

To establish Theorem 3, we give a simple characterization for yes-instances of Directed Token Sliding on polytrees. This characterization is described by the concept of directed path matchings, which will be given in the next subsection, and is a vital role in proving Theorem 3. We would like to mention that our characterization is rather simple but its proof is highly non-trivial. Given this characterization, we devise a linear time algorithm for Directed Token Sliding on polytrees and a quadratic time algorithm for constructing an actual reconfiguration sequence if the answer is affirmative, which will be given in Section 4.4.

4.1 Directed path matching

Let I^{s} and I^{t} be independent sets in T with $|I^{\mathsf{s}}| = |I^{\mathsf{t}}|$ such that I^{s} is reconfigurable into I^{t}. In a reconfiguration sequence from I^{s} to I^{t}, the i-th token moves along some directed path P_i. Clearly $\mathcal{P} := \{P_1, \dots, P_k\}$ forms a directed path matching from I^{s} to I^{t}. Thus, the existence of a directed path matching is a trivial necessary condition for yes-instances. However, the converse is not true in general. Let us consider a digraph such that its underlying graph is the star with four leaves and its center has two in-neighbors and two out-neighbors. We set I^{s} (resp. I^{t}) to be the set of in-neighbors (resp. out-neighbors) of the center. Then this graph has a directed path matching from I^{s} to I^{t}, while it is a no-instance. Nevertheless, directed path matchings still give us some insight on Directed Token Sliding for polytrees, which is vital for our linear-time algorithm. To this end, in this subsection, we give a characterization of the existence of a directed path matching between given two sets of vertices in a polytree. This characterization is described in terms of an invariant associated with each arc e.

Let X and Y be (not necessarily disjoint) sets of vertices of a polytree $T = (V, A)$ with $|X| = |Y|$, and π a bijection from X to Y. Since T is a polytree, for each $x \in X$ an (undirected) x-$\pi(x)$ path P_x is uniquely determined in T^{und}. For each $e \in A$, we define $w(e; X, Y, \pi) \coloneqq |\{x \in X \mid \overrightarrow{P_x} \text{ has } e\}| - |\{x \in X \mid \overrightarrow{P_x} \text{ has the reverse of } e\}|$, where $\overrightarrow{P_x}$ is a directed path obtained from P_x by orienting arcs from x to $\pi(x)$. The following lemma states that $w(e; X, Y, \pi)$ does not depend on a particular π. Let T' be the polyforest obtained from T by removing an arc e. Let C_e^+ (resp. C_e^-) denote the vertices of the (weakly) connected component in T' containing the head of e (resp. the tail of e).

▶ **Lemma 4** (⋆). *Let X and Y be (not necessarily disjoint) sets of vertices of T with $|X| = |Y|$, and $e \in A$ an arc of T. For each bijection $\pi : X \to Y$, we have $w(e; X, Y, \pi) = |C_e^- \cap X| - |C_e^- \cap Y|$.*

Based on this fact, we define a function w on A with respect to two vertex sets X and Y:

$$w(e; X, Y) \coloneqq |C_e^- \cap X| - |C_e^- \cap Y| \qquad (e \in A).$$

Then, we show the necessary and sufficient conditions mentioned above.

▶ **Lemma 5** (⋆). *There exists a directed path matching from X to Y if and only if $w(e; X, Y) \geq 0$ for every arc $e \in A$.*

By Lemmas 4 and 5, we can observe the following corollaries.

▶ **Corollary 6.** *For a yes-instance, the number of tokens passing through an arc e is equal to $w(e; I^{\mathrm{s}}, I^{\mathrm{t}})$ in every reconfiguration sequence. In particular, all reconfiguration sequences have the same length $\sum_{e \in A} w(e; I^{\mathrm{s}}, I^{\mathrm{t}})$.*

▶ **Corollary 7.** *If there exists an arc $e \in A$ such that $w(e; I^{\mathrm{s}}, I^{\mathrm{t}}) < 0$, then the instance is not reconfigurable.*

In the following argument, we assume $w(e; I^{\mathrm{s}}, I^{\mathrm{t}}) \geq 0$ for all $e \in A$. By depth-first search on T, we can compute the function $w(e; I^{\mathrm{s}}, I^{\mathrm{t}})$ for all e in linear time.

4.2 Tokens that move at most once

In a reconfiguration sequence, there may be a token that moves at most once. In other words, a token may not move at all or may move from the initial position to one of its out-neighbors and stay there. Such a token causes an exception in our further discussion, and thus we want to remove such tokens in advance. In this subsection, we show that such tokens are determined regardless of reconfiguration sequences, and that we can remove such tokens from the input without changing the reconfigurability.

First, we consider tokens that never move. The following lemma states that such tokens are uniquely determined from the instance $(T, I^{\mathrm{s}}, I^{\mathrm{t}})$ (not depending on an actual reconfiguration sequence). From now on, we simply write $w(e)$ to denote $w(e; I^{\mathrm{s}}, I^{\mathrm{t}})$.

▶ **Lemma 8** (⋆). *Let $(T, I^{\mathrm{s}}, I^{\mathrm{t}})$ be a yes-instance. For every reconfiguration sequence, the set of vertices containing tokens that do not move in the sequence is equal to*

$$R \coloneqq \{v \in I^{\mathrm{s}} \cap I^{\mathrm{t}} \mid w(e) = 0 \text{ for all } e \in \Gamma(v)\}.$$

A token on a vertex $v \in R$ is said to be *rigid*. By Lemma 8, all rigid tokens do not move in any reconfiguration sequence.

For some $v \in R$, suppose that there exists $u \in N(v)$ such that T has an arc $e \in \Gamma(u)$ with $w(e) > 0$. If $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance, in any reconfiguration sequence, some token passes through e, implying that this token is placed on u at some point. However, since the token on v is rigid, these tokens must be adjacent. Thus, we obtain the following corollary.

▶ **Corollary 9.** *For $v \in R$, if there exists $u \in N(v)$ such that T has an arc $e \in \Gamma(u)$ with $w(e) > 0$, then the instance $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ is a no-instance.*

Next, we consider tokens that move exactly once. We can show that the set of arcs used by such tokens is uniquely determined from the instance $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ (not depending on an actual reconfiguration sequence) by a similar argument as in Lemma 8.

▶ **Lemma 10** ($\star$)**.** *Let $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ be a yes-instance. For every reconfiguration sequence, the set of arcs used by tokens that move exactly once in the sequence is equal to*

$$B \coloneqq \{e = (u, v) \in A \mid w(e) = 1 \text{ and } w(e') = 0 \text{ for every } e' \in (\Gamma(u) \cup \Gamma(v)) \setminus \{e\}\}.$$

Take any $(u, v) \in B$. By Lemma 10, for every reconfiguration sequence the token on u slides to an out-neighbor v of u and stays there, which also implies $u \in I^{\mathsf{s}}$ and $v \in I^{\mathsf{t}}$. Since the tokens must form an independent set, the other tokens can be placed on neither u nor v. Given this, we refer to an arc $e \in B$ as a *blocking arc* in $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$.

We can compute the rigid tokens and blocking arcs from $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ in linear time. Let R be the set of vertices containing rigid tokens and B the set of blocking arcs. Let T' be the polyforest obtained from T by removing each arc f such that f is incident to a vertex in R, or $f \notin B$ and f is incident to an endpoint of $e \in B$. Then, every component of T' is either an isolated vertex in R, a component containing the two vertices connected by an arc in B, or a component without any rigid tokens or blocking arcs. The following lemma reduces our problem to a slightly simplified one.

▶ **Lemma 11** ($\star$)**.** *Suppose that for every $v \in R$, there is no $u \in N(v)$ such that T has an arc $e \in \Gamma(u)$ with $w(e) > 0$. Then, $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance if and only if $(T', I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance.*

It is easy to see that if T' has more than one connected components, then we can solve the problem independently on each connected component. Moreover, the problem is trivial on a component of size at most two. Hence, we can assume that the input polytree has no vertices on which rigid tokens are placed or blocking arcs.

4.3 Necessary and sufficient conditions for yes-instances

In this subsection, we show the necessary and sufficient conditions for yes-instances. By Lemma 11, we may assume that the instance does not contain rigid tokens and blocking arcs. For a yes-instance, pick some reconfiguration sequence and let P_i be a directed path along which the i-th token moves in the reconfiguration sequence. $\mathcal{P} = \{P_1, \dots, P_k\}$ is obviously a directed path matching from I^{s} to I^{t}. Since the instance does not contain rigid tokens and blocking arcs, we have $|P_i| \geq 2$ for every $i \in [k]$. In addition, the successors of source vertices in $\mathcal{P}$ must be distinct. To see this, observe that if the source vertices of two paths P_i and P_j in $\mathcal{P}$ have a common successor $x \coloneqq s'(P_i) = s'(P_j)$, then the tokens on $s(P_i)$ and $s(P_j)$ are both "gazing" the vertex x, and thus we cannot slide either of the tokens on $s(P_i)$ and $s(P_j)$ at all. Therefore, $\mathcal{P}$ must satisfy that $s'(P_i) \neq s'(P_j)$ for all $i, j \in [k]$ with $i \neq j$. Symmetrically, the predecessors of sink vertices must be distinct, that is, $t'(P_i) \neq t'(P_j)$ for all $i, j \in [k]$ with $i \neq j$. The goal of this subsection is to show that these necessary conditions are also sufficient.

▶ **Lemma 12.** *Let $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ be an instance of* Directed Token Sliding *without rigid tokens and blocking arcs. Then $(T, I^{\mathsf{s}}, I^{\mathsf{t}})$ is a yes-instance if and only if there exists a set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of directed paths satisfying all the following conditions:*

(P1) $|P_i| \geq 2$ *for all* $i \in [k]$,

(P2) $\mathcal{P}$ *is a directed path matching from* I^{s} *to* I^{t},

(P3) $s'(P_i) \neq s'(P_j)$ *for all* $i, j \in [k]$ *with* $i \neq j$, *and*

(P4) $t'(P_i) \neq t'(P_j)$ *for all* $i, j \in [k]$ *with* $i \neq j$.

We refer to the four conditions (P1)–(P4) as the *path-set conditions.*

We have already seen the only-if direction as above. In the following, we show the other direction. We call a vertex in a path other than the source or the sink an *internal vertex.* For vertices $u, v \in V$, we define $\mathrm{dist}(u, v)$ as the length of the unique directed u-v path in T if such a path exists. We refer to a pair of directed paths P and P' satisfying the following conditions as a *biased path pair*:

- P and P' have a common internal vertex x,
- $\mathrm{dist}(s(P), x) > \mathrm{dist}(s(P'), x)$ and $\mathrm{dist}(x, t(P)) > \mathrm{dist}(x, t(P'))$.

▶ **Lemma 13** ($\star$). *If there exists a set of paths satisfying the path-set conditions, then there also exists a set of paths that satisfies the path-set conditions and does not include any biased path pair.*

In the following, let $\mathcal{P}^* = \{P_1^*, \ldots, P_k^*\}$ be a set of paths that satisfies the path-set conditions and has no biased path pairs. We show that there exists a bijection $\pi : [k] \to [k]$ having the following property $(*)$:

$(*)$ I^{s} is reconfigurable into I^{t} in a path-by-path manner following π; that is, by first moving the $\pi(1)$-th token from $s(P^*_{\pi(1)})$ all the way to $t(P^*_{\pi(1)})$ along $P^*_{\pi(1)}$, then moving the $\pi(2)$-th token from $s(P^*_{\pi(2)})$ all the way to $t(P^*_{\pi(2)})$ along $P^*_{\pi(2)}$, and so on.

From now on, we consider how to construct π satisfying $(*)$. For a vertex v and a directed path P, we say that v *touches* P or P *touches* v if $N[v] \cap V(P) \neq \emptyset$, where $V(P)$ denotes the set of vertices in P. For $i \neq j$, let (A) and (B) be the following conditions:

(A) $s(P_i^*)$ touches P_j^*;

(B) $t(P_j^*)$ touches P_i^*.

A binary relation $\dashleftarrow$ is defined as: $i \dashleftarrow j$ if and only if at least one of (A) and (B) holds for different i and j. Let $G_{\dashleftarrow}$ be the directed graph such that the vertex set of $G_{\dashleftarrow}$ is $[k]$ and, for $i, j \in [k]$, $G_{\dashleftarrow}$ has the arc (i, j) if and only if $j \dashleftarrow i$ holds.

If $G_{\dashleftarrow}$ is a directed acyclic graph, then we can construct π satisfying $(*)$ as follows. Since $G_{\dashleftarrow}$ is a DAG, there is a vertex i such that its out-degree is 0. For any $j \neq i$, every vertex in P_i^* does not belong to $N[s(P_j^*)]$ since P_i^* does not touch $s(P_j^*)$, and every vertex in P_j^* does not belong to $N[t(P_i^*)]$ since P_j^* does not touch $t(P_i^*)$. The former implies that the token on $s(P_i^*)$ sliding to $t(P_i^*)$ along P_i^* does not make adjacent token pairs and hence forms a reconfiguration sequence from I^{s} to $I^{\mathsf{s}} \setminus \{s(P_i^*)\} \cup \{t(P_i^*)\}$. The latter implies that the graph $G_{\dashleftarrow}$ for the resulting independent set $I^{\mathsf{s}} \setminus \{s(P_i^*)\} \cup \{t(P_i^*)\}$ is obtained by deleting the vertex i, which is still a DAG. By repeating the above procedure, I^{s} is reconfigurable into I^{t} in a path-by-path manner. Thus a bijection $\pi : [k] \to [k]$ satisfying $\pi(1) < \pi(2) < \cdots < \pi(k)$ with respect to a topological order of $G_{\dashleftarrow}$ admits $(*)$, as required.

The following lemma says that $G_{\dashleftarrow}$ is actually a directed acyclic graph, which verifies the if-condition of Lemma 12.

▶ **Lemma 14.** *$G_{\dashleftarrow}$ is a directed acyclic graph.*

Proof. Suppose, to the contrary, that there is a directed cycle in $G_{\dashleftarrow}$. We can assume that $(1, 2, \ldots, \ell, 1)$ is a minimal one. For convenience, the addition $+$ and subtraction $-$ are taken over modulo ℓ, i.e., $\ell + 1$ is regarded as 1 and 0 is regarded as ℓ.

Suppose moreover that P^*_i and P^*_{i+1} have a common internal vertex for all $i \in [k]$. Let x and y be the source and the sink of the maximal common subpath of P^*_i and P^*_{i+1}, respectively. Since T is a polytree, these x and y are uniquely determined. For $i \in [\ell]$, at least one of (A) and (B) holds. If (A) $s(P^*_i)$ touches P^*_{i+1}, then we have $\mathrm{dist}(s(P^*_i), x) \leq 1$. If $\mathrm{dist}(s(P^*_i), x) \geq \mathrm{dist}(s(P^*_{i+1}), x)$, either $s'(P^*_i) = s'(P^*_{i+1})$ or $s(P^*_i)$ and $s(P^*_{i+1})$ are adjacent, which do not hold as (P3) or the fact that I^{s} is an independent set. Thus, we have $\mathrm{dist}(s(P^*_i), x) < \mathrm{dist}(s(P^*_{i+1}), x)$. If (B) $t(P^*_{i+1})$ touches P^*_i, then we have $\mathrm{dist}(y, t(P^*_{i+1})) \leq 1$. Then, by a symmetric argument, we have $\mathrm{dist}(y, t(P^*_i)) > \mathrm{dist}(y, t(P^*_{i+1}))$. Let z_i be an arbitrary common internal vertex of P^*_i and P^*_{i+1}. The above argument is summarised as

$$\mathrm{dist}(s(P^*_i), z_i) < \mathrm{dist}(s(P^*_{i+1}), z_i) \text{ or } \mathrm{dist}(z_i, t(P^*_i)) > \mathrm{dist}(z_i, t(P^*_{i+1})). \tag{1}$$

Equation (1) and the non-existence of biased pairs together imply that for every i, we have

$$\mathrm{dist}(s(P^*_i), z_i) \leq \mathrm{dist}(s(P^*_{i+1}), z_i) \text{ and } \mathrm{dist}(z_i, t(P^*_i)) \geq \mathrm{dist}(z_i, t(P^*_{i+1})). \tag{2}$$

For a walk W in T^{und}, let $\overrightarrow{W}$ be an oriented walk obtained from W by orienting each edge from (arbitrary) one of the end vertices to the other. Let $\mathrm{fwd}(\overrightarrow{W})$ and $\mathrm{rev}(\overrightarrow{W})$ be the numbers of times that $\overrightarrow{W}$ passes through arcs in the forward and reverse directions in T, respectively. Here, by the fact that $\mathrm{dist}(s(P^*_i), z_i) \leq \mathrm{dist}(s(P^*_{i+1}), z_i)$ (Equation (2)), there exists an oriented walk $\overrightarrow{W}$ from $s(P^*_i)$ to $s(P^*_{i+1})$ satisfying $\mathrm{fwd}(\overrightarrow{W}) \leq \mathrm{rev}(\overrightarrow{W})$. This can be obtained by traversing T^{und} from $s(P^*_i)$ to $s(P^*_{i+1})$ via z_i (which we denote by $s(P^*_i) \to z_i \to s(P^*_{i+1})$). In particular, if $\mathrm{dist}(s(P^*_i), z_i) < \mathrm{dist}(s(P^*_{i+1}), z_i)$, then $\mathrm{fwd}(\overrightarrow{W}) < \mathrm{rev}(\overrightarrow{W})$ holds. Suppose that $\mathrm{dist}(s(P^*_1), z_1) < \mathrm{dist}(s(P^*_2), z_1)$ holds. Then, the closed oriented walk

$$\overrightarrow{W_s} \coloneqq s(P^*_1) \to z_1 \to s(P^*_2) \to \cdots \to s(P^*_\ell) \to z_\ell \to s(P^*_1)$$

satisfies $\mathrm{fwd}(\overrightarrow{W_s}) < \mathrm{rev}(\overrightarrow{W_s})$. See Figure 1 for an illustration. This implies that there is an arc e in T such that e occurs in $\overrightarrow{W_s}$ at least once and the number of occurrences of e is strictly smaller than that of the reverse of e in $\overrightarrow{W_s}$. This contradicts the fact that T is a polytree. Thus suppose $\mathrm{dist}(s(P^*_1), z_1) \geq \mathrm{dist}(s(P^*_2), z_1)$ holds. Then $\mathrm{dist}(z_1, t(P^*_1)) > \mathrm{dist}(z_1, t(P^*_2))$ follows from Equation (1). By a similar argument as above, we can construct a closed oriented walk $\overrightarrow{W_t}$ from $t(P^*_1)$ to $t(P^*_1)$ satisfying $\mathrm{fwd}(\overrightarrow{W_t}) > \mathrm{rev}(\overrightarrow{W_t})$. This also contradicts the fact that T is a polytree. Thus, we derive a contradiction, assuming that $G_{\dashleftarrow}$ has a directed cycle and P^*_i and P^*_{i+1} have a common internal vertex for all $i \in [k]$.

The remaining task is to show that P^*_i and P^*_{i+1} have a common internal vertex for all i under the assumption that $G_{\dashleftarrow}$ has a directed cycle. Suppose to the contrary that P^*_i and P^*_{i+1} have no common internal vertex. We only consider the case where the condition (A) in the definition of $\dashleftarrow$ holds for $i \dashleftarrow i{+}1$, that is, P^*_{i+1} touches $s(P^*_i)$; the case for (B) is symmetric. As $|P^*_i| \geq 2$, P^*_i has an arc $(s'(P^*_i), x^*)$. See Figure 2 for an illustration. For $x \in N_T(s'(P^*_i))$, we denote by C_x the weakly connected component containing x in the polyforest obtained from T by deleting the arc $(s'(P^*_i), x)$. Then observe that $V(P^*_{i+1}) \cap C_{x^*} = \emptyset$. To see this, if $V(P^*_{i+1}) \cap C_{x^*} \neq \emptyset$, then P^*_{i+1} must have the arc $(s'(P^*_i), x^*)$ as P^*_{i+1} touches $s(P^*_i)$. This particularly implies that $s'(P^*_i)$ belongs to P^*_{i+1} and is different from $t(P^*_{i+1})$ due to the direction of arc $(s'(P^*_i), x^*)$. Since I^{s} is an independent set, we have $s'(P^*_i) \neq s(P^*_{i+1})$. Thus P^*_i and P^*_{i+1} have a common internal vertex $s'(P^*_i)$, contradicting the assumption that P^*_i and P^*_{i+1} have no common internal vertex. Hence we obtain $V(P^*_{i+1}) \cap C_{x^*} = \emptyset$.

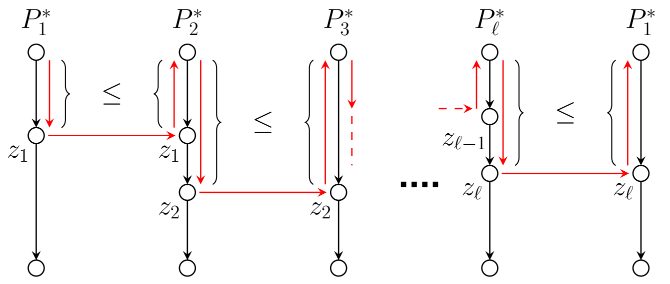


Figure 1 Closed oriented walk $s(P_1^*) \to z_1 \to s(P_2^*) \to \cdots \to s(P_\ell^*) \to z_\ell \to s(P_1^*)$.

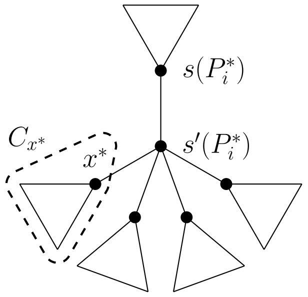


Figure 2 An illustration of a polytree considered in the proof of Lemma 14.

Suppose $i-1 = i+1$, i.e., $i+1 \dashleftarrow i$ (which implies $\ell = 2$). Then P_i^* touches $s(P_{i+1}^*)$ or P_{i+1}^* touches $t(P_i^*)$. In the former case, since $V(P_{i+1}^*) \cap C_{x^*} = \emptyset$ and I^{s} is an independent set, $s(P_{i+1}^*)$ must belong to $N_T(s'(P_i^*))$. Then we have $s'(P_{i+1}^*) = s'(P_i^*)$ by $i \dashleftarrow i+1$, which contradicts (P3). In the latter case, since $V(P_{i+1}^*) \cap C_{x^*} = \emptyset$, we have $x^* = t(P_i^*)$ and $s'(P_i^*) \in V(P_{i+1}^*)$. Moreover, neither $s'(P_i^*) = s(P_{i+1}^*)$ nor $s'(P_i^*) = t(P_{i+1}^*)$, since I^{s} and I^{t} are independent sets. Thus $s'(P_i^*)$ must be an internal vertex of P_{i+1}^*, contradicting the assumption.

Suppose $i-1 \neq i+1$ (which implies $\ell \geq 3$). Observe that $V(P_{i-1}^*) \cap C_{s(P_i^*)} = \emptyset$. To see this, suppose $V(P_{i-1}^*) \cap C_{s(P_i^*)} \neq \emptyset$. As $i-1 \dashleftarrow i$, either $s(P_{i-1}^*)$ touches P_i^* or $t(P_i^*)$ touches P_{i-1}^*. In both cases, $s(P_i^*)$ touches P_{i-1}^* and hence $i \dashleftarrow i-1$, contradicting the minimality of the cycle $(1, 2, \ldots, \ell, 1)$. Observe also that $s'(P_i^*) \notin V(P_{i-1}^*)$ as otherwise we obtain $i \dashleftarrow i-1$, which again contradicts the minimality of the cycle $(1, 2, \ldots, \ell, 1)$. Here we additionally distinguish the two cases: (i) $s'(P_i^*) \in V(P_{i+1}^*)$ and (ii) $s'(P_i^*) \notin V(P_{i+1}^*)$.

(i) $s'(P_i^*) \in V(P_{i+1}^*)$. Since P_i^* and P_{i+1}^* have no common internal vertices, we have $t(P_{i+1}^*) = s'(P_i^*)$. By the minimality of the cycle $(1, 2, \ldots, \ell, 1)$, P_{i-1}^* has none of vertices in $N_T[s'(P_i^*)]$. This implies, together with $i-1 \dashleftarrow i$ and $V(P_{i-1}^*) \cap C_{s(P_i^*)} = \emptyset$, that $V(P_{i-1}^*) \subseteq C_{x^*}$ and $x^* \notin V(P_{i-1}^*)$. By $i+1 \dashleftarrow i+2 \dashleftarrow \cdots \dashleftarrow i-1$, there is an index m with $i-1 \neq m \neq i+1$ such that $x^* \in V(P_m^*)$. This implies $m \dashleftarrow i+1$, contradicting the minimality of the cycle.

(ii) $s'(P_i^*) \notin V(P_{i+1}^*)$. In this case, P_{i+1}^* has a vertex in $N[s(P_i^*)] \setminus \{s'(P_i^*)\}$ and does not have the arc $(s(P_i^*), s'(P_i^*))$. Thus we have $V(P_{i+1}^*) \subseteq C_{s(P_i^*)}$. By $i+1 \dashleftarrow i+2 \dashleftarrow \cdots \dashleftarrow i-1$, there is an index m with $i-1 \neq m \neq i+1$ such that $s'(P_i^*) \in V(P_m^*)$. This implies $i \dashleftarrow m$, contradicting the minimality of the cycle.

This completes the proof of Lemma 14. ◀

4.4 Algorithms

In this subsection, we provide an algorithm for checking the reconfigurability in $\mathrm{O}(|V|)$ time, and that for constructing a reconfiguration sequence in $\mathrm{O}(k|V|)$ time, where k is the size of the input independent set. (if the answer is affirmative), proving Theorem 3.

For $U \subseteq V$, we define $N^+(U)$ (resp. $N^-(U)$) as $N^+(U) := \bigcup_{u \in U} N^+(u) \setminus U$ (resp. $N^-(U) := \bigcup_{u \in U} N^-(u) \setminus U$). For a mapping $f \colon I^{\mathsf{s}} \to N^+(I^{\mathsf{s}})$, let $f(I^{\mathsf{s}})$ denote the image of f, i.e., $f(I^{\mathsf{s}}) := \{f(x) \mid x \in I^{\mathsf{s}}\}$. Similarly, the image of $g \colon I^{\mathsf{t}} \to N^-(I^{\mathsf{t}})$ is denoted as $g(I^{\mathsf{t}})$.

4.4.1 Algorithm for checking the reconfigurability

Suppose that there is a set $\mathcal{P} = \{P_1, \dots, P_k\}$ of directed paths satisfying the path-set conditions. Then the set of paths obtained from $\mathcal{P}$ by exchanging each P_i with the subpath from $s'(P_i)$ to $t'(P_i)$ is a directed path matching from $\{s'(P_i) \mid i \in [k]\}$ to $\{t'(P_i) \mid i \in [k]\}$. In this case, the mappings $f \colon I^{\mathsf{s}} \to N^+(I^{\mathsf{s}})$ and $g \colon I^{\mathsf{t}} \to N^-(I^{\mathsf{t}})$ defined by $f(s(P_i)) := s'(P_i)$ and $g(t(P_i)) := t'(P_i)$, respectively, satisfy the following four conditions:

(C1) f and g are injective,
(C2) $(s, f(s)) \in A$ for all $s \in I^{\mathsf{s}}$,
(C3) $(g(t), t) \in A$ for all $t \in I^{\mathsf{t}}$, and
(C4) $w(e; f(I^{\mathsf{s}}), g(I^{\mathsf{t}})) \geq 0$ for each arc e.

In particular, (C1) follows from (P3) and (P4), and (C4) follows from Lemma 5.

Conversely, if there are mappings $f \colon I^{\mathsf{s}} \to N^+(I^{\mathsf{s}})$ and $g \colon I^{\mathsf{t}} \to N^-(I^{\mathsf{t}})$ satisfying the above four conditions (C1)–(C4), then we can construct a set of directed paths satisfying the path-set conditions (P1)–(P4) as follows. By Lemma 5 and (C4), there exists a directed path matching $\mathcal{P}' = \{P'_1, \dots, P'_k\}$ from $f(I^{\mathsf{s}})$ to $g(I^{\mathsf{t}})$. Define a set $\mathcal{P} = \{P_1, \dots, P_k\}$ of paths from $\mathcal{P}'$ by appending $f^{-1}(s(P'_i))$ and $g^{-1}(t(P'_i))$ before the source and after the sink for each P'_i, respectively. Then $|P_i| \geq 2$ for each $i \in [k]$, and $\mathcal{P}$ is a directed path matching from I^{s} and I^{t}. Moreover, since $\mathcal{P}'$ is a directed path matching from $f(I^{\mathsf{s}})$ to $g(I^{\mathsf{t}})$, we have $s'(P_i) \neq s'(P_j)$ and $t'(P_i) \neq t'(P_j)$ for $i \neq j$, implying that $\mathcal{P}$ satisfies the path-set conditions.

By the above argument, we obtain the following necessary and sufficient conditions for the reconfigurability:

▶ **Lemma 15.** *For an instance without rigid tokens and blocking arcs, there exists a set of paths satisfying the path-set conditions if and only if there exist mappings $f \colon I^{\mathsf{s}} \to N^+(I^{\mathsf{s}})$ and $g \colon I^{\mathsf{t}} \to N^-(I^{\mathsf{t}})$ satisfying* (C1)*–*(C4)*.*

By Lemmas 12 and 15, for checking the reconfigurability it suffices to compute f and g satisfying (C1)–(C4) or determine that such f and g do not exist; it can be done in $\mathrm{O}(|V|)$ time in a greedy manner as follows. In the following description, we regard T as a rooted tree with an arbitrary root. We initialize the values of f and g as $f(s) := \bot$ for all $s \in I^{\mathsf{s}}$ and $g(t) := \bot$ for all $t \in I^{\mathsf{t}}$ ($\bot$ means “undefined”). To let f and g satisfy the conditions in Lemma 15, we define each value of f and g one by one. We write $I^{\mathsf{s}}_{\mathrm{undef}} \subseteq I^{\mathsf{s}}$ (resp. $I^{\mathsf{s}}_{\mathrm{def}} \subseteq I^{\mathsf{s}}$) as the set of the vertices for which the values of f have been undefined (resp. defined). We also define $I^{\mathsf{t}}_{\mathrm{undef}}$ and $I^{\mathsf{t}}_{\mathrm{def}}$ in the same way. Throughout the following process to determine f and g, we keep an invariant that the conditions (C1)–(C3) hold for $s \in I^{\mathsf{s}}_{\mathrm{def}}$ and $t \in I^{\mathsf{t}}_{\mathrm{def}}$, moreover, for each arc e, $w(e; I^{\mathsf{s}}_{\mathrm{undef}} \cup f(I^{\mathsf{s}}_{\mathrm{def}}), I^{\mathsf{t}}_{\mathrm{undef}} \cup g(I^{\mathsf{t}}_{\mathrm{def}})) \geq 0$.

In the following, we describe each step of the algorithm. Let $v^* \in I^{\mathsf{s}}_{\mathrm{undef}} \cup I^{\mathsf{t}}_{\mathrm{undef}}$ be a vertex with the largest depth among $I^{\mathsf{s}}_{\mathrm{undef}} \cup I^{\mathsf{t}}_{\mathrm{undef}}$ in the rooted tree. If $v^* \in I^{\mathsf{s}}_{\mathrm{undef}}$, we define

$$N_{\mathrm{cand}}(v^*) := \left\{u \in N^+(v^*) \setminus f(I^{\mathsf{s}}_{\mathrm{def}}) \;\middle|\; w((v^*, u); I^{\mathsf{s}}_{\mathrm{undef}} \cup f(I^{\mathsf{s}}_{\mathrm{def}}), I^{\mathsf{t}}_{\mathrm{undef}} \cup g(I^{\mathsf{t}}_{\mathrm{def}})) \geq 1\right\}.$$

Otherwise, i.e., $v^* \in I^{\mathsf{t}}_{\mathrm{undef}} \setminus I^{\mathsf{s}}_{\mathrm{undef}}$, define

$$N_{\mathrm{cand}}(v^*) := \left\{u \in N^-(v^*) \setminus g(I^{\mathsf{t}}_{\mathrm{def}}) \;\middle|\; w((u, v^*); I^{\mathsf{s}}_{\mathrm{undef}} \cup f(I^{\mathsf{s}}_{\mathrm{def}}), I^{\mathsf{t}}_{\mathrm{undef}} \cup g(I^{\mathsf{t}}_{\mathrm{def}})) \geq 1\right\}.$$

If $N_{\mathrm{cand}}(v^*)$ is empty, terminate the execution. Otherwise, choose $u^* \in N_{\mathrm{cand}}(v^*)$ with the largest depth among $N_{\mathrm{cand}}(v^*)$ and define $f(v^*) := u^*$ if $v^* \in I^{\mathsf{s}}_{\mathrm{undef}}$ and define $g(v^*) := u^*$ otherwise. When $f(v^*)$ or $g(v^*)$ is newly defined in this step, we accordingly update $I^{\mathsf{s}}_{\mathrm{def}}$, $I^{\mathsf{s}}_{\mathrm{undef}}$, $I^{\mathsf{t}}_{\mathrm{def}}$, and $I^{\mathsf{t}}_{\mathrm{undef}}$.

Note that each vertex in $I^{\mathsf{s}} \cup I^{\mathsf{t}}$ is selected as v^* at most twice. After the execution of the above algorithm, output f and g if $f(s) \neq \bot$ for all $s \in I^{\mathsf{s}}$ and $g(t) \neq \bot$ for all $t \in I^{\mathsf{t}}$; otherwise, we conclude that no such f and g exist. In each iteration, only one arc (v^*, u^*) or (u^*, v^*) decreases its w-value by 1: If $v^* \in I^{\mathsf{s}}_{\mathrm{undef}}$,

$$\begin{aligned} &w(e; I^{\mathsf{s}}_{\mathrm{undef}} \setminus \{v^*\} \cup f(I^{\mathsf{s}}_{\mathrm{def}}) \cup \{u^*\}, I^{\mathsf{t}}_{\mathrm{undef}} \cup g(I^{\mathsf{t}}_{\mathrm{def}})) \\ &= \begin{cases} w(e; I^{\mathsf{s}}_{\mathrm{undef}} \cup f(I^{\mathsf{s}}_{\mathrm{def}}), I^{\mathsf{t}}_{\mathrm{undef}} \cup g(I^{\mathsf{t}}_{\mathrm{def}})) & \text{if } e \neq (v^*, u^*) \\ w(e; I^{\mathsf{s}}_{\mathrm{undef}} \cup f(I^{\mathsf{s}}_{\mathrm{def}}), I^{\mathsf{t}}_{\mathrm{undef}} \cup g(I^{\mathsf{t}}_{\mathrm{def}})) - 1 & \text{if } e = (v^*, u^*); \end{cases} \end{aligned}$$

and otherwise $w(e; I^{\mathsf{s}}_{\mathrm{undef}} \cup f(I^{\mathsf{s}}_{\mathrm{def}}), I^{\mathsf{t}}_{\mathrm{undef}} \setminus \{v^*\} \cup g(I^{\mathsf{t}}_{\mathrm{def}}) \cup \{u^*\})$ can be computed in a symmetric fashion. Using this property, we can implement the algorithm that runs in $\mathrm{O}(|V|)$ time. More precisely, we maintain the values of w and the indicator functions of $I^{\mathsf{s}}_{\mathrm{def}}$ and $I^{\mathsf{t}}_{\mathrm{def}}$ in tables. By referring the tables, we can compute $N_{\mathrm{cand}}(v^*)$ in $\mathrm{O}(|N(v^*)|)$ time. Moreover, we can update the tables in $\mathrm{O}(1)$ time. Therefore each iteration runs in $\mathrm{O}(|N(v^*)|)$ time, and thus the algorithm runs in total in $\mathrm{O}(|V|)$ time since each edge is touched twice.

▶ **Lemma 16** ($\star$). *If there exist mappings f and g that satisfy the conditions in Lemma 15, then the algorithm described above correctly outputs one of such mappings. Otherwise the algorithm reports that no such mappings exist.*

4.4.2 Algorithm for constructing a reconfiguration sequence

By Corollary 6, if the instance is a yes-instance, all reconfiguration sequences have the same length. In this subsection, we present an algorithm to construct one of them in $\mathrm{O}(k|V|)$ time, where k is the size of the input independent set. We first assume that two mappings f and g satisfying conditions (C1)–(C4) have already been computed. If such f and g do not exist, the instance is a no-instance by Lemma 15. Here, our target is to fill in gaps between $f(I^{\mathsf{s}})$ and $g(I^{\mathsf{t}})$ avoiding biased path pairs. For preparation of further arguments, we define *weakly biased path pairs* by relaxing "common internal vertex" to "common vertex" in the definition of biased path pairs. If there is a directed path matching $\mathcal{P} = \{P_1, \dots, P_k\}$ from $f(I^{\mathsf{s}})$ to $g(I^{\mathsf{t}})$ without any weakly biased path pairs (not necessarily satisfying the path-set conditions), we can easily construct a directed path matching $\mathcal{P}' = \{P'_1, \dots, P'_k\}$ from I^{s} to I^{t} satisfying the path-set conditions: P'_i is obtained from P_i by appending $f^{-1}(s(P_i))$ before P_i and $g^{-1}(t(P_i))$ after P_i. Thus, in the following, it suffices to show that $\mathcal{P}$ can be constructed in $\mathrm{O}(k|V|)$ time.

Fix an arbitrary vertex $r \in V$. For each $v \in V$, we denote by $P_{r,v}$ the unique path between r and v in T^{und}. Let $\overrightarrow{P_{r,v}}$ be a directed path obtained by orienting each edge of $P_{r,v}$ from r to v. We define $d_r(v) = \mathrm{fwd}(\overrightarrow{P_{r,v}}) - \mathrm{rev}(\overrightarrow{P_{r,v}})$, where $\mathrm{fwd}(\overrightarrow{P_{r,v}})$ and $\mathrm{rev}(\overrightarrow{P_{r,v}})$ denote the numbers of arcs in T that $\overrightarrow{P_{r,v}}$ passes in the forward and reverse directions, respectively.

Using the values of $d_r(v)$ for $v \in V$, we iteratively construct a directed path matching $\mathcal{P} = \{P_1, \dots, P_k\}$ from $f(I^{\mathsf{s}})$ to $g(I^{\mathsf{t}})$ as follows. In the i-th step of the algorithm, we construct P_i. Let $X_i \subseteq f(I^{\mathsf{s}})$ and $Y_i \subseteq g(I^{\mathsf{t}})$ be the sets of vertices that are not chosen for the sources and sinks of $P_1, \dots, P_{i-1}$, respectively. Throughout the execution of the algorithm, we keep an invariant that for each arc e, $w(e; X_i, Y_i) \geq 0$, which means that there is a directed path matching from X_i to Y_i by Lemma 5.

Let $x \in X_i$ be a vertex with the smallest value of $d_r(x)$ among X_i. Let $Y'_i \subseteq Y_i$ be the set of vertices to which there is a directed path from x consisting of only arcs with $w(e; X_i, Y_i) > 0$. Since there exists some directed path matching from X_i to Y_i by our invariant that $w(e, X_i, Y_i) \geq 0$ for $e \in A$, the set Y'_i is not empty. Choose an arbitrary vertex $y \in Y_i'$ with the smallest value of $d_r(y)$. We set P_i to the unique directed path from x to y in T. Since we use only arcs e with $w(e; X_i, Y_i) > 0$ to construct P_i, we have $w(e; X_{i+1}, Y_{i+1}) \geq 0$, where $X_{i+1} = X_i \setminus \{x\}$ and $Y_{i+1} = Y_i \setminus \{y\}$, for all arcs and the invariant still holds. We repeat this until $X_i = \emptyset$ (and hence $Y_i = \emptyset$).

Here, we show that the obtained directed path matching $\{P_1, \ldots, P_k\}$ does not contain any weakly biased path pairs. Assume that P_i and P_j $(i < j)$ have a common vertex v. By the choice of x in the construction above, $d_r(s(P_i)) \leq d_r(s(P_j))$ holds. Thus, we have

$$\mathrm{dist}(s(P_i), v) = -d_r(s(P_i)) + d_r(v) \geq -d_r(s(P_j)) + d_r(v) = \mathrm{dist}(s(P_j), v).$$

As v is a common vertex of P_i and P_j, $t(P_j) \in Y_i'$ and thus $d_r(t(P_i)) \leq d_r(t(P_j))$ holds by the choice of y. Then, similarly we have

$$\mathrm{dist}(v, t(P_i)) = -d_r(v) + d_r(t(P_i)) \leq -d_r(v) + d_r(t(P_j)) = \mathrm{dist}(v, t(P_j)).$$

Therefore, P_i and P_j do not form a weakly biased path pair. We can compute each P_i in $\mathrm{O}(|V|)$ time, and thus the whole running time (including computing f and g) is $\mathrm{O}(k|V|)$.

References

1 Oswin Aichholzer, Jean Cardinal, Tony Huynh, Kolja Knauer, Torsten Mütze, Raphael Steiner, and Birgit Vogtenhuber. Flip distances between graph orientations. *Algorithmica*, 83:116–143, 2021. doi:10.1007/s00453-020-00751-1.

2 Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *LIPIcs*, pages 44:1–44:17, 2020. doi:10.4230/LIPIcs.ISAAC.2020.44.

3 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory of Computing Systems*, 65:662–686, 2021. doi:10.1007/s00224-020-09967-8.

4 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2017)*, volume 10520 of *LNCS*, pages 127–139, 2017. doi:10.1007/978-3-319-68705-6_10.

5 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *LIPIcs*, pages 80:1–80:14, 2019. doi:10.4230/LIPIcs.MFCS.2019.80.

6 Marthe Bonamy, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Moritz Mühlenthaler, Akira Suzuki, and Kunihiro Wasa. Diameter of colorings under Kempe changes. *Theoretical Computer Science*, 838:45–57, 2020. doi:10.1016/j.tcs.2020.05.033.

7 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009. doi:10.1016/j.tcs.2009.08.023.

8 Paul S. Bonsma, Marcin Kaminski, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2014)*, volume 8503 of *LNCS*, pages 86–97, 2014. doi:10.1007/978-3-319-08404-6_8.

9 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenthaler. Shortest reconfiguration of matchings. In *Proceedings of the 45th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2019)*, volume 11789 of *LNCS*, pages 162–174, 2019. doi:10.1007/978-3-030-30786-8_13.

10 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token jumping in minor-closed classes. In *Proceedings of the 21st International Symposium on Fundamentals of Computation Theory (FCT 2017)*, volume 10472 of *LNCS*, pages 136–149, 2017. doi:10.1007/978-3-662-55751-8_12.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

12 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.

13 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC 2015)*, volume 9472 of *LNCS*, pages 237–247, 2015. doi:10.1007/978-3-662-48971-0_21.

14 Komei Fukuda, Alain Prodon, and Tadashi Sakuma. Notes on acyclic orientations and the shelling lemma. *Theoretical Computer Science*, 263(1-2):9–16, 2001. doi:10.1016/S0304-3975(00)00226-7.

15 Curtis Greene and Thomas Zaslavsky. On the interpretation of Whitney numbers through arrangements of hyperplanes, zonotopes, non-Radon partitions, and orientations of graphs. *Transactions of the American Mathematical Society*, 280(1):97–126, 1983.

16 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016. doi:10.1016/j.tcs.2016.08.016.

17 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

18 Duc A. Hoang and Ryuhei Uehara. Sliding tokens on a cactus. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *LIPIcs*, pages 37:1–37:26, 2016. doi:10.4230/LIPIcs.ISAAC.2016.37.

19 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

20 Takehiro Ito, Yuni Iwamasa, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, Shun ichi Maezawa, Yuta Nozaki, Yoshio Okamoto, and Kenta Ozeki. Monotone edge flips to an orientation of maximum edge-connectivity à la Nash-Williams. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 1342–1355, 2022.

21 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, Masahiro Takahashi, and Kunihiro Wasa. Independent set reconfiguration on directed graphs. *CoRR*, abs/2203.13435, 2022. doi:10.48550/arXiv.2203.13435.

22 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, and Kunihiro Wasa. Reconfiguring directed trees in a digraph. In *Proceedings of the 27th International Computing and Combinatorics Conference (COCOON 2021)*, volume 13025 of *LNCS*, pages 343–354, 2021.

23 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *LIPIcs*, pages 61:1–61:15, 2019. doi:10.4230/LIPIcs.ESA.2019.61.

24 Takehiro Ito, Marcin Kaminski, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In *Proceedings of the 11th Annual Conference on Theory and Applications of Models of Computation (TAMC 2014)*, volume 8402 of *LNCS*, pages 341–351, 2014. doi:10.1007/978-3-319-06089-7_24.

25 Takehiro Ito, Marcin Jakub Kaminski, and Hirotaka Ono. Fixed-parameter tractability of token jumping on planar graphs. In *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, volume 8889 of *LNCS*, pages 208–219, 2014. doi:10.1007/978-3-319-13075-0_17.

26 Takehiro Ito, Marcin Jakub Kaminski, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. Parameterized complexity of independent set reconfiguration problems. *Discrete Applied Mathematics*, 283:336–345, 2020. doi:10.1016/j.dam.2020.01.022.

27 Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. In *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC 2015)*, volume 9076 of *LNCS*, pages 212–223, 2015. doi:10.1007/978-3-319-17142-5_19.

28 Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

29 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Transactions on Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.

30 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, 2018. doi:10.1016/j.jcss.2018.02.004.

31 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

32 Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Algorithms for coloring reconfiguration under recolorability constraints. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *LIPIcs*, pages 37:1–37:13, 2018. doi:10.4230/LIPIcs.ISAAC.2018.37.

33 Ken Sugimori. A polynomial-time algorithm for shortest reconfiguration of sliding tokens on a tree. Master's thesis, The University of Tokyo, 2019. (In Japanese).

34 Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. Reconfiguration of dominating sets. *Journal of Combinatorial Optimization*, 32(4):1182–1195, 2016. doi:10.1007/s10878-015-9947-x.

35 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

Automating OBDD proofs is NP-hard

Dmitry Itsykson
Steklov Institute of Mathematics at St. Petersburg, Russia

Artur Riazanov
Steklov Institute of Mathematics at St. Petersburg, Russia

Abstract

We prove that the proof system OBDD($\wedge$, weakening) is not automatable unless P = NP. The proof is based upon the celebrated result of Atserias and Müller [5] about the hardness of automatability for resolution. The heart of the proof is lifting with multi-output indexing gadget from resolution block-width to dag-like multiparty number-in-hand communication protocol size with $o(n)$ parties, where n is the number of variables in the non-lifted formula. A similar lifting theorem for protocols with $n + 1$ participants was proved by Göös et. el. [12] to establish the hardness of automatability result for Cutting Planes.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases proof complexity, OBDD, automatability, lifting, dag-like communication

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.59

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2022/046/

Funding The research is supported by Russian Science Foundation (project 18-71-10042).

Acknowledgements The authors thank Anastasia Sofronova and Michal Garl´ık for fruitful discussions, Anastasia Sofronova for her comments on the draft. They also thank anonymous referees for their feedback. Artur Riazanov additionally thanks Dmitry Sokolov for his patient explanations of the lifting technique in [10].

1 Introduction

Boolean satisfiability is one of the central problems in Computer Science. The input to this problem is a CNF formula and the goal is to determine whether it is satisfiable or not. This is a standard example of an NP-complete problem, and it has been very thoroughly studied. While the consensus is that there is no polynomial algorithm for satisfiability, contemporary SAT-solvers have been quite successful in solving it for many instances appearing "in practice".

SAT-solvers are tightly connected to proof complexity. A propositional proof system is a formal way of certifying that a CNF formula is unsatisfiable. The execution log of an SAT-solver running on an unsatisfiable input φ can serve as a certificate of unsatisfiability of φ. Then SAT-solvers face the following trade-off: on the one hand, their underlying proof system must be sufficiently strong to have short proofs of all formulas of interest, on the other hand, it must be sufficiently weak so short proofs can be found fast. This tradeoff is witnessed by the success of CDCL-solvers, which are based on (subsystems of) Resolution which is a pretty weak proof system. Nevertheless, so far SAT-solvers based on stronger proof systems have not enjoyed the widespread success of resolution-based solvers.

A propositional proof system Π is called automatable (quasi-automatable) if there exists an algorithm $\mathcal{E}$ that given an unsatisfiable CNF φ produces a Π-proof of φ in time polynomial (quasi-polynomial) in size of φ plus the size of the shortest Π-proof of φ.

However, for many non-trivial proof systems, there are known pieces of evidence that they likely are not automatable or quasi-automatable. A long line of results on resolution automatability [15, 19, 2, 3] is concluded with the recent breakthrough result by Atserias and Müller [5] stating that resolution is not automatable unless P = NP and not quasi-automatable under a stronger assumption. This result sparked a series of follow-up results that establish the hardness of automating for many other proof systems; these results are either based on Atserias-Müller's result directly or follow their plan closely. If P $\neq$ NP, then the following proof systems are not automatable: Nullstellensatz and Polynomial Calculus [13]; Cutting Planes [12]; Res(2) [11]. Under stronger assumptions one can show non-automatability of Frege systems [17, 7, 6].

We continue this line of research and study the automatability of OBDD-based systems. OBDD (or ordered binary decision diagram) is a simple but rather expressive way to represent Boolean functions introduced by Bryant [8]. An OBDD is a very restricted case of a branching program, wherein for all paths from the source to a sink, variables appear in the same order. It is known that branching programs are at least as powerful as Boolean formulas, hence proving superpolynomial lower bounds on size of branching programs for explicit functions is an extremely hard problem. But exponential lower bounds are known for restricted versions of branching programs, including OBDDs. However, this restriction allows performing many important operations with OBDDs very efficiently: testing satisfiability, computing binary operations, applying restrictions, minimization, and so on. These properties have paved the way for OBDD-based propositional proof systems introduced by Atserias, Kolaites, and Vardi [4] to serve as a base for OBDD based SAT-solvers [1, 20].

An OBDD($\wedge$, weakening) refutation of a CNF φ is a sequence of OBDDs that query variables in the same order; the last OBDD in the sequence is identically false and each of those diagrams either represents a clause of φ or follows semantically from two OBDDs that appear earlier in the sequence (formally there are two rules: by the first ($\wedge$) we can derive conjunction of two OBDDs and by the second (weakening) we can derive any semantic implication of a single OBDD). The correctness of application of these rules can be efficiently verified since binary operations for two OBDDs with the same order of variab can be computed in polynomial time. This system simulates Resolution and CP* (Cutting Planes with unary coefficients); it has short refutations of unsatisfiable linear systems over $\mathbb{F}_2$ [4] and clique-coloring tautologies [9] (the latter are hard for Cutting Planes [22]).

Atserias-Müller's approach for establishing hardness of automatability requires proving a lower bound on the proof size of some specific CNF-formula. Unfortunately the tools for proving lower bounds on OBDD($\wedge$, weakening) are quite limited and related to monotone circuit complexity. All known lower bound proofs consist of two steps.

1. To prove the lower bound for a fixed order of variables in OBDDs. Such a lower bound was proved by Atserias et. al. [4]; an exponential lower bound on the size of OBDD($\wedge$, weakening) refutations of clique-coloring tautologies with a particular order of variables follows from monotone interpolation.
2. To transform a formula that is hard for one order into a formula that is hard for all orders. The first transformation of this kind was devised by Krajićek [16]: formulas are equipped with additional variables that parameterize a permutation of main variables such that by fixing these additional variables we can get the initial formula, where variables are permuted by any desired permutation. Segerlind [23, 24] invented a more concise transformation using 2-independent permutation family together with orification of variables; Segerlind used it to prove that OBDD($\wedge$, weakening) may require exponentially longer proofs than Res($O(\log n)$).

Our contribution. Our main result is the following theorem:

▶ **Theorem 1.** *There exist a constant α and a polynomially computable function $\mathcal{R}$ mapping CNF formulas to CNF formulas with the following properties. For any 3-CNF φ with n variables such that: if φ is satisfiable, then $\mathcal{R}(\varphi)$ has a resolution refutation of size at most n^α; if φ is unsatisfiable, then any* OBDD($\wedge$, *weakening) refutation of $\mathcal{R}(\varphi)$ has size $2^{\Omega(n)}$.*

Since $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ simulates resolution, any automation algorithm for $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ can be used to solve 3-SAT: if it finds proofs in fixed polynomial time, then the input formula is satisfiable, otherwise, it is unsatisfiable.

Our technique can be applied to other proof systems as well since the only thing that we use about OBDDs is that the value of an OBDD of size S can be computed using $O(\ell \log S)$ bits of communication in the ℓ-party number-in-hand communication model if the partition of variables agrees with the order. For example, this property holds for k-OBDDs for small k, hence our technique can be applied for proof system k-OBDD($\wedge$, weakening) [14].

Technique. The proof consists of two parts:

1. Prove the weaker version of Theorem 1, where the lower bound holds only for refutations that consist of OBDDs in some particular order π.
2. Devise a polynomial-time algorithm that transforms formulas with short resolution refutations to formulas with short resolution refutations; and transforms formulas that are hard for $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ with a specific order to formulas that are hard for $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ for all orders.

To implement the second part we use Segerlind's transformation. It almost suits our case, but the property for resolution works only with an additional condition: if a formula has a short resolution proof with at most constant number negative literals in every clause (we say that the *negative width* of the proof is $O(1)$), then the result of Segerlind's transformation has a short resolution proof.

The first part is much more involved. The construction is built on the following result proved by Atserias and Müller [5]. There exists an algorithm $\mathcal{E}$ that given a 3-CNF formula φ produces in polynomial time another CNF formula $\mathcal{E}(\varphi)$ such that

- if φ is satisfiable, $\mathcal{E}(\varphi)$ admits a polynomial-size resolution refutation;
- if φ is unsatisfiable, the shortest refutation of $\mathcal{E}(\varphi)$ has size $2^{|\varphi|^{\Omega(1)}}$.

We get our result by applying lifting to $\mathcal{E}(\varphi)$. Lifting is a technique to obtain lower bounds for strong computational models from lower bounds for weaker models. Recently, Garg, et. al. [10] proved two similar lifting theorems lifting from resolution width to refutation size in (1) any semantic proof system operating with proof lines of small 2-party communication complexity and (2) cutting planes (precisely it works for proof systems, where proof lines can be computed by 1-round real communication protocol).

The first lifting theorem (applied to $\mathcal{E}(\varphi)$) seems enticing for us since a function computable by an OBDD can be computed with small 2-party communication. Unfortunately, we can not apply this theorem directly since $\mathcal{E}(\varphi)$ can have large resolution width even for a satisfiable φ so after the application of lifting the resulting CNF might have only exponential-size $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ refutations. Göös et. al. [12] face the same problem for Cutting Planes and deal with it by lifting from block-width instead of the plain width. However the lifting theorem in [10] does not work for block-width, so Göös et. al. [12] prove a weaker version of it: they lift from resolution block-width to k-dimensional simplex-dags, where k is the number of variables in the lifted formula plus one. Cutting planes refutations can be converted to k-dimensional simplex-dags of the same size. However, for OBDD($\wedge$, weakening) refutations, the size is raised to the power of k, hence we need a lifting theorem for a smaller value of k.

We prove another lifting theorem: we lift from resolution block-width to k-dimensional box dag size, where k is the size of the largest block in the partition w.r.t. which the block-width is computed, plus one. In our proof, we use the structural properties of rectangles from [10] and extend them to show the structural properties of boxes. The same theorem seems to hold for simplex-dags (the proof in [12] can be adapted as well), but it is not clear whether there exist context where such change in the dimension matters.

We also show that OBDD($\wedge$, weakening) refutations with a specific order of variables of size S can be converted to k-dimensional box dags of size $S^{O(k)}$. In actuality, we prove it for every proof system that operates with proof lines that can be computed by k party communication protocols in the number-in-hand model with a small cost.

2 Preliminaries

Notation. We use the standard notation $[n] = \{1, \dots, n\}$. $\mathrm{Vars}(\varphi)$ denotes the set of propositional variables of a formula φ. We refer to a uniform distribution over a set X by $\mathcal{U}(X)$.

Resolution. A resolution refutation of an unsatisfiable CNF φ is a sequence of clauses ending with the empty clause such that each clause of the sequence is either a clause of φ or is derived from the previous clauses in the sequence with a resolution rule: $\frac{A \vee x \quad B \vee \neg x}{A \vee B}$.

The *width* of a clause is the number of literals in it, and the width of a formula is the maximum width of a clause in it. The *size* of a resolution refutation is the number of clauses in it. The width of a resolution refutation is the largest width of a clause in it.

Let X be a set of propositional variables and $U = U_1, \dots, U_k$ be a partition of X. Let us define the *block-width* of a clause C over variables X as the number of blocks among $U_1, \dots, U_k$ that contain variables of C: $|\{i \in [k] \mid \mathrm{Vars}(C) \cap U_i \neq \emptyset\}|$. The block-width of a resolution refutation is the maximum block-width of a clause in it. For an unsatisfiable CNF φ we denote $\mathrm{bw}(\varphi)$ as the smallest block-width of a resolution refutation of φ.

Ordered Binary Decision Diagrams. A branching program (BP) is a directed acyclic graph with a single source and two sinks: 0-sink and 1-sink. Each of the nodes of the BP except the sinks is labeled with a variable x_i for $i \in [n]$ and has two outgoing edges, one labeled with 1 and another with 0. Let us define the function computed by a BP. For a node u in a BP let $f_u : \{0,1\}^n \to \{0,1\}$ be a function computed by it. We then define $f_{\text{0-sink}} \equiv 0$, $f_{\text{1-sink}} \equiv 1$, $f_u(x) := f_v(x)$ if $x_i = 0$ and $f_u(x) := f_w(x)$ if $x_i = 1$ where u is labeled with the variable x_i, v is 0-successor of u and w is the 1-successor of u. Then we define the function computed by the BP itself as the function computed by its source.

A π-OBDD where $\pi \in S_n$ is a BP computing a function $f : \{0,1\}^n \to \{0,1\}$ such that for any path from the source to a sink each of the node labels appears at most once and the order of the labels appearing in the path respects π. That is, the labels appearing on the path always have form $x_{\pi(i_1)}, x_{\pi(i_2)}, \dots, x_{\pi(i_k)}$ where $1 \le i_1 < i_2 < \dots < i_k \le n$.

OBDDs have the following nice property: for every order of variables every Boolean function has a unique minimal OBDD. For a given order π, the minimal π-OBDD of a function f may be constructed in polynomial time from any π-OBDD for the same function [18]. There are also known polynomial-time algorithms that efficiently perform all the Boolean binary operations, negation and projection (elimination of the existential quantifier) to π-OBDDs [18] (we refer to [26] for an introduction to OBDDs).

OBDD refutations. A π-OBDD-refutation of a CNF formula φ is a sequence of π-OBDDs $D_1, \dots, D_s$ such that D_s computes the identically false function and each D_i either computes a clause of φ or is obtained from the previous diagrams in the sequence by one of the rules below.

conjunction rule ($\wedge$) D_i computes the conjunction of D_j and D_k for $j, k < i$;

weakening rule D_i computes a function implied by D_j where $j < i$;

projection rule ($\exists$) D_i computes a function $\exists x f$ where f is computed by D_j with $j < i$, and $x \in \mathrm{Vars}(\varphi)$.

The size of an π-OBDD-refutation is the sum of sizes of all diagrams in it. Using the properties of OBDD it is easy to see that the correctness of a π-OBDD-refutation can be verified in time polynomial in its size and the size of the refuted formula [4]. An OBDD refutation is a π-OBDD refutation for some order π.

Depending on the set of the allowed rules we have several different propositional proof systems: $\mathrm{OBDD}(\wedge)$ where only the conjunction rule is allowed, $\mathrm{OBDD}(\wedge, \exists)$ where the conjunction and the projection rules are allowed, and $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ where the conjunction and the weakening rules are allowed. Since the projection rule is a special case of the weakening rule, we do not include both of them simultaneously.

For an unsatisfiable CNF φ we denote by $\pi\text{-}\mathrm{OBDD}(\varphi)$ the size of the smallest π-$\mathrm{OBDD}(\wedge, \mathrm{weakening})$ refutation of φ and by $\mathrm{OBDD}(\varphi)$ the size of the smallest $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ refutation of φ.

▶ **Proposition 2** ([4]). $\mathrm{OBDD}(\wedge, \exists)$ *(and, thus,* $\mathrm{OBDD}(\wedge, \mathrm{weakening})$*) polynomially simulates resolution: if an unsatisfiable CNF has a resolution refutation of size S, then it has an* $\mathrm{OBDD}(\wedge, \exists)$ *refutation of size* $\mathrm{poly}(S)$*.*

Search$_\varphi$. Search_φ is the following search problem: given an assignment to the variables of the unsatisfiable CNF φ, find a clause that is falsified by this assignment. Formally it can be defined as a relation $\{(x, C) \mid x \in \{0,1\}^{\mathrm{Vars}(\varphi)};\ C \in \varphi;\ C(x) = 0\}$.

Dags solving relations.

▶ **Definition 3** ([25]). *Let $\mathcal{F}$ be a family of subsets of a finite set $\mathcal{X}$ and $S \subseteq \mathcal{X} \times \mathcal{O}$ be a relation. Let $\mathcal{D}$ be a single-source (which we refer to as root) acyclic graph. We call $\mathcal{D}$ an* $\mathcal{F}$-dag *solving S if for every node u there exists a set $R_u \in \mathcal{F}$ such that:*

(root condition) *for the root r of the dag $R_r = \mathcal{X}$;*

(leaf condition) *for each leaf (sink) ℓ of the dag there exists $o \in \mathcal{O}$ such that for all $x \in R_\ell$, $(x, o) \in S$;*

(local condition) *each inner node u has out-degree 2 and its two descendants v and w satisfy the property $R_u \subseteq R_v \cup R_w$.*

The size of an $\mathcal{F}$-dag *is the number of nodes in it. We denote the smallest size of* $\mathcal{F}$-dag *solving S by* $\mathcal{F}$-dag(S)*. We usually identify the nodes of an* $\mathcal{F}$-dag *with the sets R_u.*

Now we define several special cases of this general definition.

Decision dag. Assume that we have Boolean domain $\mathcal{X} = \{0,1\}^n$ that we view as a set of values of n propositional variables. A partial assignment is an element of $\{0,1,*\}^n$, where $*$ means that the corresponding variable is not assigned. Let $\mathrm{fix}(\rho) = \rho^{-1}(\{0,1\})$ be the set of assigned variables. If $\mathrm{fix}(\rho) = [n]$ then ρ is a full assignment.

Any partial assignment defines a subcube $\mathrm{Cube}(\rho) = \{\alpha \in \{0,1\}^n \mid \forall i \in \mathrm{fix}(\rho) : \rho(i) = \alpha(i)\}$ that is the set of all full assignments agreeing with ρ.

Let $S \subseteq \{0,1\}^n \times \mathcal{O}$ be a relation and $\mathcal{F}$ be a set of all subcubes $\{\mathrm{Cube}(\rho) \mid \rho \in \{0,1,*\}^n\}$, then we call an $\mathcal{F}$-dag for S a decision dag. We denote the smallest size of a decision dag solving S by $\mathsf{dec\text{-}dag}(S)$.

Observe that a decision tree is a decision dag: a node u of a decision tree can be labeled with a set $\mathrm{Cube}(\rho)$, where ρ is a partial assignment corresponding to the path from the root to u. Hence, since for any total relation there exists a decision tree solving it, any total relation has a decision dag as well.

Let $U = U_1, \dots, U_k$ be a partition of $[n]$. The block-width of a decision dag is defined as follows: for a node labeled with $\mathrm{Cube}(\rho)$ we compute $|\{i \in [k] \mid U_i \cap \mathsf{fix}(\rho) \neq \emptyset\}|$, the blockwidth of a decision dag is the maximum of this value among the nodes. For a relation S we denote the smallest block-width of a decision dag that solves it as $\mathrm{bw}(S)$.

Observe that a resolution refutation of an unsatisfiable CNF φ can be converted to a decision dag solving Search_φ of the same size: the topology of the dag is the topology of the resolution refutation, a node corresponding to a clause C is labeled with a set $C^{-1}(0) = \{x \in \{0,1\}^n \mid C(x) = 0\}$. It is easy to see that this set is a subcube. If C is derived from D and E via a resolution rule then C is implied by the conjunction of D and E thus $C^{-1}(0) \subseteq (D \wedge E)^{-1}(0) = D^{-1}(0) \cup E^{-1}(0)$. Clearly the root and the leaf properties of the constructed decision dag also hold: for a leaf ℓ labeled with $C^{-1}(0)$ for $C \in \varphi$ every point in $C^{-1}(0)$ falsifies φ by definition; the root corresponds to the empty clause so it is labeled with $\{0,1\}^n$. The reverse also holds, one can convert a decision dag solving Search_φ to a resolution refutation of φ of the same size.

▶ **Proposition 4** ([10]). *There exists a resolution refutation of φ of size S and block-width b if and only if there exists a decision dag solving* Search_φ *of size S and block-width b.*

Box dag. Let $S \subseteq \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_k \times \mathcal{O}$ be a relation. Let $\mathcal{F}$ be a set of boxes $\{A_1 \times A_2 \times \dots \times A_k \mid A_1 \subseteq \mathcal{X}_1, A_2 \subseteq \mathcal{X}_2, \dots, A_k \subseteq \mathcal{X}_k\}$. Then we call an $\mathcal{F}$-dag a box dag. Let $U = U_1, \dots, U_k$ be a partition of $[n]$. If $\mathcal{X}_i = \{0,1\}^{U_i}$ for all $i \in [k]$, then we denote the class of box dags as $\mathsf{box\text{-}dag}_U$ or $\mathsf{box\text{-}dag}_{U_1,\dots,U_k}$.

▶ Remark 5. We can convert a π-OBDD refutation of a formula φ of size S to an $\mathcal{F}$-dag for Search_φ, where $\mathcal{F}$ consists of zero-sets of π-OBDDs of size at most S. In Section 5 we show that if a partition of variables into k parts agrees with an order π, such a dag can be converted to a box dag of size $S^{O(k)}$.

Automatability. A propositional proof system Π is called *automatable* if there exists an algorithm $\mathcal{A}_\Pi$ that given an unsatisfiable CNF φ produces its refutation in Π in time polynomial in $|\varphi|$ and the size of the shortest refutation of φ in Π.

3 The outline of the proof of Theorem 1

Our starting point is the following theorem that is essentially proved in [13].

▶ **Theorem 6** (Lemma 2.2 from [13]). *For any constant $c \geq 2$ there exists a polynomial-time algorithm $\mathcal{E}$ such that given a 3-CNF formula φ of size n it produces a $O(\log n)$-CNF formula $\mathcal{E}(\varphi)$ such that*

- *there exists a partition $B_1, \dots, B_k$ of the variables of $\mathcal{E}(\varphi)$ such that $|B_1| = |B_2| = \dots = |B_k| = O(n)$ and $k = O(n^{c+1})$ and this partition can be computed in polynomial time;*
- *if $\varphi \in$ SAT then $\mathcal{E}(\varphi)$ has a resolution refutation π such that $|\pi| = n^{O(c)}$ and* $\mathrm{bw}(\pi) = O(1)$ *w.r.t. partition $B_1, \dots, B_k$;*
- *if $\varphi \notin$ SAT then any resolution refutation of $\mathcal{E}(\varphi)$ has block-width at least n^{c-1} w.r.t. $B_1, \dots, B_k$.*

Notice that the statement of Theorem 6 is slightly different from one explicitly stated in [13]. First, it is not stated that all blocks B_i have equal sizes and their sizes are $O(n)$, but this is clear from the definition in Section 3.1 of [13]. Second, the theorem is stated and proved only for $c = 2$ but essentially the same proof holds for larger c, the only change is that we should reduce from rPHP_{n^c} instead of rPHP_{n^2} (see Section 5 of [13] for details).

To prove Theorem 1 we follow the plan below:

Lifting with multi-output indexing function. In Section 4 we define a block-wise indexing function $\textsc{Ind}_{\ell\times m}$ and its composition with relations and formulas. In Section 4 we will see that if a CNF formula φ has short resolution refutation of constant block-width then $\varphi \circ \textsc{Ind}^n_{\ell\times m}$ has a short resolution refutation. In the remainder of Section 4 we show that if a CNF formula φ with variables partitioned into n blocks of size ℓ requires resolution refutations of block-width at least b, then $\mathrm{Search}_\varphi \circ \textsc{Ind}^n_{\ell\times m}$ and consequently $\mathrm{Search}_{\varphi\circ\textsc{Ind}^n_{\ell\times m}}$ requires large $(\ell+1)$-dimensional box dags.

Making box dags out of π-OBDD refutations. In Section 5 we show that if Search_φ requires k-dimensional box dags of size S, then it requires π-OBDD$(\wedge, \text{weakening})$ refutations of size $S^{\Omega(1/k)}$ for some fixed π.

Making all orders hard. In Section 6 we adapt Segerlind's transformation from [24] to show that there exists a CNF-to-CNF mapping that maps CNF formulas with polynomial resolution size to CNF formulas with polynomial resolution size and maps CNF formulas that are hard for π-OBDD$(\wedge, \text{weakening})$ with a fixed π to CNF formulas that are hard for OBDD$(\wedge, \text{weakening})$.

Putting the pieces together. In Section 7 we compose $\mathcal{E}_c$ with the two mappings above to obtain Theorem 1.

4 Lifting with multi-output indexing function

In this section, we prove the lifting theorem for box dags. First, let us formally define the gadget we are going to lift with.

▶ **Definition 7** (Block-wise indexing, [12]). *$\textsc{Ind}_{\ell\times m} : [m] \times \{0,1\}^{\ell\times m} \to \{0,1\}^\ell$ is defined as $\textsc{Ind}_{\ell\times m}(x, y) = (y_{1,x}, y_{2,x}, \dots, y_{\ell,x})$ i.e. given an index $x \in [m]$ and a matrix $y \in \{0,1\}^{\ell\times m}$, it returns the xth column of y. For a set $R \subseteq [m]^n \times (\{0,1\}^{\ell\times m})^n$ we define $\textsc{Ind}^n_{\ell\times m}(R) = \{(\textsc{Ind}_{\ell\times m}(x_1, y_1), \dots, \textsc{Ind}_{\ell\times m}(x_n, y_n)) \in \{0,1\}^{n\ell} \mid (x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n) \in R\}$.*

Let $\varphi = \bigwedge_{i=1}^t C_i$ be an unsatisfiable CNF with $n\ell$ variables that are partitioned into n blocks of size ℓ. Let us define a CNF $\psi = \varphi \circ \textsc{Ind}^n_{\ell\times m}$. First let us define $C \circ \textsc{Ind}^n_{\ell\times m}$ for a clause C. The resulting CNF formula will compute the function $C \circ \textsc{Ind}^n_{\ell\times m} = C(\textsc{Ind}_{\ell\times m}(x_1, y_1), \dots, \textsc{Ind}_{\ell\times m}(x_n, y_n))$. Then we define $\varphi\circ\textsc{Ind}^n_{\ell\times m} := \bigwedge_{i=1}^t \left(C_i \circ \textsc{Ind}^n_{\ell\times m}\right)$.

Now let us construct a CNF representation of $C \circ \textsc{Ind}^n_{\ell\times m}$. Let $z_{i,j}$ for $i \in [n]$, $t \in [\ell]$ be the tth variable of the ith block of φ. Let $i_1, \dots, i_b \in [n]$ be indices of the blocks that are touched by C and let C_j for $j \in [b]$ be the part of the variables of C from the i_jth block: $C = C_1 \vee \dots \vee C_b$. Let $P_j := \{k \in [\ell] \mid z_{i_j,k} \in C\}$ be the indices (inside a block) of positive literals in C_j and $N_j := \{k \in [\ell] \mid \neg z_{i_j,k} \in C\}$ be the indices of negative literals in C_j. Then the CNF representation of $C \circ \textsc{Ind}^n_{\ell\times m}(x_1, y_1, \dots, x_n, y_n)$ consists of clauses of form $\left(\left(\bigwedge_{j=1}^b (x_{i_j} = \alpha_j)\right) \to \left(\bigvee_{j=1}^b \left(\bigvee_{k\in P_j} y_{k,\alpha_j} \vee \bigvee_{k\in N_j} \neg y_{k,\alpha_j}\right)\right)\right)$ for each $\alpha_1, \dots, \alpha_b \in [m]$.

The size of this representation is $|\varphi| \cdot m^b$ where b is the largest block-width of a clause in φ, so this representation is short for formulas of constant block-width.

▶ **Theorem 8** (the last inequality in Theorem 4 from [12]). *Let φ be an unsatisfiable CNF with $n\ell$ variables that are partitioned into n blocks of size ℓ such that there exists a resolution refutation of φ of size s and block-width b. Then there exists a resolution refutation of $\varphi \circ \textsc{Ind}^n_{\ell\times m}$ of size $m^{O(b)} \cdot s$.*

4.1 Lifting theorem

For a relation $S \subseteq (\{0,1\}^\ell)^n \times \mathcal{O}$ its composition with block-wise indexing is defined as
$$S \circ \textsc{Ind}^n_{\ell\times m} := \left\{ (x_1,\dots,x_n,y_1,\dots,y_n,o) \;\middle|\; \begin{array}{l} x_i \in [m];\ y_i \in \{0,1\}^{\ell\times m};\ o \in \mathcal{O}; \\ (\textsc{Ind}_{\ell\times m}(x_1,y_1),\dots,\textsc{Ind}_{\ell\times m}(x_n,y_n),o) \in S \end{array} \right\}.$$

This is a direct analog of the composition of two functions: we first plug the output of indexing to each ℓ-bit block of the input of S and then “compute” S on the resulting input.

We assume that m is a power of 2 so the relation $S \circ \textsc{Ind}^n_{\ell\times m}$ can be viewed as defined on a binary domain $\{0,1\}^{n\log_2 m + \ell n m}$.

Let us define a partition of the input bits of relation $S \circ \textsc{Ind}^n_{\ell\times m}$. Consider an element of the input domain $(x_1,\dots,x_n,y_1,\dots,y_n) \in [m]^n \times \left(\{0,1\}^{\ell\times m}\right)^n$ where $x_1,\dots,x_n \in [m]$ and $y_1,\dots,y_n$ are matrices in $\{0,1\}^{\ell\times m}$. Let A consist of bits corresponding to of $x_1,\dots,x_n$, (in other words A corresponds to the first $n\log_2 m$ bits of the input), B_j for $j \in [\ell]$ consists of bits corresponding to jth rows of all the matrices $y_1,\dots,y_n$. We are going to imagine that we have $\ell+1$ parties: Alice who receives the bits A of the input, Bob$_1$, Bob$_2$, …, Bob$_\ell$, where Bob$_j$ receives the bits B_j of the input.

Then let $\mathcal{A} := \{0,1\}^A = [m]^n$ be the set of Alice's inputs and let $\mathcal{B}_j := \{0,1\}^{B_j} = \{0,1\}^m$ be the set of Bob$_j$'s inputs.

The following theorem is similar with Theorem 8 form [12], but for box dags instead of simplex dags and, crucially, for a smaller number of parties, $\ell+1$ instead of $n\ell+1$.

▶ **Theorem 9.** *Let Δ be a large enough integer constant. Let $S \subseteq (\{0,1\}^\ell)^n \times \mathcal{O}$ be a total relation where $\ell < \frac{n}{2}$ and $m = (n\ell)^\Delta$. Then $m^{\Omega(\mathrm{bw}(S))} \le \textsf{box-dag}_{A,B_1,\dots,B_\ell}(S \circ \textsc{Ind}^n_{\ell\times m})$, where the block partition of inputs of S is the natural partition into n blocks of size ℓ.*

Let us outline the proof of Theorem 9. The proof is constructive, i.e., we take a box dag $\mathbb{B}$ solving $S \circ \textsc{Ind}^n_{\ell\times m}$ and extract from it a decision dag solving S of block-width $O(\log|\mathbb{B}|/\log m)$. The idea is to split boxes in the box dag into “structured” boxes that naturally correspond to partial assignments from $\{0,1,*\}^n$ (notice that there is a one-to-one correspondence between partial assignments and subcubes). We then take the assignments that our structured boxes correspond to and construct a decision dag for S out of them (we will need some auxiliary partial assignments as well). A first attempt to formulate what this “structuredness” could mean is the following: a box B is ρ-like if $\textsc{Ind}^n_{\ell\times m}(B) = \mathrm{Cube}(\rho)$. It turns out that we actually can (with some caveats) partition any box in $\mathcal{A} \times \mathcal{B}_1 \times \cdots \times \mathcal{B}_\ell$ into boxes that are ρ-like for some assignments ρ. Unfortunately, we need some additional properties of these boxes to be able to connect them into a valid decision dag.

Our definition of structured boxes is different from the one given in [12], we formulate it in a different way reducing the structuredness of boxes to the structuredness of rectangles (2-dimensional boxes) that is used to prove the lifting theorem in [10]. In Subsection 4.2 we formulate the properties of structured rectangles that we need, in Subsection 4.3 we define and prove the analogous properties for structured boxes, and in Subsection 4.4 we construct the decision dag solving S.

4.2 Structured Rectangles

Lifting theorems from [10] rely heavily on the notion of *structuredness* of rectangles. To simplify things we will not define it explicitly, but instead, state its properties that we are going to use.

Let $\mathsf{Rect}_{m,n}$ be the set of subrectangles of $[m]^n \times \left(\{0,1\}^{1\times m}\right)^n$: $\{A \times B \mid A \subseteq [m]^n;\ B \subseteq \left(\{0,1\}^{1\times m}\right)^n\}$. We are going to define several properties of predicates on $\mathsf{Rect}_{m,n} \times \{0,1,*\}^n$ i.e. predicates on pairs of form (rectangle, partial assignment). Let $\mathcal{W}$ be a predicate on $\mathsf{Rect}_{m,n} \times \{0,1,*\}^n$.

▶ **Definition 10.** *We say that $\mathcal{W}$* observes row-structure *if $\mathcal{W}(X \times Y, \rho)$ implies that for all $x \in X$, $\textsc{Ind}^n_{1\times m}(\{x\} \times Y) \subseteq \mathrm{Cube}(\rho)$, and $\Pr_{x\leftarrow\mathcal{U}(X)}\left[\textsc{Ind}^n_{1\times m}(\{x\} \times Y) \neq \mathrm{Cube}(\rho)\right] \leq \frac{2}{n}$.*

▶ **Definition 11.** *We say that $\mathcal{W}$ is* partitionable *if for every $X \subseteq [m]^n$ there exist a partition $X := \bigsqcup_{j\in J} \tilde{X}_j$ and a family $\{F_j\}_{j\in J}$, $F_j \subseteq [n]$, and for every $R = X \times Y \in \mathsf{Rect}_{m,n}$, for every parameter $k \leq n \log n$ there exists a partition $R = \bigsqcup_{i\in I} R_i$, where $R_i = X_i \times Y_i \in \mathsf{Rect}_{m,n}$, a family of assignments $\{\rho_i\}_{i\in I}$, and sets $X_{err} \subseteq X, Y_{err} \subseteq Y$ such that $|X_{err}| \leq m^n/2^k$, $|Y_{err}| \leq 2^{mn-k}$ and the following properties hold:*

1. *for each i one of the following holds: either $\mathcal{W}(R_i, \rho_i)$ and $|\mathrm{fix}(\rho_i)| = O(k/\log n)$; or R_i is covered by $X_{err} \times \left(\{0,1\}^{1\times m}\right)^n \cup [m]^n \times Y_{err}$.*
2. *For every $i \in I$ there exists $j \in J$ such that $\tilde{X}_j = X_i$ and $\mathrm{fix}(\rho_i) = F_j$* [1]*.*

▶ **Definition 12.** *We say that $\mathcal{W}$* respects largeness *if for all $X \times Y$ such that $|X| \geq m^n \cdot 0.99$ and $|Y| \geq 2^{mn} \cdot 0.99$ $\mathcal{W}(X \times Y, *^n)$ holds.*

▶ **Theorem 13** (Lemma 4.4, Lemma 4.5 from [10])**.** *There exists a constant Δ such that for any $m \geq n^{\Delta}$ there exists a predicate $\mathcal{W}$ on $\mathsf{Rect}_{m,n} \times \{0,1,*\}^n$ such that it* observes row-structure[2]*;* is partitionable*;* respects largeness[3]*. We say that a rectangle R is ρ-*structured *iff $\mathcal{W}(R, \rho)$ holds.*

4.3 Structured Boxes

Now let us generalize the notion of structuredness from rectangles to boxes.

▶ **Definition 14.** *Let $R = X \times Y_1 \times \cdots \times Y_\ell$, where $X \subseteq \mathcal{A} = [m]^n$, $Y_j \subseteq \mathcal{B}_j = (\{0,1\}^{1\times m})^n$ be a box and $\rho \in \{0,1,*\}^{n\ell}$ be a partial assignment. We view ρ as an assignment to variables of input to $S \subseteq (\{0,1\}^\ell)^n \times \mathcal{O}$ that are partitioned into n blocks of size ℓ. Let $\rho_i \in \{0,1,*\}^n$ for $i \in [\ell]$ be the marginal assignment of ρ assigning the ith variable of each block in the partition of variables of S. We say that R is a ρ-*structured box *if for each $i \in [\ell]$ the rectangle $X \times Y_i$ is ρ_i-structured.*

We now show that our definition of the structuredness satisfies the analogues of conditions from Definitions 10, 11, and 12.

▶ **Lemma 15.** *Assume that $n > 2\ell$. Let $R = X \times Y_1 \times \cdots \times Y_\ell \subseteq \mathcal{A} \times \mathcal{B}_1 \times \cdots \times \mathcal{B}_\ell$ be a ρ-structured box where $\rho \in \{0,1,*\}^{n\ell}$. Then for all $x \in X$, $\textsc{Ind}^n_{\ell\times m}(\{x\} \times Y_1 \times \cdots \times Y_\ell) \subseteq \mathrm{Cube}(\rho)$ and there exists $x \in X$ such that $\textsc{Ind}^n_{\ell\times m}(\{x\} \times Y_1 \times \cdots \times Y_\ell) = \mathrm{Cube}(\rho)$.*

[1] This property is not explicitly stated in [10], although it is clear from the Rectangle Scheme that generates the partition: first X is partitioned and then each part $X_i \times Y$ is partitioned separately.

[2] Although Lemma 4.4 of [10] is not stated in strong enough form to satisfy Definition 10, the needed property is actually proved in Section 9 of [10].

[3] This property is implicit in [10], see the full version of the paper for the details.

Proof. If there exist $x \in X, y_1 \in Y_1, \dots, y_\ell \in Y_\ell$ such that $\alpha := \textsc{Ind}^n_{\ell\times m}(x, y_1, \dots, y_\ell)$ does not agree with ρ, then there exists $i \in [\ell]$ such that $\textsc{Ind}^n_{1\times m}(x, y_i)$ does not agree with ρ_i which violates Definition 10.

Now let us prove the second statement. By Definition 10 for each $i \in [\ell]$ we have $\Pr_{x\leftarrow\mathcal{U}(X)}\left[\textsc{Ind}^n_{1\times m}(\{x\}\times Y_i) \neq \mathrm{Cube}(\rho_i)\right] \leq \frac{2}{n}$. Then $\Pr_{x\leftarrow\mathcal{U}(X)}\left[\textsc{Ind}^n_{\ell\times m}(\{x\}\times Y_1 \times Y_2 \times \cdots \times Y_\ell) \neq \mathrm{Cube}(\rho)\right]$ is bounded by $\sum_{i=1}^{\ell} \Pr_{x\leftarrow\mathcal{U}(X)}\left[\textsc{Ind}^n_{1\times m}(\{x\}\times Y_i) \neq \mathrm{Cube}(\rho_i)\right] \leq \frac{2\ell}{n} < 1$. ◀

▶ **Lemma 16.** *If $R = X \times Y_1 \times \cdots \times Y_\ell \subseteq \mathcal{A} \times \mathcal{B}_1 \times \cdots \times \mathcal{B}_\ell$ is such that $|X| \geq m^n \cdot 0.99$ and $|Y_i| \geq 2^{mn} \cdot 0.99$ for each $i \in [\ell]$, then R is $*^{n\ell}$-structured.*

Proof. By Definition 12 we have that each of the $X \times Y_i$ is $*^n$-structured which by definition implies $*^{n\ell}$-structuredness of R. ◀

▶ **Lemma 17.** *Let $R = X \times Y_1 \times \cdots \times Y_\ell \subseteq \mathcal{A} \times \mathcal{B}_1 \times \cdots \times \mathcal{B}_\ell$ be an arbitrary box and $k \leq n \log n$ be a parameter. Then there exist sets $X^{err} \subseteq \mathcal{A}, Y_1^{err} \subseteq \mathcal{B}_1, \dots, Y_\ell^{err} \subseteq \mathcal{B}_\ell$, a partition $R = \bigsqcup_{i\in I} R_i$, and a family of partial assignments $\{\rho^i\}_{i\in I}$, where $R_i = X^i \times Y_1^i \times \cdots \times Y_\ell^i$ is a box and $\rho^i \subseteq \{0, 1, *\}^{n\ell}$ satisfying the following conditions.*

(1*) $|X^{err}| \leq \frac{m^n\cdot\ell}{2^k}$, $|Y_i^{err}| \leq 2^{nm-k}$.

(2*) *For each $i \in I$ at least one of the following statements holds:*
- *R_i is ρ^i-structured and ρ^i assigns $O(k/\log n)$ blocks from the standard partition of $[n\ell]$ into n blocks of size ℓ;*
- *R_i is covered by one of the error sets i.e. $X^i \subseteq X^{err}$ or there exists $j \in [\ell]$ such that $Y_j^i \subseteq Y_j^{err}$.*

(3*) *For each $x \in X \setminus X^{err}$ there exists a set $I_x \subseteq [n\ell]$ that is a union of $O(k/\log n)$ blocks (i.e. it either contains all the indices from a block or none) such that $x \in X^i$ implies $\mathrm{fix}(\rho^i) \subseteq I_x$.*

4.4 Proof of Theorem 9

Recall that the inequality we are to prove is $m^{\Omega(\mathrm{bw}(S))} \leq \mathsf{box\text{-}dag}_{A,B_1,\dots,B_\ell}(S \circ \textsc{Ind}^n_{\ell\times m})$. It is equivalent to $\mathrm{bw}(S) = O\left(\log \mathsf{box\text{-}dag}_{A,B_1,\dots,B_\ell}(S \circ \textsc{Ind}^n_{\ell\times m})/\log m\right)$.

Consider the smallest $\mathsf{box\text{-}dag}_{A,B_1,\dots,B_\ell}$ $\mathbb{B}$ solving $S \circ \textsc{Ind}^n_{\ell\times m}$. We construct a decision dag solving S of block-width $O(\log|\mathbb{B}|/\log m) = O(\log|\mathbb{B}|/\log n)$.

Similarly to [10] we first assume that partitions yielded by Lemma 17 are always errorless, i.e. $X^{err} = Y_1^{err} = \cdots = Y_\ell^{err} = \emptyset$. Then we will fix the proof so it works without this assumption, this part of the proof repeats the argument from Section 5.3 in [10] more or less verbatim, so we omit it in this version of the paper. We apply Lemma 17 to each of the boxes in $\mathbb{B}$ with some parameter k that we fix later to achieve the needed lower bound.

Let us construct a decision dag $\mathbb{D}$ that solves S. Each node of a decision dag labeled with function f naturally corresponds to a partial assignment ρ_f such that $\mathrm{Cube}(\rho_f) = f^{-1}(0)$. We will identify nodes of a decision dag with the assignments corresponding to them. That suggests the construction of $\mathbb{D}$: for each of the nodes of $\mathbb{B}$ we apply Lemma 17 to it and for each ρ-structured box in the resulting partition add the node ρ to $\mathbb{D}$. To turn this collection of nodes into a correct decision dag, we need to locate the root, the leaves, and connect (via auxiliary nodes) the nodes between each other such that the conditions on dags are met. More precisely, it is sufficient to show that:

1. The partition of the root of $\mathbb{B}$ consists of a single $*^{n\ell}$-structured box.
2. If an o-labeled leaf of $\mathbb{B}$ contains a ρ-structured box in its partition, then for every $x \in \mathrm{Cube}(\rho)$, $(x, o) \in S$.
3. Suppose a node u in $\mathbb{B}$ has direct descendants v_1 and v_2. Then let $\rho^u_1, \dots, \rho^u_{t_u}$ be the assignments yielded by the partition of the box u, $\rho^{v_q}_1, \dots, \rho^{v_q}_{t_{v_q}}$ be the assignments yielded by the partition of the box v_q for $q \in \{1, 2\}$. Then there exists a assignment-labeled dag with sources $\rho^u_1, \dots, \rho^u_{t_u}$, leaves $\rho^{v_q}_1, \dots, \rho^{v_q}_{t_{v_q}}$ for $q \in \{1, 2\}$ that satisfies the local condition of a decision dag having block-width $O(k/\log n)$.

Proof of 1. By Lemma 16 we have that the entire root of $\mathbb{B}$ is $*^{n\ell}$-structured, thus we may assume that its partition is a single box.

Proof of 2. Let u be an o-labeled leaf of $\mathbb{B}$. Suppose that $B = X \times Y_1 \times \cdots \times Y_\ell$ is a ρ-structured box in the partition of u. By Lemma 15 there exists x_0 such that $\mathrm{I}\textsc{nd}^n_{\ell\times m}(\{x_0\} \times Y_1 \times \cdots \times Y_\ell) = \mathrm{Cube}(\rho)$, i.e. for every $\alpha \in \mathrm{Cube}(\rho)$ there exist $y_1, \dots, y_\ell$ such that $(x_0, y_1, \dots, y_\ell) \in B$ and $\mathrm{I}\textsc{nd}^n_{\ell\times m}(x_0, y_1, \dots, y_\ell) = \alpha$. Then since $\mathbb{B}$ is a box-dag for $S \circ \mathrm{I}\textsc{nd}^n_{\ell\times m}$, $(\alpha, o) \in S$.

Proof of 3. It is sufficient to construct a separate dag with local property rooted in ρ^u_i with leaves from $\mathcal{L} := \{\rho^{v_q}_p\}_{q\in\{1,2\},\, p\in[t_{v_q}]}$ of block-width $O(k/\log n)$.

Recall that we abuse notation by identifying nodes of a box dag with their underlying boxes. Let $B = X \times Y_1 \times \cdots \times Y_\ell$ be a ρ^u_i-structured box from the partition of u. And let $x \in X$ be such that $\mathrm{I}\textsc{nd}^n_{\ell\times m}(\{x\} \times Y_1 \times \cdots \times Y_\ell) = \mathrm{Cube}(\rho^u_i)$. By the property of a box-dag, B is covered by the union of boxes v_1 and v_2. Thus $\{x\} \times Y_1 \times \cdots \times Y_\ell$ is also covered by $v_1 \cup v_2$. Let $I^{v_1}_x, I^{v_2}_x \subseteq [n\ell]$ be the variable sets from Lemma 17. Let our ρ^u_i-rooted decision dag consist of two parts. The first part is a decision tree querying one by one all variables from $I^{v_1}_x \cup I^{v_2}_x \setminus \mathrm{fix}(\rho^u_i)$. From each leaf of this decision tree we direct both edges to one of the nodes of $\mathcal{L}$. Observe that by the part (3) of Lemma 17, $I^{v_1}_x$ and $I^{v_2}_x$ are unions of $O(k/\log n)$ blocks and $\mathrm{fix}(\rho^u_i)$ touches $O(k/\log n)$ blocks. Thus block-width of the resulting dag is also $O(k/\log n)$.Consider any leaf of the decision tree $\theta \in \{0, 1, *\}^{n\ell}$. Since $\mathrm{I}\textsc{nd}^n_{\ell\times m}(\{x\} \times Y_1 \times \cdots \times Y_\ell) = \mathrm{Cube}(\rho^u_i)$ and θ extends ρ^u_i (i.e., $\mathrm{Cube}(\theta) \subseteq \mathrm{Cube}(\rho^u_i)$), there exist $y_1 \in Y_1, \dots, y_\ell \in Y_\ell$ such that $\mathrm{I}\textsc{nd}^n_{\ell\times m}(x, y_1, \dots, y_\ell) \in \mathrm{Cube}(\theta)$. Then consider an ω-structured box B_0 from a partition of v_1 or v_2 for $\omega \in \mathcal{L}$ that contains $(x, y_1, \dots, y_\ell)$. Observe that $\mathrm{fix}(\omega) \subseteq I^{v_1}_x \cup I^{v_2}_x \subseteq \mathrm{fix}(\theta)$. The first inclusion holds by the part (3) of Lemma 17, the second holds by the construction of the decision tree. Since by Lemma 15, $\mathrm{I}\textsc{nd}^n_{\ell\times m}(x, y_1, \dots, y_\ell) \in \mathrm{Cube}(\omega)$, $\mathrm{Cube}(\omega)$ and $\mathrm{Cube}(\theta)$ have a point in common, then $\mathrm{fix}(\omega) \subseteq \mathrm{fix}(\theta)$ implies $\mathrm{Cube}(\omega) \supseteq \mathrm{Cube}(\theta)$. Then we can direct both edges from ω to θ. That finishes the proof under the errorless assumption.

5 From Box-Dags to OBDD Refutations

▶ **Lemma 18** (a generalization of a similar lemma in [25]). *Let $U_1, \dots, U_k$ be a partition of $[n]$. Let $\mathcal{F}$ be the class of* functions *that are computable by k-party number-in-hand communication protocol*[4] *of cost c w.r.t. partition $U_1, \dots, U_k$ of $[n]$. Let $S \subseteq \{0,1\}^{U_1} \times \cdots \times \{0,1\}^{U_k} \times Y$ be a relation and let D be a $\mathcal{F}$-dag that solves it. Then there exists a* $\mathsf{box\text{-}dag}_{U_1,\dots,U_k}$ *D' of size $O(|D| \cdot 2^{3c})$ that solves S.*

[4] For a formal definition of number-in-hand protocol see e.g. [21].

Let X be a set of propositional variables of size n, $\mathcal{V} := (V_1, \dots, V_k)$ be a partition of X: $X = V_1 \sqcup \dots \sqcup V_k$, and $\pi : [n] \to X$ be a bijection (order on the variables X). We say that a partition $\mathcal{V}$ *agrees with* π if V_1 comes first in the order, then goes V_2 and so on until V_k.

▶ **Theorem 19.** *Let φ be an unsatisfiable CNF over variables X. Let $\pi : [n] \to X$ be an order of variables and $\mathcal{V}$ be a partition of X agreeing with π. Let $D_1, \dots, D_t$ be a π-OBDD($\wedge$, weakening) refutation of φ of size S. Then $\textsf{box-dag}_{\mathcal{V}}(\mathrm{Search}_{\varphi}) \le S^{O(k)}$.*

▶ **Lemma 20.** *Let D be a π-OBDD over variables X computing a function f and $\mathcal{V} = (V_1, \dots, V_k)$ be a partition of X that agrees with π. Then there exists a k-party number-in-hand communication protocol computing f with cost $k\lceil \log_2 |D| \rceil$.*

Proof of Theorem 19. By Lemma 20, a π-OBDD refutation of φ of size $S = \sum_{i=1}^{t} |D_i|$ can be viewed as an $\mathcal{F}$-dag solving $\mathrm{Search}_{\varphi}$ (for the diagrams derived via the weakening rule we direct both of the outgoing edges to the same node), where $\mathcal{F}$ is the class of functions that can be computed with cost at most $k\lceil \log_2 S \rceil$ by a k-party number-in-hand communication protocol with input partition $\mathcal{V}$. Then by Lemma 18, there exists a $\textsf{box-dag}_{\mathcal{V}}$ of size $S \cdot 2^{3k \log S} = S^{O(k)}$ solving $\mathrm{Search}_{\varphi}$. ◀

6 Making all orders hard

Let *negative width* of a resolution refutation be the maximal number of negative literals in a clause of the refutation.

▶ **Theorem 21** ([24]). *There exists a polynomial-time algorithm $\mathcal{T}_0$ that given a CNF φ over n variables returns a CNF-formula $\mathcal{T}_0(\varphi)$ such that*

- *for any variable ordering π, π-OBDD$(\varphi) \le$ OBDD$(\mathcal{T}_0(\varphi))$ (Lemma 14 from [24]);*
- *If φ has a resolution refutation of size s and* negative width *w, then $\mathcal{T}_0(\varphi)$ has resolution size at most $s \cdot n^{O(w)}$, (Corollary 9 and Lemma 12 from [24]).*

▶ **Lemma 22.** *If a CNF-formula φ has a resolution refutation of size s and the size of the smallest π-OBDD refutation of φ is t, then there exists polynomial-time algorithm that given φ outputs a formula φ' and a variable order π' such that φ' has a resolution refutation of size $O(s)$ and negative width $O(1)$, and the size of the smallest π'-OBDD refutation of φ' is at least t.*

▶ **Corollary 23.** *There exists a polynomial-time algorithm $\mathcal{T}$ that given a CNF φ over n variables returns a CNF-formula $\mathcal{T}(\varphi)$ such that for any variable ordering π, π-OBDD$(\varphi) \le$ OBDD$(\mathcal{T}(\varphi))$; and if φ has a resolution refutation of size s, then the resolution size of $\mathcal{T}(\varphi)$ is at most $s \cdot n^{O(1)}$.*

Proof. The new algorithm $\mathcal{T}$ first applies the transformation from Lemma 22 to a CNF formula and only then applies the algorithm $\mathcal{T}_0$ from Theorem 21 to it. ◀

7 Putting the pieces together

▶ **Theorem 1.** *There exist a constant α and a polynomially computable function $\mathcal{R}$ mapping CNF formulas to CNF formulas with the following properties. For any 3-CNF φ with n variables such that: if φ is satisfiable, then $\mathcal{R}(\varphi)$ has a resolution refutation of size at most n^{α}; if φ is unsatisfiable, then any* OBDD$(\wedge$, weakening$)$ *refutation of $\mathcal{R}(\varphi)$ has size $2^{\Omega(n)}$.*

Proof. Let $\mathcal{E}$ be the algorithm from Theorem 6 with the parameter $c = 3$, and $\mathcal{T}$ be the algorithm from Corollary 23. Let n be the number of variables of φ and let n_φ be the number of variables in $\mathcal{E}(\varphi)$. Let ℓ_φ be the size of the blocks in the block partition in Theorem 6, $\ell_\varphi = O(n)$. Then let $m_\varphi = (n_\varphi \ell_\varphi)^\Delta$ where Δ is from Theorem 9 and let $\mathcal{R}(\varphi) := \mathcal{T}(\mathcal{E}(\varphi) \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi})$.

Let us first consider the case of $\varphi \in \mathrm{SAT}$. Then by Theorem 6, $\mathcal{E}(\varphi)$ has a resolution refutation π such that $|\pi| = |\varphi|^{O(1)}$ and $\mathrm{bw}(\pi) = O(1)$. Then applying Theorem 8 we get that there exists a resolution refutation of $\mathcal{E}(\varphi) \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}$ of size $|\varphi|^{O(1)}$. Then by Corollary 23 $\mathcal{T}(\mathcal{E}(\varphi) \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi})$ has a resolution refutation of size $|\varphi|^{O(1)}$.

Let us proceed with the case $\varphi \notin \mathrm{SAT}$. Suppose $\mathcal{R}(\varphi)$ has a $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ refutation of size S. Then by Corollary 23 the formula $\mathcal{E}(\varphi) \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}$ has a π-$\mathrm{OBDD}(\wedge, \mathrm{weakening})$ refutation of size S for any variable order π. Then consider the order of variables π_0 where the variables of $\mathcal{E}(\varphi) \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}$ are ordered as follows:

- All the variables corresponding to the indices in an arbitrary order (denote this set by A);
- All the variables from the first rows of the matrices (denote this set by B_1);
- ...
- All the variables from the ℓ_φth rows of the matrices (denote this set by B_{ℓ_φ}).

The size of π_0-$\mathrm{OBDD}(\wedge, \mathrm{weakening})$ refutation of $\mathcal{E}(\varphi) \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}$ is at most S which by Theorem 19 implies that $\textsf{box-dag}_{A,B_1,\ldots,B_\ell}\left(\mathrm{Search}_{\mathcal{E}(\varphi)\circ\textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}}\right) \le S^{O(\ell_\varphi + 1)}$.

Then the fact that $\mathrm{Search}_{\mathcal{E}(\varphi)\circ\textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}}$ is at least as hard as $\mathrm{Search}_{\mathcal{E}(\varphi)} \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}$ and the inequality $\textsf{box-dag}_{A,B_1,\ldots,B_\ell}\left(\mathrm{Search}_{\mathcal{E}(\varphi)} \circ \textsc{Ind}^{n_\varphi}_{\ell_\varphi \times m_\varphi}\right) \ge m_\varphi^{\Omega(\mathrm{bw}(\mathcal{E}(\varphi)))}$ implied by Theorem 9 together imply that $S \ge m_\varphi^{\Omega(\mathrm{bw}(\mathcal{E}(\varphi))/(\ell_\varphi+1))}$. By Theorem 6 using Proposition 4 to switch from decision dag to resolution refutation we have $\mathrm{bw}(\mathcal{E}(\varphi)) = \Omega(n^{c-1}) = \Omega(n^2)$ which implies that $S \ge m_\varphi^{\Omega(n)}$ since $\ell_\varphi = O(n)$. This completes the proof of the theorem since $m_\varphi \ge 2$. ◀

▶ **Corollary 24.** *If* $\mathrm{OBDD}(\wedge, \mathrm{weakening})$ *is automatable then* $\mathrm{P} = \mathrm{NP}$.

References

1 Alfonso San Miguel Aguirre and Moshe Y. Vardi. Random 3-sat and bdds: The plot thickens further. In *Principles and Practice of Constraint Programming – CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 – December 1, 2001, Proceedings*, pages 121–136, 2001. doi:10.1007/3-540-45578-7_9.

2 Michael Alekhnovich, Samuel R. Buss, Shlomo Moran, and Toniann Pitassi. Minimum propositional proof length is np-hard to linearly approximate. *J. Symb. Log.*, 66(1):171–191, 2001. doi:10.2307/2694916.

3 Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM J. Comput.*, 38(4):1347–1363, 2008. doi:10.1137/06066850X.

4 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.

5 Albert Atserias and Moritz Müller. Automating resolution is np-hard. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 498–509. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00038.

6 Maria Luisa Bonet, Carlos Domingo, Ricard Gavaldà, Alexis Maciel, and Toniann Pitassi. Non-automatizability of bounded-depth frege proofs. *Comput. Complex.*, 13(1-2):47–68, 2004. doi:10.1007/s00037-004-0183-5.

7 Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. No feasible interpolation for tc0-frege proofs. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 254–263. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646114.

8 Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagram. *ACM Computing Surveys*, 24(3):293–318, 1992.

9 Sam Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering rule makes OBDD proof systems stronger. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 16:1–16:24, 2018. doi:10.4230/LIPIcs.CCC.2018.16.

10 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*, STOC 2018, page 902–911, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188838.

11 Michal Garlík. Failure of feasible disjunction property for k-dnf resolution and np-hardness of automating it. *Electron. Colloquium Comput. Complex.*, page 37, 2020. URL: https://eccc.weizmann.ac.il/report/2020/037.

12 Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is np-hard. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 68–77, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384248.

13 Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, Dmitry Sokolov, and Susanna F. de Rezende. Automating algebraic proof systems is np-hard. *Electron. Colloquium Comput. Complex.*, 27:64, 2020. URL: https://eccc.weizmann.ac.il/report/2020/064.

14 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.

15 Kazuo Iwama. Complexity of finding short resolution proofs. In Igor Prívara and Peter Ruzicka, editors, *Mathematical Foundations of Computer Science 1997, 22nd International Symposium, MFCS'97, Bratislava, Slovakia, August 25-29, 1997, Proceedings*, volume 1295 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 1997. doi:10.1007/BFb0029974.

16 Jan Krajiček. An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. *Journal of Symbolic Logic*, 73(1):227–237, 2008. doi:10.2178/jsl/1208358751.

17 Jan Krajícek and Pavel Pudlák. Some consequences of cryptographical conjectures for s_2^1 and EF. *Inf. Comput.*, 140(1):82–94, 1998. doi:10.1006/inco.1997.2674.

18 Christoph Meinel and Anna Slobodova. On the complexity of Constructing Optimal Ordered Binary Decision Diagrams. In *Proceedings of Mathematical Foundations of Computer Science*, volume 841, pages 515–524, 1994.

19 Ian Mertz, Toniann Pitassi, and Yuanhao Wei. Short proofs are hard to find. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 84:1–84:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.84.

20 Guoqiang Pan and Moshe Y. Vardi. Symbolic techniques in satisfiability solving. *Journal of Automated Reasoning*, 35(1-3):25–50, 2005. doi:10.1007/s10817-005-9009-7.

21 Jeff M. Phillips, Elad Verbin, and Qin Zhang. Lower bounds for number-in-hand multiparty communication complexity, made easy. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 486–501, USA, 2012. Society for Industrial and Applied Mathematics.

22 Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997. doi:10.2307/2275583.

23 Nathan Segerlind. Nearly-exponential size lower bounds for symbolic quantifier elimination algorithms and OBDD-based proofs of unsatisfiability. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(009), 2007. URL: http://eccc.hpi-web.de/eccc-reports/2007/TR07-009/index.html.

24 Nathan Segerlind. On the relative efficiency of resolution-like proofs and ordered binary decision diagram proofs. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, pages 100–111. IEEE Computer Society, 2008. doi:10.1109/CCC.2008.34.

25 Dmitry Sokolov. Dag-like communication and its applications. In *Computer Science – Theory and Applications – 12th International Computer Science Symposium in Russia, CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings*, pages 294–307, 2017. doi:10.1007/978-3-319-58747-9_26.

26 I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000. URL: https://books.google.ru/books?id=xqqJj42ZoXcC.

Not All Strangers Are the Same: The Impact of Tolerance in Schelling Games

Panagiotis Kanellopoulos ✉
School of Computer Science and Electronic Engineering, University of Essex, UK

Maria Kyropoulou ✉
School of Computer Science and Electronic Engineering, University of Essex, UK

Alexandros A. Voudouris ✉
School of Computer Science and Electronic Engineering, University of Essex, UK

Abstract

Schelling's famous model of segregation assumes agents of different types, who would like to be located in neighborhoods having at least a certain fraction of agents of the same type. We consider natural generalizations that allow for the possibility of agents being tolerant towards other agents, even if they are not of the same type. In particular, we consider an ordering of the types, and make the realistic assumption that the agents are in principle more tolerant towards agents of types that are closer to their own according to the ordering. Based on this, we study the strategic games induced when the agents aim to maximize their utility, for a variety of tolerance levels. We provide a collection of results about the existence of equilibria, and their quality in terms of social welfare.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Schelling games, Equilibria, Price of anarchy, Price of stability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.60

1 Introduction

Residential segregation is a broad phenomenon affecting most metropolitan areas, and is known to be caused due to racial or socio-economic differences. The severity of its implications to society [5] is the main reason for the vast research attention it has received, with many different models being proposed over the years that aim to conceptualize it (e.g., see [23]). The most prominent of those models is that of Schelling [21, 22], which studies how motives at an individual level can lead to macroscopic behavior and, ultimately, to segregation. In particular, the individuals are modelled as agents of two different types (usually referred to using colors, such as red and blue), and the environment is abstracted by a topology (such as a grid graph), representing a city. The agents occupy nodes of the topology, and prefer neighborhoods in which the presence of their own type exceeds a specified tolerance threshold. If an agent is unhappy with her current location, then she either jumps to a randomly selected empty node of the topology, or swaps positions with another random unhappy agent. Schelling's crucial observation was that such dynamics might lead to largely segregated placements, even when the agents are relatively tolerant of mixed neighborhoods.

A recent series of papers (discussed in Section 1.2) have generalized Schelling's model to include more than two types, and have taken a game-theoretic approach, according to which the agents behave strategically rather than randomly, aiming to maximize their individual *utility*. There are many ways to define the utility of an agent i of type T. For instance, Elkind et al. [16] defined it as the ratio of the number of agents of type T in i's neighborhood over the *total* number of agents therein. Echzell et al. [15] proposed a similar definition, which however does not take into account all the agents of different type in the denominator, but only those of the *majority* type. The first definition essentially assumes that the agents view

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 60; pp. 60:1–60:14
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

all the agents of different type as *enemies*. On the other hand, the second definition assumes that the agents view only the majority type as hostile. An alternative way of thinking about these particular utility functions is as if the agents have binary *tolerance* towards other agents in the sense that agents are either friends or enemies; in the case of Elkind et al. all the neighbors of an agent are taken into account when computing her utility, whereas in the case of Echzell et al. some of her neighbors are ignored.

These functions are natural generalizations of the quantity that determines the happiness of agents in Schelling's original model for two types. However, they fail to capture realistic scenarios in which the agents do not have a single-dimensional view of the other agents, but rather have different *preferences* over the different types of agents. For example, suppose that the agents correspond to voters while the types correspond to political parties. In this case the preferences of voters over other voters are defined based on the distances of the political views expressed by the parties they are affiliated with. Another example is when the types correspond to research areas, in which case people working on a specific research agenda will be more willing to collaborate with other people working on related problems.

1.1 Our Contribution

To capture scenarios like the examples above, we propose a clean model that naturally extends the model of Elkind et al. [16] by incorporating *different levels of tolerance* among agent types, and study the induced strategic games in terms of the existence and quality of their equilibria; in Section 5, we discuss potential generalizations of our model.

To be more specific, our model consists of a set of agents who are partitioned into $\lambda \geq 2$ types of equal size, a graph topology, and an ordering of the different agent types which determines the relative tolerance among agents of different types. Naturally, we assume that there is higher tolerance between agents whose types are closer according to the ordering. The exact degree of tolerance between the different types is specified by a *tolerance vector*, which consists of weights representing the tolerance between the different types depending on their distance in the given ordering. For example, agents of the same type are in distance 0 and are fully tolerant towards each other, which is captured by a weight of 1. The utility of an agent can then be computed as a weighted average of the tolerance that she has towards her neighbors, and every agent aims to occupy a node of the topology to maximize her utility; agents are allowed to unilaterally *jump* to empty nodes to increase their utility.

We study the dynamics of such *tolerance Schelling games*. We first focus on questions related to equilibrium existence. For general games, we show that equilibria are not guaranteed to exist if agents are not fully tolerant towards agents in type-distance 1 (Theorem 2). We complement this impossibility by showing many positive results for important subclasses of games, in which the topology is a structured graph (such a 4-grid or a tree) and the tolerance vector satisfies certain properties (Theorems 3, 5, 6 and 7). We then turn our attention to the quality of equilibria measured by the social welfare objective, defined as the total utility of the agents, and prove (asymptotically tight) bounds on the price of anarchy [19] (Theorems 8 and 9) and price of stability [2] (Theorem 14), which depend on the number of types, the number of agents and/or the tolerance parameters.

1.2 Related Work

Residential segregation, and Schelling's original randomized model in particular, has been the basis of a continuous stream of multidisciplinary research in Sociology [13], Economics [20, 25], Physics [24], and Computer Science [4, 6, 8, 9, 17].

Most related to our work is a quite recent series of papers in the TCS and AI communities, which deviated from the premise of random behavior, and instead studied the strategic games induced when the agents act as utility-maximizers. Chauhan et al. [12] studied questions related to dynamics convergence in games with two types of agents who can either jump to empty nodes of a topology (as in our case) or *swap* locations with other agents to minimize a *cost* function; their model was generalized to multiple types of agents by Echzell et al. [15]. In this paper we extend the utility model of Elkind et al. [16], who initially refined the cost model of Chauhan et al. [12]. They introduced a simpler utility function (fraction of same-type agents in the one's neighborhood) which the agents aim to maximize, and studied the existence, complexity and quality of equilibria in jump games with multiple types of agents and general topologies. They also proposed many interesting variants, such as *enemy aversion* (agents might prefer being alone to being in a group full of agent of different type than their own) and *social Schelling games* (where the agents types are determined by a social network), which have been partially studied by Kanellopoulos et al. [18] and Chan et al. [11], respectively. Agarwal et al. [1] studied similar existence, complexity and qualitative questions for swap games. Bilò et al. [7] also focused on swap games, and in particular, on a constrained setting, where the agents can only view a small part of the topology near their current location and can only swap with agents in this part of the topology. Finally, Bullinger et al. [10] and Deligkas et al. [14] studied the (parameterized) complexity of computing assignments with good welfare guarantees, focusing on the social welfare, Nash welfare, and Pareto optimality, and many other welfare objectives.

2 Preliminaries

A λ-*type tolerance Schelling game* consists of:

- A set N of $n \geq 4$ *agents*, partitioned into $\lambda \geq 2$ disjoint sets $T_1, \dots, T_\lambda$ representing *types*, such that $\bigcup_{\ell \in [\lambda]} T_\ell = N$.
- A simple connected undirected graph $G = (V, E)$ called *topology*, such that $|V| > n$.
- A *tolerance vector* $\mathbf{t}_\lambda = [t_0, \dots, t_{\lambda-1}]$ consisting of λ parameters, such that t_d represents the tolerance that agents of type T_ℓ have towards agents of type T_k in Manhattan distance $|\ell - k| = d \in \{0, \dots, \lambda - 1\}$ according to a given ordering $\succ$ of the types (say, $T_1 \succ \dots \succ T_\lambda$). We assume that agents are more tolerant towards agents of types that are closer to their own according to $\succ$, and we thus have that $1 = t_0 \geq \dots \geq t_{\lambda-1} \geq 0$. We also assume that $t_{\lambda-1} < 1$; otherwise, all agents are completely tolerant towards all others and the game is trivial. Let $\tau = \sum_{d=0}^{\lambda-1} t_d$ be the sum of all tolerance parameters.

Clearly, the class of λ-type tolerance Schelling games includes as a special case the classic Schelling games studied in the related literature (e.g., see [16]), for which $t_0 = 1$ and $t_d = 0$ for every $d \in \{1, \dots, \lambda - 1\}$. Because of this particular tolerance vector, we will use the term λ-*type zero-tolerance games* to refer to the classic Schelling games.

In this paper we consider *balanced* games, in which the agents are partitioned in types of equal size, such that $|T_\ell| = n/\lambda \geq 2$ for every $\ell \in [\lambda]$; thus, n is a multiple of λ. Balanced games are the most fundamental ones that admit non-trivial and interesting results[1]. We use the abbreviation λ-*TS* to refer to such a balanced λ-type tolerance Schelling game $\mathcal{I} = (N, G, \mathbf{t}_\lambda)$. For convenience, we will also use the abbreviation λ-*ZTS* to refer to a balanced λ-type zero-tolerance game $\mathcal{I} = (N, G)$.

[1] Note that equilibria are not guaranteed to exist even for balanced games (see Theorem 2), while it is not hard to observe that the price of anarchy can be unbounded when the sizes of the types are arbitrary [16].

Let $\mathbf{v} = (v_i)_{i \in N}$ be an *assignment* specifying the node v_i of G that each agent $i \in N$ *occupies*, such that $v_i \neq v_j$ for $i \neq j$. The *neighborhood* of a node v consists of the nodes at distance 1 from v in G. For every node v, we denote by $n_\ell(v|\mathbf{v})$ the number of agents of type T_ℓ that occupy nodes in the neighborhood of v according to the assignment $\mathbf{v}$, and also let $n(v|\mathbf{v}) = \sum_{\ell \in [\lambda]} n_\ell(v|\mathbf{v})$. Given an assignment $\mathbf{v}$, the utility of agent i of type T_ℓ is computed as

$$u_i(\mathbf{v}) = \frac{1}{n(v_i|\mathbf{v})} \sum_{k \in [\lambda]} t_{|\ell-k|} \cdot n_k(v_i|\mathbf{v}),$$

if $n(v_i|\mathbf{v}) \neq 0$, and 0 otherwise (in which case we say that the agent is *isolated*). The agents are *strategic* and aim to maximize their utility by *jumping* to empty nodes of the topology if they can increase their utility by doing so. We say that an assignment $\mathbf{v}$ is an *equilibrium* if no agent i of any type T_ℓ has incentive to jump to any empty node v of the topology, that is, $u_i(\mathbf{v}) \geq u_i(v, \mathbf{v}_{-i})$, where $(v, \mathbf{v}_{-i})$ is the assignment resulting from this jump. Let $\text{EQ}(\mathcal{I})$ be the set of equilibrium assignments of a given λ-TS game $\mathcal{I}$.

The *social welfare* of an assignment $\mathbf{v}$ is defined as the total utility of the agents, that is,

$$\text{SW}(\mathbf{v}) = \sum_{i \in N} u_i(\mathbf{v}).$$

Let $\text{OPT}(\mathcal{I}) = \max_{\mathbf{v}} \text{SW}(\mathbf{v})$ be the maximum social welfare among all possible assignments in the λ-TS game $\mathcal{I}$. For a given subclass $\mathcal{C}$ of λ-TS games, the *price of anarchy* is defined as the worst-case ratio, over all possible games $\mathcal{I} \in \mathcal{C}$ such that $\text{EQ}(\mathcal{I}) \neq \varnothing$, between $\text{OPT}(\mathcal{I})$ and the *minimum* social welfare among all equilibria:

$$\text{PoA}(\mathcal{C}) = \sup_{\mathcal{I} \in \mathcal{C}: \text{EQ}(\mathcal{I}) \neq \varnothing} \frac{\text{OPT}(\mathcal{I})}{\min_{\mathbf{v} \in \text{EQ}(\mathcal{I})} \text{SW}(\mathbf{v})}.$$

Similarly, the *price of stability* takes into account the ratio between $\text{OPT}(\mathcal{I})$ and the *maximum* social welfare among all equilibria:

$$\text{PoS}(\mathcal{C}) = \sup_{\mathcal{I} \in \mathcal{C}: \text{EQ}(\mathcal{I}) \neq \varnothing} \frac{\text{OPT}(\mathcal{I})}{\max_{\mathbf{v} \in \text{EQ}(\mathcal{I})} \text{SW}(\mathbf{v})}.$$

3 Equilibrium Existence

In this section, we show several positive and negative results about the existence of equilibrium assignments, for interesting subclasses of tolerance Schelling games. We start with the relation of equilibrium assignments in λ-ZTS games and general λ-TS games.

▶ **Theorem 1.** *Consider a λ-ZTS game $\mathcal{I} = (N, G)$ and a λ-TS game $\mathcal{I}' = (N, G, \mathbf{t}_\lambda)$. For $\lambda = 2$, $EQ(\mathcal{I}') \subseteq EQ(\mathcal{I})$ and $EQ(\mathcal{I}) \setminus EQ(\mathcal{I}')$ consists of assignments with isolated agents. For $\lambda \geq 3$, $EQ(\mathcal{I})$ and $EQ(\mathcal{I}')$ are incomparable.*

Proof. We start with $\lambda = 2$; for convenience, we will refer to the two types as red and blue. Let $\mathbf{v}$ be an equilibrium of $\mathcal{I}'$. Clearly, for $\mathcal{I}$ and $\mathcal{I}'$ to be different, it must be the case that $t_1 > 0$. Consequently, there are no isolated agents in $\mathbf{v}$ as they would have incentive to deviate to nodes that are adjacent to any other agent and increase their utility from 0 to (at least) t_1. We will show that $\mathbf{v}$ is an equilibrium of $\mathcal{I}$ as well. Without loss of generality, consider a red agent i who occupies a node v_i that is adjacent to $n_r(v_i)$ red and $n_b(v_i)$ blue agents. Since agent i is not isolated, it holds that $n_r(v_i) + n_b(v_i) \geq 1$. If $n_b(v_i) = 0$, then agent i has maximum utility 1 in both $\mathcal{I}$ and $\mathcal{I}'$. Hence, we can assume that $n_b(v_i) \geq 1$. Since $\mathbf{v}$ is an equilibrium of $\mathcal{I}'$, agent i has no incentive to unilaterally jump to any empty node v of the topology. That is,

$$\frac{n_r(v_i) + t_1 \cdot n_b(v_i)}{n_r(v_i) + n_b(v_i)} \geq \frac{n_r(v) + t_1 \cdot n_b(v)}{n_r(v) + n_b(v)} \Leftrightarrow (1 - t_1) \left(\frac{n_r(v_i)}{n_b(v_i)} - \frac{n_r(v)}{n_b(v)} \right) \geq 0,$$

where $n_r(v)$ and $n_b(v)$ are the number of red and blue agents that are adjacent to v *after* agent i jumps to v; observe that $n_b(v) \geq 1$, as otherwise agent i would obtain maximum utility of 1 by jumping to v, contradicting that $\mathbf{v}$ is an equilibrium of $\mathcal{I}'$. Since $t_1 < 1$, we equivalently have that

$$\frac{n_r(v_i)}{n_b(v_i)} \geq \frac{n_r(v)}{n_b(v)} \Leftrightarrow \frac{n_r(v_i)}{n_r(v_i) + n_b(v_i)} \geq \frac{n_r(v)}{n_r(v) + n_b(v)}.$$

Therefore, agent i has no incentive to jump to the empty node v in $\mathcal{I}$, and $\mathbf{v}$ is an equilibrium of $\mathcal{I}$ as well. Using similar arguments, we can show that any equilibrium of $\mathcal{I}$ such that there is no isolated agent is also an equilibrium of $\mathcal{I}'$.

For $\lambda \geq 3$, to show that $\text{EQ}(\mathcal{I})$ is incomparable to $\text{EQ}(\mathcal{I}')$, consider the tolerance vector $\mathbf{t_3} = (1, 1/2, 0)$ and the following two partial assignments $\mathbf{v}$ and $\mathbf{v}'$:

- In $\mathbf{v}$, an agent i of type T_1 occupies a node v_i which is adjacent to two nodes, one occupied by an agent of type T_1 and one occupied by an agent of type T_3. There is also an empty node v which is adjacent to two nodes, one occupied by an agent of type T_1 and one occupied by an agent of type T_2. In $\mathcal{I}$, agent i has no incentive to jump from v_i to v as both nodes give her utility 1/2. On the other hand, in $\mathcal{I}'$, agent i has utility $(1+t_2)/2 = 1/2$ and has incentive to jump to v to increase her utility to $(1+t_1)/2 = 3/4$. Hence, $\mathbf{v}$ can be an equilibrium of $\mathcal{I}$, but not of $\mathcal{I}'$.
- In $\mathbf{v}'$, an agent i of type T_1 occupies a node v_i which is adjacent to three nodes, one occupied by an agent of type T_1, one occupied by an agent of type T_2 and one occupied by an agent of type T_3. There is also an empty node v which is adjacent to two nodes, one occupied by an agent of type T_1 and one occupied by an agent of type T_3. In $\mathcal{I}$, agent i has incentive to jump from v_i to v in order to increase her utility from 1/3 to 1/2. However, in $\mathcal{I}'$, agent i has no incentive to jump as she has utility $(1 + t_1)/3 = 1/2$ by occupying node v_i, which is exactly the utility she would also obtain by jumping to v. Consequently, $\mathbf{v}'$ can be an equilibrium of $\mathcal{I}'$, but not of $\mathcal{I}$.

This completes the proof. ◀

Since there exist simple 2-ZTS games that do not admit any equilibria [16], the first part of Theorem 1 implies that equilibria are not guaranteed to exist for general 2-TS games as well. In fact, by carefully inspecting the proof of Elkind et al. [16] that λ-ZTS games played on trees do not always admit equilibria for every $\lambda \geq 2$, we can show the following stronger impossibility result.

▶ **Theorem 2.** *For every $\lambda \geq 2$ and every tolerance vector $\mathbf{t}_\lambda$ such that $t_1 < 1$, there exists a λ-TS game $\mathcal{I} = (N, G, \mathbf{t}_\lambda)$ in which G is a tree and does not admit any equilibrium.*

Since Theorem 2 implies that it is impossible to hope for general positive existence results, in the remainder of this section we focus on games with structured topologies and tolerance vectors. In particular, we consider the class of α-binary λ-TS games with $\alpha \in \{1, \dots, \lambda\}$ in which the tolerance vector $\mathbf{t}_\lambda$ is such that

$$t_d = \begin{cases} 1, & \text{if } d < \alpha \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, the class of 1-binary λ-TS games coincides with that of λ-ZTS.

We next show that when the topology is a grid[2] or a tree, there exist values of $\alpha \in \{1, \dots, \lambda\}$ for which α-binary λ-TS games played on such a topology always admit at least one equilibrium. Our first result for grids is the following.

▶ **Theorem 3.** *Every 2-ZTS game $\mathcal{I} = (N, G)$ in which G is a grid admits at least one equilibrium.*

The proof of Theorem 3 is constructive and such that in the computed equilibrium no agent is isolated. Consequently, in combination with Theorem 1, it further implies the following:

▶ **Corollary 4.** *Every 2-TS game $\mathcal{I} = (N, G, \mathbf{t_2})$ in which G is a grid admits at least one equilibrium.*

Unfortunately, showing a result similar to Theorem 3 for every $\lambda \geq 3$ is a very challenging task. Instead, we show the following result for 2-binary games.

▶ **Theorem 5.** *Every 2-binary λ-TS game $\mathcal{I} = (N, G, \mathbf{t}_\lambda)$ in which G is a grid admits at least one equilibrium.*

Proof. Consider a 2-binary λ-TS game with n agents played on an $m \times M$ grid (m rows and M columns) such that $m \leq M$. Let $x = n/\lambda \geq 2$ be the number of agents per type and $e = mM - n$ be the number of empty nodes.

We compute an equilibrium assignment $\mathbf{v}$ using Algorithm 1, which in turn relies on the TILE procedure described in Algorithm 2. In particular, Algorithm 2 considers the yet unassigned agents in increasing type according to the ordering $\succ$, and assigns them in an $r \times M$ subgrid having row s as the top row, so that the k leftmost nodes of the top row are left empty, while all other nodes host an agent (assuming the number of unassigned agents is large enough). TILE visits these rows in a column-major order, skipping the empty nodes. Informally, Algorithm 1 repeatedly calls Algorithm 2 to compute an assignment for consecutive sub-grids, along the largest dimension of the topology. The exact size of the sub-grid considered at each time is determined by the number of remaining rows of the topology.

Algorithm 1 terminates immediately (at any step) when all agents have been assigned. First, observe that if it terminates in Lines 3 or 7, each agent of type ℓ has neighbors of types in $\{\ell - 1, \ell, \ell + 1\}$, and, hence, $\mathbf{v}$ is an equilibrium. Note that the algorithm cannot terminate at Line 11 since $e < M$, so let us assume that the algorithm terminates in Line 14. Again, all agents placed in Line 3 have utility 1. Each agent i of type ℓ placed during Line 11 has utility at least 2/3; indeed, i has at least one neighbor of type ℓ, at least one neighbor of a type in $\{\ell - 1, \ell + 1\}$, and at most one neighbor of type at distance at least 2. If $\alpha = 1$ all agents placed in Line 14 have utility 1. Otherwise, agents placed in Line 14 at the last $x - 1$ rows have utility 1, while any agent on the row with the empty nodes has utility at least 1/2 when $e = M - 1$, and at least 2/3 otherwise.

[2] We focus on 4-grids where internal nodes have 4 neighbors.

Algorithm 1 Equilibrium construction for a 2-binary λ-TS game on an $m \times M$ grid.

```
   /* $x$: number of agents per type                                              */
   /* $e$: number of empty nodes                                                  */
   /* The algorithm terminates immediately when all agents have been assigned.    */
1  Initialize $k = 0$
2  while $x \leq m - k$ and $e \geq M$ do
3      Tile$(k+1, x, 0)$
4      leave the next row empty
5      update $k := k + x + 1$, $e := e - M$
6  if $x > m - k$ then
7      Tile$(k+1, m-k, 0)$
8  else /* In this case it holds that $e < M$ and $x \leq m - k$                 */
9      Define non-negative integers $\alpha \in \mathbb{N}_{>0}$ and $\beta \leq x - 1$ such that $m - k = \alpha x + \beta$
10     for $i = 1, \dots, \alpha - 1$ do
11         Tile$(k+1, x, 0)$
12         update $k := k + x$
13     if $\beta = 0$ then
14         Tile$(k+1, x, e)$
15     else if $\beta = 1$ then
16         if Line 3 was executed then
17             Shift all agents down by one row
18             Tile$(1, 1, e)$
19             Tile$(k+2, x, 0)$
20         else
21             Tile$(k+1, 1, e)$
22             Tile$(k+2, x, 0)$
23     else
24         Tile$(k+1, x, 0)$
25         Tile$(k+x+1, \beta, e)$
```

Algorithm 2 Tile(s, r, k).

```
/* $s, r$: starting row and number of rows defining an $r \times M$ grid         */
/* $k$: number of nodes to be left empty                                          */
for $i = 1$ to $k$ do
    mark node $(s, i)$ as empty
for $j = 1$ to $M$ do
    for $i = s$ to $s + r - 1$ do
        if node $(i, j)$ is unmarked then
            place the next agent (if one exists) according to the ordering $\succ$ at node
             $(i, j)$
```

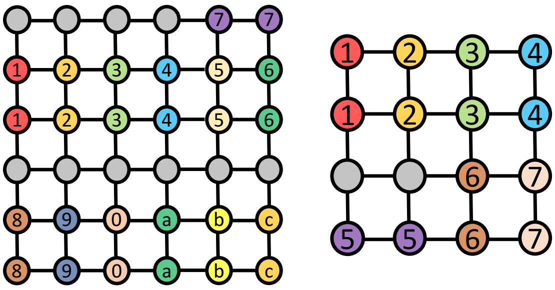


Figure 1 On the left, an example of how Algorithm 1 operates when it terminates in Line 19. On the right, an example when the algorithm terminates in Line 14. Agents of the same number and color are of the same type, while $\succ$ is $\{1, 2, \dots, 9, 0, a, b, c\}$.

If the algorithm terminates in Line 19 (see also the leftmost part of Figure 1), agents placed in Lines 3, 11, or 19 have utility at least 2/3, while agents placed in Line 18 have utility at least 1/2. If the algorithm terminates in Line 22, agents placed in Line 3 have utility 1, agents placed in Lines 11 and 22 have utility at least 2/3, while agents placed in Line 21 have utility at least 2/3 except (perhaps) the first and the last agent on the row that have utility at least 1/3. If the algorithm terminates in Line 25, again all agents placed in Line 3 have utility 1, while agents placed in Lines 11 and 24 have utility at least 2/3. Finally, the agents placed in Line 25 have utility at least 1/2 if $e = M - 1$ and at least 2/3 otherwise.

We now argue that no agent has an incentive to jump. Note that an empty node may have another empty node as a top or bottom neighbor if the algorithm terminates in Line 3, or in Line 7, or in Line 14 in case $\alpha = 1$. In all these cases, by the discussion above, all agents have utility 1 and the assignment is an equilibrium. Also, note that an empty node has always a bottom neighbor, while the only case the empty node has no top neighbor is if the algorithm terminates in Line 19. In that case, any agent with utility less than 1 can obtain utility at most 1/2 by jumping; again, $\mathbf{v}$ is an equilibrium.

So, in the following we assume that any empty node has a top and bottom neighboring agent. Observe that, in that case, an agent gets utility at most 2/3 by jumping to an empty node, since either the top or the bottom neighbor will have a large type distance and there is no left neighbor. As in almost all cases, agents in $\mathbf{v}$ have utility at least 2/3, it remains to argue about the nodes that have utility less than that. The agent in Line 14 with utility 1/2 (when $\alpha > 1$) obtains utility at most 1/2 by jumping, the agents in Line 21 with utility at least 1/3 obtain utility at most 1/3 by jumping, and, finally, the agent in Line 25 with utility 1/2 obtains utility at most 1/2 by jumping. We conclude that $\mathbf{v}$ is an equilibrium and the theorem follows. ◀

Note that Algorithm 1 may fail to return an equilibrium for lexicographically larger tolerance vectors. Indeed, consider a 4×4 grid and 7 types of two agents each. Algorithm 1 puts agents of types 1 to 4 in each of the first two rows, skips 2 nodes, puts agents of types 6 and 7 in the third row, and places agents of types 5, 5, 6 and 7 in the last row; see also the rightmost example in Figure 1. Under tolerance vector $\mathbf{t_7} = \{1, 1, 0, 0, 0, 0, 0\}$ the assignment is an equilibrium (by Theorem 5), while under tolerance vector $\mathbf{t}'_\mathbf{7} = \{1, 1, t_2 > \frac{1}{2}, 0, 0, 0, 0\}$, the agent of type 4 in the second row has utility 2/3, but can obtain utility $\frac{1+2t_2}{3} > 2/3$ by jumping to the rightmost empty node.

So, a different algorithm is needed for computing equilibria in α-binary games with $\alpha \geq 3$. While we have not been able to show this result for every α, we do show it for $\alpha \geq \sqrt{\lambda}$. In particular, the equilibrium constructed in the proof of the next theorem guarantees a utility of 1 to all agents, and thus it is also an equilibrium for games with lexicographically larger tolerance vectors, not necessarily binary ones.

Algorithm 3 Equilibrium construction for a $\lfloor\frac{\lambda}{2}\rfloor$-binary λ-TS game on a tree (or games with lexicographically larger tolerance vectors).

```
   /* $tree_1,\dots,tree_k$ denote the subtrees of the tree topology in non-increasing order
      by size, when the topology is rooted at a centroid node.                          */
1  Run Bottom-Up($tree_1, T_1, T_2, \dots, T_{\lceil\frac{\lambda}{2}\rceil}$). If at least one agent of type $T_1$ remains
     unassigned, repeat with the next subtree. Let $a \leq \lceil\frac{\lambda}{2}\rceil$ be the smallest type index
     among unassigned agents, and let $tree_{k_1}$ be the last subtree considered in this step.
2  Run Bottom-Up($tree_{k_1+1}, \mathcal{T}_\lambda, \mathcal{T}_{\lambda-1}, \dots, \mathcal{T}_{\lceil\frac{\lambda+1}{2}\rceil}$), where $\mathcal{T}_i$ are the unassigned
     agents of types $T_i$, $i = \lambda, \dots, \lceil\frac{\lambda+1}{2}\rceil$. If at least one agent of type $T_\lambda$ remains
     unassigned, repeat with the next subtree. Let $b \geq \lceil\frac{\lambda+1}{2}\rceil$ be the largest type index
     among unassigned agents, and let $tree_{k_2}$ be the last subtree considered in this step.
3  Run Bottom-Up($tree_{k_2+1}, \mathcal{T}_a, \mathcal{T}_{a+1}, \dots, \mathcal{T}_b$), where $\mathcal{T}_i$ are the unassigned agents of
     types $T_i$, $i = a, \dots, b$. Repeat with the next subtree and the unassigned agents of
     these types, until all agents have been assigned.
4  If the last subtree among the ones considered in the previous steps contains at least
     two isolated agents, then rearrange them within this subtree so that each of them
     has at least one neighbor. If the last subtree contains a single isolated agent, then
     move this agent to the root of the tree.
```

▶ **Theorem 6.** *For $\lambda \geq 3$, every $\sqrt{\lambda}$-binary λ-TS game $\mathcal{I} = (N, G, \mathbf{t}_\lambda)$ in which G is a grid admits at least one equilibrium.*

Next we turn our attention to games in which the topology is a tree. We show the following result for α-binary games when $\lambda \geq 3$.

▶ **Theorem 7.** *Every 2-binary 3-TS game $\mathcal{I} = (N, G, \mathbf{t_3})$ and every α-binary λ-TS game $\mathcal{I} = (N, G, \mathbf{t}_\lambda)$ where $\alpha \geq \lfloor\frac{\lambda}{2}\rfloor$ for $\lambda \geq 4$, in which G is a tree, admit at least one equilibrium.*

Proof. To construct an equilibrium, we exploit the following known property of trees: Every tree with $x \geq 3$ nodes contains a *centroid* node, whose removal splits the tree into at least two subtrees with at most $x/2$ nodes each. We root the tree from such a centroid node, and leave the root empty. This leads to a partition of the topology in $k \geq 2$ subtrees, which we order in non-increasing size and denote by $tree_1, \dots, tree_k$.

To assign the agents we use Algorithm 3, which in turn uses the Bottom-Up allocation procedure (described in Algorithm 4). The procedure Bottom-Up($tree, \mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_s$) assigns the unassigned agents of types $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_s$ to the nodes of the subtree *tree* from bottom to top (higher to lower depth), so that all the agents of $\mathcal{T}_1$ are covered by either agents of the same type or agents of type $\mathcal{T}_2$, and the assignment for the remaining agents is connected. Informally, Algorithm 3 roots the topology at a centroid node and considers subtrees in non-increasing size. As long as agents of type T_1 are remaining, Algorithm 3 applies the Bottom-Up procedure to the next subtree with agents in increasing type index. Then, as long as agents of type T_λ are remaining, Algorithm 3 applies the Bottom-Up procedure to the next subtree with agents in decreasing type index. The remaining (smaller) subtrees are filled with the remaining agents, again using the Bottom-Up procedure.

We first claim that at the end of Step 3 of Algorithm 3, every agent either gets utility 1 or gets utility 0 if she is isolated. Indeed, it holds that agents of type T_1 can only be adjacent to agents of type T_1 and T_2. Similarly, the agents of type T_λ can only be adjacent to agents of type T_λ and $T_{\lambda-1}$. In addition, by design, the maximum type distance among all the other agents assigned in Steps 1 and 2 is $\lceil\frac{\lambda}{2}\rceil - 2$. By this discussion, all agents have utility 1 when $\lambda = 3$ and the game is 2-binary. Below, we assume that $\lambda \geq 4$.

Algorithm 4 Bottom-Up($tree, \mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_s$).

```
/* For $i = 1, \dots, s$, $\mathcal{T}_i$ is the set of unassigned agents of a given type */
/* The algorithm terminates immediately when all agents have been assigned or all
   nodes of $tree$ have been occupied. */
1 Start at the lowest level of $tree$ and place agents of type $\mathcal{T}_1$ so that an agent of type
   $\mathcal{T}_1$ is placed at level $h$ only if all nodes at levels at least $h + 1$ have been filled.
   Furthermore, and assuming the previous condition holds, after filling a node at level
   $h$ we give priority to its sibling nodes. Continue until all agents of type $\mathcal{T}_1$ have been
   assigned.
2 Consider the agents of type $\mathcal{T}_2$. Begin by placing an agent of type $\mathcal{T}_2$ to any empty
   node having a child occupied by an agent of type $\mathcal{T}_1$ and repeat until the parent
   nodes of all agents of type $\mathcal{T}_1$ are occupied. This is feasible as long as there are at
   least as many agents of type $\mathcal{T}_2$ as there are agents of type $\mathcal{T}_1$. Continue by placing
   agents of type $\mathcal{T}_2$ arbitrarily in $tree$ by maintaining a connected assignment.
3 Arbitrarily assign the remaining agents in order of input so that the assignment
   remains connected after assigning each agent.
```

To see the claim is true for agents assigned in Step 3, observe that if Step 2 is applied on a subtree of at least $n/3$ nodes, then since we visit subtrees in non-increasing order of their size, Step 1 is also applied on a subtree of at least $n/3$ nodes. Hence, at most $n/3$ agents remain to be allocated. Otherwise, if no subtree on which Step 2 is applied has at least $n/3$ nodes, then, again due to the order we visit subtrees, any subtree to which we perform Step 3 has less than $n/3$ nodes. In any case, at most $n/3$ agents will be allocated at Step 3 at any given subtree. These agents belong to at most $\lceil \lambda/3 \rceil + 1$ different types and, due to Steps 2 and 3 in Algorithm 4, we are guaranteed that no agent allocated in Step 3 will have a neighbor of type-distance $\lceil \lambda/3 \rceil$. Since $\lceil \lambda/3 \rceil - 1 \leq \lfloor \lambda/2 \rfloor - 1$, such agents either get utility 1, or 0 if they are isolated, as required.

It remains to argue that after a possible execution of Step 4, no agent has a profitable deviation. We distinguish between the following two cases when Step 4 is performed:

- Case I: There are at least two isolated agents in the last subtree among those considered in the first three steps. First observe that, since the subtrees are considered in non-increasing order by size and the last subtree contains at least two agents, there is no subtree with a single isolated agent. Now, by the definition of the bottom-up-like allocation algorithm, all these agents must be of the last type T_b, since if agents of two or more types are assigned in the same subtree, the resulting assignment therein is by construction connected. Therefore, by rearranging the agents of type T_b in the last subtree so that all of them have at least one neighbor, each of them gets utility 1 and the assignment is an equilibrium.
- Case II: There is a single isolated agent i in the last subtree of the last type T_b considered, who is moved to the root of the tree. Since Step 4 is performed, all the subtrees that have been considered in the first three steps are full, with the exception of the last subtree which has been left empty after moving agent i. Thus, the empty nodes of the topology are only adjacent to other empty nodes or the root. As a result, an agent of some type $\ell \in [\lambda]$ would be able to get utility $t_{|\ell-b|}$ by jumping to an empty node that is adjacent to the root, and utility 0 by jumping to any other empty node. However, every agent $j \neq i$ already has utility at least $t_{|\ell-b|}$. In particular, agent j has utility 1 if she is not adjacent to the root, utility at least $\frac{1+t_{|\ell-b|}}{2} \geq t_{|\ell-b|}$ if she is adjacent to the root but not isolated before moving i to the root, and utility exactly $t_{|\ell-b|}$ if she is adjacent to the root and was isolated before moving i to the root.

This completes the proof. ◀

For $\lambda = 3$, Theorem 7 is tight in the sense that equilibria are not guaranteed to exist when $t_1 < 1$ (Theorem 2). For $\lambda \geq 4$, it is not hard to observe that the assignment computed is also an equilibrium in games with lexicographically larger vectors (not necessarily binary ones) than the one stated.

4 Quality of Equilibria

In this section, we consider the quality of equilibria measured in terms of social welfare, and bound the price of anarchy and price of stability. Recall that these notions compare the social welfare achieved in the *worst* and *best* equilibrium to the maximum possible social welfare achieved in any assignment. We start with a general upper bound on the price of anarchy, whose proof follows by bounding the social welfare at equilibrium by the total utility the agents would be able to obtain by jumping to an arbitrary empty node. Recall that $\tau = \sum_{d=0}^{\lambda-1} t_d$.

▶ **Theorem 8.** *The price of anarchy of λ-TS games with tolerance vector $\mathbf{t}_\lambda$ is at most $\frac{\lambda n}{\tau n - \lambda}$.*

Proof. Consider a λ-TS game $\mathcal{I} = (N, G, \mathbf{t}_\lambda)$ with $\text{EQ}(\mathcal{I}) \neq \varnothing$. Let $\mathbf{v}$ be an equilibrium, and denote by v an empty node. The utility that an agent of type T_ℓ, $\ell \in [\lambda]$ would obtain by unilaterally jumping to v is

- $\frac{1}{n(v)} \sum_{k\in[\lambda]} t_{|\ell-k|} \cdot n_k(v)$ if she is not adjacent to v;
- $\frac{1}{n(v)-1} \left(\sum_{k\in[\lambda]} t_{|\ell-k|} \cdot n_k(v) - 1\right)$ otherwise.

Also observe that for every type T_ℓ, $\ell \in [\lambda]$ there are exactly $\frac{n}{\lambda} - n_\ell(v)$ agents that are not adjacent to v, and $n_\ell(v)$ agents that are adjacent to v. Since $\mathbf{v}$ is an equilibrium, every agent of type T_ℓ is guaranteed to have at least as much utility as if she were to deviate to v, and therefore the social welfare is

$$\begin{aligned}
\text{SW}(\mathbf{v}) &\geq \frac{1}{n(v)} \sum_{\ell\in[\lambda]} \left(\frac{n}{\lambda} - n_\ell(v)\right) \sum_{k\in[\lambda]} t_{|\ell-k|} \cdot n_k(v) \\
&\quad + \frac{1}{n(v)-1} \sum_{\ell\in[\lambda]} n_\ell(v) \cdot \left(\sum_{k\in[\lambda]} t_{|\ell-k|} \cdot n_k(v) - 1\right) \\
&\geq \frac{1}{n(v)} \sum_{\ell\in[\lambda]} \left(\frac{n}{\lambda} \sum_{k\in[\lambda]} t_{|\ell-k|} \cdot n_k(v) - n_\ell(v)\right) \\
&= \frac{1}{n(v)} \sum_{\ell\in[\lambda]} n_\ell(v) \cdot \left(\frac{n}{\lambda} \sum_{k\in[\lambda]} t_{|\ell-k|} - 1\right) \\
&= \frac{1}{\lambda \cdot n(v)} \sum_{\ell\in[\lambda]} n_\ell(v) \left(n \sum_{k\in[\lambda]} t_{|k-\ell|} - \lambda\right).
\end{aligned}$$

The second inequality is due to increasing the denominator of the second fraction. The first equality follows by aggregating the factors of $n_\ell(v)$ for every $\ell \in [\lambda]$. Finally, the second equality follows by factorizing λ. Now observe that because the tolerance vector $\mathbf{t}_\lambda$ is non-increasing, we have that $\sum_{k\in[\lambda]} t_{|\ell-k|} \geq \sum_{d=0}^{\lambda-1} t_d = \tau$. Combining this together with the fact that $n(v) = \sum_{\ell\in[\lambda]} n_\ell(v)$, we obtain

$$\text{SW}(\mathbf{v}) \geq \frac{\tau n - \lambda}{\lambda}.$$

The bound on the price of anarchy follows by the fact that the optimal welfare is at most n (the maximum utility of any agent is 1). ◀

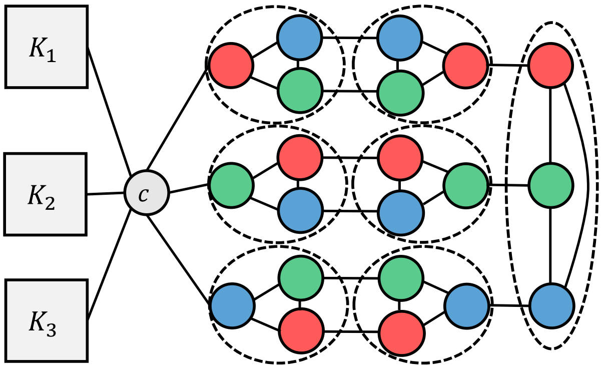


Figure 2 An instance used for the proof of Theorem 9 for the case of 3 types and 21 agents, so that each type has 7 agents. The big squares K_1, K_2, K_3 correspond to cliques of size 7 (the number of agents per type), while the ovals represent cliques of size 3 (the number of types). In an optimal assignment, each large clique contains agents of the same type and each agent gets utility 1. In a bad equilibrium, each small clique contains a single agent of each type and all gray nodes are left empty. For each type $\ell \in [3]$, all but one agents of type ℓ get utility $\tau_\ell/3$, while the last agent gets utility $(\tau_\ell - 1)/2$.

For each $\ell \in \{1, \dots, \lambda\}$, let $\tau_\ell = \sum_{k\in[\lambda]} t_{|\ell-k|}$ be the total tolerance of agents of type ℓ towards any subset containing one agent of every type. We can show the following general lower bound on the price of anarchy, as a function of these parameters; see Figure 2 for a sketch of the proof for $\lambda = 3$.

▶ **Theorem 9.** *The price of anarchy of λ-TS games with tolerance vector $\mathbf{t}_\lambda$ is at least*

$$\frac{\lambda n}{\frac{\sum_{\ell\in[\lambda]} \tau_\ell}{\lambda} n - \frac{\lambda^2 - \sum_{\ell\in[\lambda]} \tau_\ell}{\lambda - 1}} \geq \frac{\lambda n}{\frac{2(\lambda-1)\tau}{\lambda} n - \frac{\lambda^2}{\lambda-1} + 2\tau}.$$

From Theorems 8 and 9 we obtain an asymptotically tight bound for general λ-TS games.

▶ **Corollary 10.** *The price of anarchy in λ-TS games is $\Theta(\lambda/\tau)$.*

Theorem 9 allows us to provide concrete bounds for subclasses of λ-TS games. In particular, for λ-ZTS games, since $\tau_\ell = 1$ for every $\ell \in [\lambda]$, we have $\sum_{\ell\in[\lambda]} \tau_\ell = \lambda$, and thus the left-hand-side of the inequality in Theorem 9 allows us to improve upon the weaker lower of [16] and obtain the following tight bound, for any values of n and λ.

▶ **Corollary 11.** *The price of anarchy of λ-ZTS games is $\frac{\lambda n}{n-\lambda}$.*

We now define the following two natural classes of λ-TS games in which the tolerance parameters are specific functions of the distance between the types. In the first one, the difference of the tolerance level is proportional to the type distance, while in the other, the difference of the tolerance is decreasing in the type distance in an inversely proportional way.

- *Proportional* λ-TS games: $t_d = 1 - \frac{d}{\lambda-1}$ for each $d \in \{0, \dots, \lambda-1\}$, while $\tau = \sum_{\ell\in[\lambda]} \frac{\ell-1}{\lambda-1} = \frac{\lambda}{2}$.
- *Inversely proportional* λ-TS games: $t_d = \frac{1}{d+1}$ for every $d \in \{0, \dots, \lambda - 1\}$. We have $\tau = \sum_{\ell\in[\lambda]} \frac{1}{\ell} = H_\lambda$, where H_λ is the λ-th harmonic number.

By Theorems 8, 9 and the above definitions, we obtain the following corollaries.

▶ **Corollary 12.** *For every $\lambda \geq 2$, the price of anarchy of proportional λ-TS games is at most $\frac{2n}{n-2}$ and at least $\frac{\lambda n}{(\lambda-1)n-\frac{\lambda}{\lambda-1}}$.*

▶ **Corollary 13.** *For every $\lambda \geq 2$, the price of anarchy of inversely proportional λ-TS games is at most $\frac{\lambda n}{H_\lambda n-\lambda}$ and at least $\frac{\lambda n}{\frac{2(\lambda-1)}{\lambda} H_\lambda n-\frac{\lambda^2}{\lambda-1}+2H_\lambda}$.*

We conclude our technical contribution with a lower bound on the price of stability for the case of two types of agents. For 2-ZTS games, the following lower bound improves upon the bound of 34/33 of Elkind et al. [16], and is also tight when the number of agents tends to infinity because of the upper bound implied by Theorem 8; recall that $\tau = 1$ for λ-ZTS games.

▶ **Theorem 14.** *The price of stability of 2-TS games is at least $2/\tau - \epsilon$, for any $\epsilon > 0$.*

5 Open Problems

The most important question that our work leaves open is the characterization of games for which equilibria always exist. As this is a quite general and challenging direction, one could start with games that exhibit some structure in terms of the topology or the tolerance vector. For instance, do equilibria exist when the topology is a grid (4-grid or 8-grid) or a regular graph, for *every* tolerance vector?

The tolerance model we defined in this paper depends on a given ordering of the types and the tolerance parameters are symmetric. While this model captures certain interesting settings, there are multiple ways in which it can be generalized. For example, the tolerance parameters do not need to be symmetric and a different tolerance vector could be defined per type. Taking this further, the tolerance between types does not need to depend on an ordering of the types. Instead, one could define a weighted, directed *tolerance graph* that is defined over the different types such that the edge weights indicate the tolerance of a type towards another type; our ordered model can be thought of as the special case with an undirected tolerance line graph. In fact, one could further generalize this idea by considering scenarios in which there are no types of agents at all, but rather the agents are connected to each other via a complete weighted social network, with the different weights indicating tolerance levels. This is essentially a generalization of the class of social Schelling games proposed by Elkind et al. [16], and is inspired by fractional hedonic games [3].

References

1 Aishwarya Agarwal, Edith Elkind, Jiarui Gan, and Alexandros A. Voudouris. Swap stability in Schelling games on graphs. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

2 Elliot Anshelevich, Anirban Dasgupta, Jon M. Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

3 Haris Aziz, Florian Brandl, Felix Brandt, Paul Harrenstein, Martin Olsen, and Dominik Peters. Fractional hedonic games. *ACM Transactions on Economics and Computation*, 7(2):6:1–6:29, 2019.

4 George Barmpalias, Richard Elwes, and Andrew Lewis-Pye. Digital morphogenesis via Schelling segregation. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 156–165, 2014.

5 Patrick Bayer, Robert McMillan, and Kim Rueben. The causes and consequences of residential segregation: An equilibrium analysis of neighborhood sorting, 2001.

6 Prateek Bhakta, Sarah Miracle, and Dana Randall. Clustering and mixing times for segregation models on $\mathbb{Z}^2$. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 327–340, 2014.

7 Davide Bilò, Vittorio Bilò, Pascal Lenzner, and Louise Molitor. Topological influence and locality in swap Schelling games. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 15:1–15:15, 2020.

8 Thomas Bläsius, Tobias Friedrich, Martin S. Krejca, and Louise Molitor. The impact of geometry on monochrome regions in the flip schelling process. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC)*, pages 29:1–29:17, 2021.

9 Christina Brandt, Nicole Immorlica, Gautam Kamath, and Robert Kleinberg. An analysis of one-dimensional Schelling segregation. In *Proceedings of the 44th Symposium on Theory of Computing (STOC)*, pages 789–804, 2012.

10 Martin Bullinger, Warut Suksompong, and Alexandros A. Voudouris. Welfare guarantees in Schelling segregation. *Journal of Artificial Intelligence Research*, 71:143–174, 2021.

11 Hau Chan, Mohammad T. Irfan, and Cuong Viet Than. Schelling models with localized social influence: A game-theoretic framework. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 240–248, 2020.

12 Ankit Chauhan, Pascal Lenzner, and Louise Molitor. Schelling segregation with strategic agents. In *Proceedings of the 11th International Symposium on Algorithmic Game Theory (SAGT)*, pages 137–149, 2018.

13 William Clark and Mark Fossett. Understanding the social context of the Schelling segregation model. *Proceedings of the National Academy of Sciences*, 105(11):4109–4114, 2008.

14 Argyrios Deligkas, Eduard Eiben, and Tiger-Lily Goldsmith. The parameterized complexity of welfare guarantees in schelling segregation. *CoRR*, abs/2201.06904, 2022. `arXiv:2201.06904`.

15 Hagen Echzell, Tobias Friedrich, Pascal Lenzner, Louise Molitor, Marcus Pappik, Friedrich Schöne, Fabian Sommer, and David Stangl. Convergence and hardness of strategic Schelling segregation. In *Proceedings of the 15th International Conference on Web and Internet Economics (WINE)*, pages 156–170, 2019.

16 Edith Elkind, Jiarui Gan, Ayumi Igarashi, Warut Suksompong, and Alexandros A. Voudouris. Schelling games on graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 266–272, 2019.

17 Nicole Immorlica, Robert Kleinberg, Brendan Lucier, and Morteza Zadimoghaddam. Exponential segregation in a two-dimensional Schelling model with tolerant individuals. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 984–993, 2017.

18 Panagiotis Kanellopoulos, Maria Kyropoulou, and Alexandros A. Voudouris. Modified Schelling games. *Theoretical Computer Science*, 880:1–19, 2021.

19 Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 404–413, 1999.

20 Romans Pancs and Nicolaas J. Vriend. Schelling's spatial proximity model of segregation revisited. *Journal of Public Economics*, 91(1–2):1–24, 2007.

21 Thomas C. Schelling. Models of segregation. *American Economic Review*, 59(2):488–493, 1969.

22 Thomas C. Schelling. Dynamic models of segregation. *Journal of Mathematical Sociology*, 1(2):143–186, 1971.

23 Charles M. Tiebout. A pure theory of local expenditures. *Journal of Political Economy*, 64(5):416–424, 1956.

24 Dejan Vinković and Alan Kirman. A physical analogue of the schelling model. *Proceedings of the National Academy of Sciences*, 103(51):19261–19265, 2006.

25 Junfu Zhang. Residential segregation in an all-integrationist world. *Journal of Economic Behavior and Organization*, 54(4):533–550, 2004.

On the Skolem Problem for Reversible Sequences

George Kenison ✉
Institute of Logic and Computation, TU Wien, Vienna, Austria

Abstract

Given an integer linear recurrence sequence $\langle X_n\rangle_{n=0}^{\infty}$, the Skolem Problem asks to determine whether there is an $n \in \mathbb{N}_0$ such that $X_n = 0$. Recent work by Lipton, Luca, Nieuwveld, Ouaknine, Purser, and Worrell proved that the Skolem Problem is decidable for a class of reversible sequences of order at most seven. Here we give an alternative proof of their result. Our novel approach employs a powerful result for Galois conjugates that lie on two concentric circles due to Dubickas and Smyth.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Computing methodologies → Algebraic algorithms

Keywords and phrases The Skolem Problem, Linear Recurrences, Verification

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.61

Funding Work supported by WWTF ICT19-018 grant *ProbInG* and ERC Consolidator Grant *ARTIST* 101002685.

Acknowledgements I am grateful to both Adam Keilthy and Florian Luca for their feedback and many helpful discussions. I also thank the anonymous reviewers for their constructive feedback.

1 Introduction

The Skolem Problem

An integer-valued *linear recurrence sequence* $\langle X_n\rangle_{n=0}^{\infty}$ satisfies a relation of the form

$$X_{n+d} = a_{d-1}X_{n+d-1} + \cdots + a_1X_{n+1} + a_0X_n \tag{1}$$

for each $n \in \mathbb{N}_0$. Without loss of generality, we shall assume that each of the coefficients $a_0, a_1, \ldots, a_{d-1} \in \mathbb{Z}$ and additionally that $a_0 \neq 0$. We call d the *length* of the recurrence relation and the *order* of $\langle X_n\rangle_n$ is the length of the shortest relation satisfied by $\langle X_n\rangle_n$. The polynomial $f(x) = x^d - a_{d-1}x^{d-1} - \cdots - a_1x - a_0$ is the *characteristic polynomial* associated with relation (1). Given such a sequence, the *Skolem Problem* [8, 11] asks to determine whether there exists an $n \in \mathbb{N}$ such $X_n = 0$. The Skolem Problem is well-motivated with connections to research topics such as program verification [27]. Take, for example, the following linear loop P with inputs $\underline{w}, \underline{b} \in \mathbb{Z}^d$ and $A \in \mathbb{Z}^{d\times d}$ where

$$P\colon \underline{v} \leftarrow \underline{w};\ \ \textbf{while}\ \ \underline{b}^{\top}\underline{v} \neq 0\ \ \textbf{do}\ \ \underline{v} \leftarrow A\underline{v}. \tag{2}$$

Let $\langle X_n\rangle_n$ be the linear recurrence sequence with terms given by $X_n = \underline{b}^{\top}A^n\underline{w}$. It is clear that loop P terminates if and only if there exists an $n \in \mathbb{N}_0$ such that $X_n = 0$.

Motivation

A recent resurgence of interest in the Skolem Problem (and related problems) has lead to the publication of a number of papers that consider restricted variants. The resulting specialised decision procedures generally fall into two categories: those that consider an infinite subset of the natural numbers [14, 20] or those that restrict the class of linear recurrence sequences. Our motivation is the latter type of specialisation and, in particular, a recent paper by Lipton et al. [19] that establishes the following theorem.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 61; pp. 61:1–61:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Theorem 1.** *The Skolem Problem is decidable for the class of reversible integer linear recurrences of order at most seven.*

An integer linear recurrence sequence $\langle X_n \rangle_{n=0}^{\infty}$ is *reversible* if it satisfies a recurrence relation of the form (1) such that $a_0 = \pm 1$. As observed in Lipton et al. [19], given an integer linear recurrence sequence $\langle X_n \rangle_{n=0}^{\infty}$, the unique bi-infinite extension $\langle X_n \rangle_{n=-\infty}^{n=\infty}$ has integral terms if and only if $\langle X_n \rangle_{n=0}^{\infty}$ is reversible (this claim follows from a classical observation for Fatou rings [9]). We can also characterise the subclass of while loops (as in (2)) naturally associated with reversible sequences: the update matrix A with characteristic polynomial f is *unimodular*; that is, A has integer entries and $\det(A) = -f(0) = \pm 1$. If A is unimodular, then A^{-1} also has integer entries. Thus, again, $\langle X_n \rangle_{n=-\infty}^{n=\infty}$ with each $X_n = \underline{b}^{\top} A^n \underline{w}$ (as above) is integer-valued.

Unimodular matrices appear elsewhere in the dynamical systems literature. Some classes lead to prototypical invertible maps with hyperbolic and ergodic properties; specifically, classes of *linear toral automorphisms* $T_A \colon \mathbb{R}^d \to \mathbb{R}^d$ given by $T_A(\underline{x}) = A\underline{x}$ [13].

A famous example of a reversible sequence is the Fibonacci sequence, which is defined by the initial values $X_0 = 0$, $X_1 = 1$ and for each $n \in \mathbb{N}_0$, $X_{n+2} = X_{n+1} + X_n$. The Fibonacci sequence can be uniquely extended to a bi-infinite sequence of integer values $\langle \ldots, 5, -3, 2, -1, 1, 0, 1, 1, 2, 3, 5, \ldots \rangle$.

In the sequel, we call the restricted variant of the Skolem Problem for reversible sequences the *Reversible Skolem Problem*.

Background

Let us assess the current state of play with regards to the decidability of the Skolem Problem. A classical result due to Skolem [31] (which was later generalised by Mahler [22, 23], and Lech [18]) states that $\{n \in \mathbb{N} : X_n = 0\}$ is the union of a finite set together with a finite number of (infinite) arithmetic progressions. The phenomenon that causes these vanishing arithmetic progressions is termed *degeneracy*. A sequence is *degenerate* when one of the ratios of two distinct characteristic roots of the sequence is a root of unity. These arithmetic progressions can be determined algorithmically and so, from the viewpoint of verification, to decide the Skolem Problem it suffices to consider *non-degenerate* recurrence sequences – those sequences that have only finitely many zeros. Indeed, this is where the difficulty lies: there is no known general method to compute this finite set. We refer the interested reader to Corollary 1.20 and Chapter 2 in [8] for further details. In summary, all known proofs of the Skolem–Mahler–Lech Theorem (as it is now known) are non-constructive and so the decidability of the Skolem Problem remains open.

Limited progress has been made on the decidability of the Skolem Problem when one considers linear recurrence sequences of low order. Groundbreaking work by Mignotte, Shorey, and Tijdeman [24], and, independently, Vereshchagin [35] establish the following.

▶ **Theorem 2.** *The Skolem Problem is decidable for the class of non-degenerate linear recurrences with at most three simple characteristic roots that are maximal in modulus.*

As a consequence, the Skolem Problem is decidable for linear recurrences of order at most four. The aforementioned papers employ techniques from p-adic analysis and algebraic number theory and, in addition, Baker's theorem for linear forms in logarithms of algebraic numbers. Unfortunately the route taken via Baker's Theorem does not appear to extend easily to recurrences of higher order.

A class of recurrence sequences of order five that the state of the art cannot handle impedes further progress on the decidability of the Skolem Problem [25]. The minimal polynomial for each member of this class has four distinct roots $\alpha, \overline{\alpha}, \beta, \overline{\beta} \in \mathbb{C}$ (two pairs of complex-conjugate roots) such that $|\alpha| = |\beta|$, and a fifth real root γ of strictly smaller modulus. Hence the terms of such a sequence $\langle X_n \rangle_n$ are given by an exponential polynomial of the form $X_n = a(\alpha^n + \overline{\alpha}^n) + b(\beta + \overline{\beta}^n) + c\gamma^n$. Here $a, b, c \in \mathbb{R}$ are algebraic numbers and, as far as we are aware, there is no known general procedure to determine $\{n \in \mathbb{N} : X_n = 0\}$ when $|a| \neq |b|$.

Complexity

In [3], Blondel and Portier proved that the Skolem Problem is NP-hard. As a brief aside, let us consider the complexity of the reversible variant of the Skolem Problem. One of the questions considered by S. Akshay et al. [1] is the complexity of the Skolem Problem for the restricted class of linear recurrence sequences whose characteristic roots are all roots of unity (the so-called *Cyclotomic Skolem Problem*). Those authors showed, by a reduction from the Subset-Sum Problem, that the Cyclotomic Skolem Problem is NP-hard. Because the characteristic polynomial associated with each element in this restricted class is given by a product of cyclotomic polynomials, it follows that each instance of the Cyclotomic Skolem Problem is an instance of the Reversible Skolem Problem. Thus the Reversible Skolem Problem is also NP-hard.

Contributions

The main contribution in this note is an alternative proof of Theorem 1. By comparison to the extensive case analysis employed in [19], we use results in number theory for Galois conjugates that obey polynomial identities. In particular, we make repeated use of a result due to Dubickas and Smyth [7] for algebraic integers that lie alongside all their Galois conjugates on two (but not one) concentric circles centred at the origin.

For context, Dubickas and Smyth's result is part of a large corpus of research on algebraic numbers whose conjugates lie on a conic or a union of conics. Let α be an algebraic number with Galois conjugates $\alpha = \alpha_1, \ldots, \alpha_d$. We call the set $S(\alpha) := \{\alpha_1, \ldots, \alpha_d\}$ the *conjugate set* of α. When $S(\alpha)$ is a subset of the unit circle, a result of Kronecker's (a weaker version of Theorem 5) proves that α is a root of unity. Number theorists have long-studied classes of algebraic integers where one or more of the conjugates leaves the circle; for example, a real algebraic integer α is a *Salem number* if $\alpha > 1$, $\alpha^{-1} \in S(\alpha)$, and the remaining conjugates all lie on the unit circle, i.e., $S(\alpha) = \{\alpha^{\pm 1}, \mathrm{e}^{\pm \mathrm{i}\theta_2}, \ldots, \mathrm{e}^{\pm \mathrm{i}\theta_d}\}$.

With regards to the Reversible Skolem Problem at order eight, we exhibit a family of recurrence sequences that, as far as we know, are not amenable to standard techniques and so decidability is very much open. The authors of [19] also demonstrated a concrete family of examples in this regard. It is interesting to note that the techniques used and families obtained are very different. Each member of our family is an octic palindromic polynomial (the same is not true for the family of examples in [19]). The coefficients of a *palindromic* polynomial form a palindromic string of integers. For example, two palindromes that are also family members are $x^8+x^7-x^6+x^5+5x^4+x^3-x^2+x+1$ and $x^8+x^7-3x^6+x^5+9x^4+x^3-3x^2+x+1$. Using any modern computer algebra system, it is easy to verify that the roots of each polynomial satisfy the following. First, the roots lie on two (but not one) concentric circles centred at the origin. Second, no ratio of any two of its roots is a root of unity. The calculations involved in preparing these (and later) examples were performed in PARI/GP [34]. In Subsection 4.2, we study the Galois groups of irreducible palindromic octics in the aforementioned family as a further investigation into the symmetries between their roots.

In Subsections 5.1 and 5.2, we prove new decidability results on restricted variants of the Positivity and Skolem Problems using Dubickas and Smyth's theorem and make suggestions for further work in these directions. For brevity, we refer to the Positivity Problem for the class of simple reversible integer linear recurrences as the *Simple Reversible Positivity Problem*. Here a linear recurrence is *simple* if the associated characteristic polynomial has no repeated roots. We have the following results:

▶ **Corollary 3.** *The Simple Reversible Positivity Problem is decidable for integer-valued linear recurrences of order at most ten.*

▶ **Corollary 4.** *The Skolem Problem is decidable for rational-valued linear recurrences that satisfy a relation of the form $X_{n+5} = a_4X_{n+4} + a_3X_{n+3} + a_2X_{n+2} + a_1X_{n+1} \pm X_n$ with $a_1, a_2, a_3, a_4 \in \mathbb{Q}$.*

Structure

The remainder of this paper is structured as follows. In the next section we review necessary preliminary material. In Section 3, we give a new and novel proof of Theorem 1. In Section 4, we construct a family of octic palindromes that shows the current state of the art cannot settle decidability of the Reversible Skolem Problem at order eight and then discuss the Galois groups associated with the irreducible members of this family. In the final section, Section 5, we discuss directions and motivate this discussion with the proofs of Corollaries 3 and 4.

2 Preliminaries

2.1 Recurrence Sequences

A sequence $\langle X_n\rangle_{n=0}^{\infty}$ of integers satisfying a recurrence relation of the form (1) with fixed integer constants $a_0, a_1, \dots, a_{d-1}$ such that $a_0 \neq 0$ is a *linear recurrence sequence*. The sequence $\langle X_n\rangle_n$ is then wholly determined by the recurrence relation and the initial values $X_0, X_1, \dots, X_{d-1}$. The polynomial $f(x) = x^d - a_{d-1}x^{d-1} - \cdots - a_1x - a_0$ is the *characteristic polynomial* associated with relation (1). From our earlier definition, it is clear that $\langle X_n\rangle_n$ is reversible if and only if $f(0) = \pm 1$. There is a recurrence relation of minimal length associated to $\langle X_n\rangle_n$ and we call the characteristic polynomial of this minimal length relation the *minimal polynomial* of $\langle X_n\rangle_n$. The *order* of a linear recurrence sequence is the degree of its minimal polynomial.

Let f be the minimal polynomial of a linear recurrence sequence $\langle X_n\rangle_n$ and K the splitting field of f. The polynomial f factorises as a product of powers of distinct linear factors like so $f(x) = \prod_{\ell=1}^{m}(x - \lambda_\ell)^{n_\ell}$. The constants $\lambda_1, \lambda_2, \dots, \lambda_m \in K$ are the *characteristic roots* of $\langle X_n\rangle_n$ with multiplicities $n_1, n_2, \dots, n_m$. One can realise the terms of a linear recurrence sequence as an *exponential polynomial* $X_n = \sum_{\ell=1}^{m} p_\ell(n)\lambda_\ell^n$ where the λ_ℓ are the aforementioned characteristic roots of $\langle X_n\rangle_n$ and the polynomial coefficients $p_\ell \in K[x]$ are determined by the initial values. We say a characteristic root of $\langle X_n\rangle_n$ is *dominant* if in the set of characteristic roots of $\langle X_n\rangle_n$ it is maximal in modulus. Thus, by Theorem 2, the Skolem Problem is decidable for the class of non-degenerate linear recurrence sequences with at most three dominant characteristic roots [24, 35].

2.2 Number Theory

We shall assume some familiarity with Galois theory and the theory of number fields. The necessary background material can be found in a number of standard textbooks [5, 33].

Recall the following theorem due to Kronecker [16].

▶ **Theorem 5.** *Let $f \in \mathbb{Z}[x]$ be a monic polynomial such that $f(0) \neq 0$. Suppose that all the roots of f have absolute value at most 1, then f is a product of cyclotomic polynomials. Therefore all the roots of f are roots of unity.*

Thus, if $f \in \mathbb{Z}[x]$ is the characteristic polynomial of a reversible linear recurrence sequence such that the roots of f all lie in the unit disk $\{z \in \mathbb{C} : |z| \leq 1\}$. Then the roots of f are all roots of unity. It follows that the associated recurrence sequence is either order one (and is thus constant) or degenerate. In either case the Skolem Problem is decidable. Thus in the sequel we shall always assume, without loss of generality, that the dominant roots of f lie on a circle with radius strictly larger than 1.

In the sequel, our construction of an infinite family of octics uses the following corollary of Vieta's formulae.

▶ **Lemma 6.** *Suppose that $f \in \mathbb{Z}[x]$ is a monic irreducible polynomial such that $f(x) = \prod_{i=1}^{d}(x - \lambda_i)$. Let $f_n(x) := \prod_{i=1}^{d}(x - \lambda_i^n)$. Then $f_n \in \mathbb{Z}[x]$ for each $n \in \mathbb{N}$.*

Proof. The coefficients of the polynomial f_n are determined by symmetric polynomials in d variables. By the fundamental theorem of symmetric polynomials, each symmetric polynomial is given by a $\mathbb{Z}$-linear combination of elementary symmetric polynomials. The result follows as a straightforward application of Vieta's formulae and the evaluation of elementary symmetric polynomials over conjugate algebraic integers. ◀

The roots of an irreducible polynomial are necessarily Galois conjugates. We use the term *conjugate ratios* for the ratios between two distinct roots of an irreducible polynomial. A non-zero algebraic number α is *reciprocal* if α is conjugate to α^{-1}. Let τ be a Salem number whose minimal polynomial has degree $2d$ or a reciprocal quadratic. In the former case, $S(\tau) = \{\tau^{\pm 1}, \tau_2^{\pm 1}, \ldots, \tau_d^{\pm 1}\}$ and in the latter, $S(\tau) = \{\tau^{\pm 1}\}$. An algebraic number ψ is a *Salem half-norm* if $\psi = \tau^{\varepsilon_1}\tau_2^{\varepsilon_2} \cdots \tau_d^{\varepsilon_d}$ for some such τ and $\varepsilon_j = \pm 1$ for each $j \in \{1, \ldots, d\}$. The properties of Salem half-norms are discussed further in [7].

In the ring of algebraic integers of a given number field K, $\gamma \in K$ is a *unit* (sometimes an *algebraic unit*) if it has a multiplicative inverse δ so that $\gamma\delta = \delta\gamma = 1$. Let $\alpha \in K$ be an algebraic integer. Then the constant coefficient of the minimal polynomial $f \in \mathbb{Z}[x]$ of α is equal to ± 1 if and only if α is a unit. This observation follows easily from norm considerations and the fact that the constant coefficient of f (up to sign) is given by the product of α and its Galois conjugates. Since the characteristic polynomial $f \in \mathbb{Z}[x]$ of a reversible sequence $\langle X_n \rangle_n$ has constant coefficient ± 1, we deduce that each of the characteristic roots of $\langle X_n \rangle_n$ is a unit. In the sequel we make frequent use of the following simple observation.

▶ **Lemma 7.** *If α is an algebraic unit that lies on the circle $|z| = R$ with $R > 1$, then a conjugate of α lies in the interior of the unit disk.*

Key to the proofs in the sequel is a powerful result due to Dubickas and Smyth [7] concerning polynomial identities between the roots of irreducible polynomials. Theorem 8 gives necessary conditions for a unit in the algebraic integers and all its Galois conjugates to lie on two (but not one) concentric circles centred at the origin [7]. In fact, Dubickas and Smyth prove a far more general result [7, Theorem 2.1], but we need only the specialised version for units below.

▶ **Theorem 8.** *Suppose that α is a unit in the algebraic integers of degree d lying, with all its Galois conjugates, on two circles $|z| = r$ and $|z| = R$, but not just one. Without loss of generality assume that at most half of the conjugates lie on $|z| = r$. Then, one of the following holds:*

1. *$d = 3m$, $R = r^{-1/2}$ such that there are $d/3$ conjugates of α on $|z| = r$, and the remaining $2d/3$ lie on $|z| = r^{-1/2}$. Assume, without loss of generality, $|\alpha| = r$. Then, in addition, there exists an $n \in \mathbb{N}$ such that α^n is a real, but non-totally real, cubic unit.*
2. *$d = 2m$, $R = r^{-1}$ where $R > 1$ without loss of generality, and $d/2$ Galois conjugates of α lie on each circle. Further, there exists an $n \in \mathbb{N}$ such that $\alpha^n =: \psi$ is a Salem half-norm defined by a Salem number or a reciprocal quadratic.*

Let us explain the term *totally real* in the last theorem. An algebraic number α is *totally real* if α and all its Galois conjugates are real. A number field K is *totally real* if $K = \mathbb{Q}[\alpha]$ such that α is totally real.

From this point to the end of the subsection, the terminology and results we recall are used only in Section 4 (and so are not required for the proof of Theorem 1).

A field is *Kroneckerian* if it is either a totally real algebraic number field or a totally imaginary quadratic extension of a totally real field. In the sequel, we make use of the following observation about complex conjugation lying in the centre of the Galois group of a Kroneckerian field (see [30, Chapter 6]).

▶ **Corollary 9.** *A number field K is Kroneckerian if and only if for every $\alpha \in K$ one has $\overline{\alpha} \in K$ and for every embedding σ of K into $\mathbb{C}$ one has $\overline{\alpha^\sigma} = \overline{\alpha}^\sigma$.*

A unit is *unimodular* if it lies on the unit circle in $\mathbb{C}$. In [21], MacCluer and Parry prove that a normal imaginary field contains unimodular units other than roots of unity exactly when its real subfield is not normal over $\mathbb{Q}$. Daileda [6] generalises this result and provides the following classification of the number fields that have unimodular units that are not roots of unity. Recall that a number field K is a *CM-field* if K is a totally complex quadratic extension of a totally real field.

▶ **Theorem 10.** *Let K be a number field closed under complex conjugation. Then K contains unimodular units that are not roots of unity if and only if K is imaginary and not a CM-field.*

2.3 Group Theory

In the sequel we employ the notation S_n for the symmetric group on n elements, A_n for the alternating group on n elements, D_n for the Dihedral group of order $2n$, C_n for the cyclic group on n elements, and K_4 for the Klein 4-group.

The action of G on a set X is *transitive* if for every pair $x, y \in X$ there is a $g \in G$ such that $gx = y$; that is to say, there is a single group orbit. We note here (and again later) that S_4, A_4, D_4, C_4, and K_4 are the transitive subgroups of S_4.

3 Proof of Theorem 1

We briefly outline our route to proving Theorem 1. We claim that if $\langle X_n \rangle_n$ is a non-degenerate reversible integer recurrence sequence of order at most seven, then $\langle X_n \rangle_n$ has at most three dominant characteristic roots. The decidability of the Skolem Problem for such instances then follows from Theorem 2. The above claim follows as a corollary of the next theorem.

▶ **Theorem 11.** *No monic polynomial $f \in \mathbb{Z}[x]$ with constant coefficient ± 1 of degree at most seven satisfies the following two properties:*

(H1) *f has at least four distinct dominant roots; and*

(H2) *no quotient of two distinct roots of f is a root of unity.*

Thus all that remains is to prove Theorem 11. This result is an immediate consequence of the sequence of Propositions 12, 14, and 15 below. In each of the proofs of these propositions we play a similar game: we assume, for a contradiction, that there exists a polynomial $f \in \mathbb{Z}[x]$ (of degree five, six, or seven respectively) that satisfies hypotheses H1 and H2. We show that such a candidate is necessarily irreducible. We then employ Theorem 8 to derive a contradiction: such a candidate cannot satisfy both H1 and H2 and, at the same time, satisfy the restrictive root identities prescribed by Theorem 8.

For the avoidance of doubt, this route to Theorem 1 is similar to that carved out by [19]. The contribution of this paper is the novel application of Theorem 8. Indeed, our assumption, that each of the characteristic roots of a recurrence in our class of non-degenerate reversible sequences is a unit, leads to (rather startling) restrictive polynomial relations between Galois conjugates.

We begin our sequence of propositions.

▶ **Proposition 12.** *No monic polynomial $f \in \mathbb{Z}[x]$ of degree at most five with constant term ± 1 satisfies hypotheses H1 and H2.*

Proof. Assume, for a contradiction, that such an f of degree $d \le 5$ exists. By Theorem 5, the dominant roots lie on the circle $|z| = R$ for some $R > 1$ for otherwise the roots of f are necessarily all roots of unity, which is not permitted under hypothesis H2. Each of the dominant roots of f is a unit in the algebraic integers and so, by Lemma 7, has a Galois conjugate in the interior of the unit disk. In order that f satisfies hypothesis H1, we conclude that f has four simple dominant roots and a single non-dominant root. Since f has degree $d = 5$ and $f(0) = \pm 1$, all the dominant roots of f are (Galois) conjugate to the single non-dominant root, f is irreducible.

Let α be a root of f. Then α is an algebraic integer of degree 5, a unit, and lies with all its Galois conjugates on two circles. We apply Theorem 8 to the quintic f and find that 5 is either even, or a multiple of 3, a contradiction. ◀

▶ Remark 13. The application of Theorem 8 in the proof of Proposition 12 is excessive (even if the derived contradiction is rather satisfying). By comparison, the approach in Lipton et al. [19] is direct. We reproduce the final part of those authors' proof below as it gives a gentle introduction to some of the techniques we apply in Proposition 14 and Proposition 15.

Proof of Proposition 12 (cf. [19]). Let $\alpha, \overline{\alpha}, \beta, \overline{\beta}$ be the four dominant roots of f and ρ the non-dominant root of f. Since f is irreducible, the Galois group G of f acts transitively on the roots of f. Thus there exists a $\sigma \in G$ such that $\sigma(\alpha) = \rho$. The element σ must preserve the equality $\alpha\overline{\alpha} = \beta\overline{\beta}$ and so we have $\rho\sigma(\overline{\alpha}) = \sigma(\beta)\sigma(\overline{\beta})$. We derive a contradiction: $|\rho||\sigma(\overline{\alpha})| \neq |\sigma(\beta)||\sigma(\overline{\beta})|$ since the two roots $\sigma(\beta)$ and $\sigma(\overline{\beta})$ on the right-hand side are necessarily dominant. ◀

▶ **Proposition 14.** *No monic polynomial $f \in \mathbb{Z}[x]$ of degree six with constant term ± 1 satisfies hypotheses H1 and H2.*

Proof. Assume, for a contradiction, that such an f exists. As in the proof of Proposition 12, f has at least four simple dominant roots $\alpha, \overline{\alpha}, \beta, \overline{\beta}$ that lie on a circle $\{z \in \mathbb{C} : |z| = R\}$ for some $R > 1$ as complex-conjugate pairs. We note that the group G of automorphisms of the splitting field of f must preserve the equality $\alpha\overline{\alpha} = \beta\overline{\beta}$. Because α is a unit on $|z| = R$, by Lemma 7 there is both a root γ of f that lies in the unit disk and a permutation $\sigma \in G$ such that $\sigma(\alpha) = \gamma$. Now consider $\sigma(\alpha)\sigma(\overline{\alpha}) = \sigma(\beta)\sigma(\overline{\beta})$. It is straightforward to elicit a contradiction that breaks this equality if either $\sigma(\overline{\alpha})$ is non-dominant, or both $\sigma(\beta)$ and $\sigma(\overline{\beta})$ are dominant. Thus we can assume that f has two non-dominant roots and further that they are of equal modulus. Clearly these two roots $\gamma, \overline{\gamma}$ are a complex-conjugate pair by H2. Combining these observations of the roots of f, we quickly deduce that f is necessarily irreducible.

Because the roots of the irreducible polynomial $f \in \mathbb{Z}[x]$ lie on two concentric circles, we can apply Theorem 8. Thus there is a non-dominant root, γ say, and $m \in \mathbb{N}$ such that γ^m is a real cubic unit. It follows that $\gamma^m/\overline{\gamma}^m = 1$ and so one of the conjugate ratios of f is a root of unity, which contradicts hypothesis H2. ◀

▶ **Proposition 15.** *No monic polynomial $f \in \mathbb{Z}[x]$ of degree seven with constant term ± 1 satisfies hypotheses H1 and H2.*

Proof. Assume, for a contradiction, that such an f exists. As in the proof of Proposition 12, f has at least four simple dominant roots $\alpha, \overline{\alpha}, \beta, \overline{\beta}$ that lie on a circle $\{z \in \mathbb{C} : |z| = R\}$ for some $R > 1$ as complex-conjugate pairs. Mutatis mutandis, one can use the methods in the proof of Proposition 14 to make the following two deductions. First, f is irreducible. Second, f has precisely one real root δ and another complex-conjugate pair of roots $\gamma, \overline{\gamma}$. Additionally, we have that $\delta, \gamma, \overline{\gamma}$ are all non-dominant roots of f such that $|\gamma| \neq |\delta|$. We note that if one supposes, for a contradiction, that the septic polynomial f has $|\gamma| = |\delta|$ or $|\alpha| = |\delta|$, then, by Theorem 8, it follows that 7 is either even, or a multiple of 3.

Because f is irreducible, the Galois group of f acts transitively on the roots of f. Thus there is a permutation σ in the Galois group of f such that $\sigma(\alpha) = \delta$. We know that σ must preserve the equality $\alpha\overline{\alpha} = \beta\overline{\beta}$ and so we have $|\delta\sigma(\overline{\alpha})| = |\sigma(\beta)\sigma(\overline{\beta})|$. The left-hand side is equal to one of $|\delta||\gamma|$ or $|\delta|R$ depending on $\sigma(\overline{\alpha})$. There are three cases to consider for the right-hand side. The roots $\sigma(\beta)$ and $\sigma(\overline{\beta})$ are either both dominant, both non-dominant, or one of each. It is clear that the modulus in each of these (respective) cases R^2, $|\gamma|^2$, and $|\gamma|R$ breaks the aforementioned equality between the left- and right-hand sides, a contradiction. ◀

Hence we have proved Theorem 11, as required.

4 Palindromic Octics

A monic polynomial $f \in \mathbb{Z}[x]$ is *palindromic* if its coefficients string together to form a palindrome. That is to say, for $f(x) = x^d + a_{d-1}x^{d-1} + \cdots + a_1x + 1$ we have that $a_k = a_{d-k}$ for each $k \in \{0, 1, \ldots, d\}$. In other sources, the term *self-reciprocal* is sometimes used since if α is a root of f then α^{-1} is also a root of f. If $\alpha \in \mathbb{C}$ is a root of f, then so are $\bar{\alpha}, 1/\alpha$, and $1/\bar{\alpha}$. Further, when α is neither real nor lies on the unit circle then these four roots are distinct. The class of palindromic polynomials appear in areas across mathematics and computer science in fields such as: coding theory, algebraic curves over finite fields, knot theory, and linear feedback shift registers, to name but a few (see the survey [12]).

▶ Remark 16. Let us further motivate the study of the *Palindromic Skolem Problem* (the Skolem Problem restricted to the study of sequences with palindromic characteristic polynomials). From the viewpoint of dynamical systems, we observe that the equations of motion (1) governing a recurrence sequence with palindromic characteristic polynomial (hereafter a *palindromic recurrence sequence*) possess a *time-reversing symmetry*; that is to say, the recurrence relation is invariant under the time reversal map $n \mapsto -n$. More concretely, let $\langle X_n \rangle_{n=0}^{\infty}$ and $\langle Y_{-n} \rangle_{n=0}^{\infty}$ be recurrence sequences satisfying the palindromic relations

$$\begin{aligned} X_{n+d} &= a_{d-1}X_{n+d-1} + \cdots + a_{d-1}X_{n+1} + X_n \qquad \text{and} \\ Y_{-n-d} &= a_{d-1}Y_{-n-d+1} + \cdots + a_{d-1}Y_{-n-1} + Y_{-n}, \end{aligned}$$

respectively. Then $Y_{-n} = X_n$ for each $n \in \mathbb{N}_0$ if $Y_{-m} = X_m$ for each $m \in \{0, 1, \ldots, d-1\}$.

For further information on the topic of time-reversing symmetries, we refer the interested reader to the exposition in Lamb's article [17] and the recent survey by Baake [2] (and the references therein).

4.1 Hard instances of the Reversible Skolem Problem at order eight

In Proposition 17, we construct an infinite family of palindromic octics that satisfy both H1 and H2. This result blocks any obvious attempt to settle decidability of the Reversible Skolem Problem at order eight with current state-of-the-art techniques. Another family of octics that satisfy both H1 and H2 is discussed in Lipton et al. [19]. The additional restriction (palindromic coefficients) we require demonstrates a refinement in the discussion of hard instances of the Reversible Skolem Problem at order eight.

▶ **Proposition 17.** *There are infinitely many palindromic octics in $\mathbb{Z}[x]$ that satisfy both H1 and H2.*

Proof. Let $f \in \mathbb{Z}[x]$ be such a palindromic octic with four simple dominant roots (that is, two complex-conjugate pairs) that lie on a circle of radius $r > 1$. Write $f(x) = \prod_{i=1}^{8}(x - \lambda_i)$ and for each $n \in \mathbb{N}$, define $f_n(x) := \prod_{i=1}^{8}(x - \lambda_i^n)$. By the symmetries in the roots λ_i^n, it is straightforward to verify the following three observations. First, each octic f_n in the sequence $\langle f_n \rangle_{n=1}^{\infty}$ is palindromic. Second, each octic f_n satisfies both H1 and H2. Third, by Lemma 6, $f_n \in \mathbb{Z}[x]$ for each $n \in \mathbb{N}$.

We finish the proof by showing that there are infinitely many distinct polynomials in the sequence $\langle f_n \rangle_{n=1}^{\infty}$. Let $r\mathrm{e}^{\pm \mathrm{i}\theta}, r\mathrm{e}^{\pm \mathrm{i}\psi}$ be the four dominant roots of f. Let $f_n(x) = x^8 + a_{7,n}x^7 + \cdots + 1$. Then, by Vieta's formulae, $a_{7,n}$ is given by the sum of the nth powers of the roots of f; that is,

$$a_{7,n} = (r^n + r^{-n})(\mathrm{e}^{n\mathrm{i}\theta} + \mathrm{e}^{-n\mathrm{i}\theta} + \mathrm{e}^{n\mathrm{i}\psi} + \mathrm{e}^{-n\mathrm{i}\psi}).$$

Assume, for a contradiction, that the non-degenerate integer linear recurrence sequence $\langle a_{7,n} \rangle_n$ takes only finitely many values. By the Pigeonhole Principle, there is an $a \in \mathbb{Z}$ and an infinite subsequence $\langle n_k \rangle_k$ of natural numbers such that $a_{7,n_k} = a$ for each $k \in \mathbb{N}$. We make two observations. First, the integer linear recurrence sequence $\langle a_{7,n} - a \rangle_n$ has characteristic polynomial $f(x)(x-1)$ (see [8, Theorem 1.1]) and so is non-degenerate. Second, the sequence $\langle a_{7,n} - a \rangle_n$ vanishes infinitely often. We have a contradiction: a non-degenerate linear recurrence sequence has only finitely many zero terms. It follows that there are infinitely many distinct palindromic octics in the sequence $\langle f_n \rangle_n$, as desired. ◀

▶ Remark 18. A *symplectic matrix* M is a $2\ell \times 2\ell$ matrix that preserves the *symplectic form* J in 2ℓ dimensions; that is to say, $M^\top JM = J$ where

$$J = \begin{pmatrix} 0 & I_\ell \\ -I_\ell & 0 \end{pmatrix}.$$

It is well-known that the characteristic polynomial of a symplectic matrix is a palindromic polynomial. In fact, the following constructive result proves the converse. Rivin [28, Theorem A.1] attributes the result to Kirby [15][1].

▶ **Theorem 19.** *For each monic palindromic polynomial $f \in \mathbb{Z}[x]$ of degree 2ℓ, there is a symplectic matrix $M \in \mathbb{Z}^{2\ell \times 2\ell}$ such that $\det(xI_{2\ell} - M) = f(x)$.*

Note the group of $2\ell \times 2\ell$ symplectic matrices with entries in $\mathbb{Z}$ is closed under multiplication. This observation leads to an alternative proof of Proposition 17 as follows. Let $f \in \mathbb{Z}[x]$ be an octic palindrome satisfying H1 and H2. Now let $M \in \mathbb{Z}^{8\times 8}$ be a symplectic matrix with characteristic polynomial f. Consider the sequence $\langle f_n \rangle_n$ where $f_n(x) := \det(xI_{2\ell} - M^n) \in \mathbb{Z}[x]$. As before, we need to verify that each polynomial in this sequence satisfies the root assumptions H1 and H2. We then proceed in a similar fashion to the proof of Proposition 17, in order to generate an infinite sequence of octic palindromes with the desired properties.

4.2 Galois theory of octic palindromes

It is interesting to consider the root symmetries of the characteristic polynomials of reversible linear recurrence sequences. In this subsection we focus our attention on the Galois groups of octic palindromes; in particular, those underlying hard instances of the Reversible Skolem Problem. We present a new result (Theorem 20). The problem of root symmetries follows naturally from our approach to Theorem 1 where we explored polynomial identities between roots of certain irreducible polynomials. Such results are also motivated by hard open problems such as (polynomial) invariant generation and loop synthesis in the field of program verification.

The task of computing the Galois groups of irreducible polynomials in $\mathbb{Z}[x]$ is well-known. Families of polynomials associated with reversible sequences include the cyclotomic and Salem polynomials. The authors of [4] discuss ramifications to the Galois group of moving two of the Galois conjugates off of the unit circle. For the interested reader, two accounts of the Galois theory of palindromic polynomials are [36, 29]. 'Generically', the Galois group of a palindromic polynomial of degree $2d$ is $S_d \wr C_2$ (the *signed permutation group* or the *hyperoctahedral group*). In the case $d = 4$, the order of $S_4 \wr C_2$ is 384.

Here we shall consider the Galois groups associated with irreducible octic palindromes of the form constructed in Proposition 17. Recall that each polynomial in that family has roots of the form $r^{\pm 1}\mathrm{e}^{\pm \mathrm{i}\theta}, r^{\pm 1}\mathrm{e}^{\pm \mathrm{i}\psi}$. When a polynomial with this root distribution is irreducible, some of the powers of these roots are given by Salem half-norms (Theorem 8).

▶ **Theorem 20.** *Suppose that $f \in \mathbb{Z}[x]$ is an irreducible and simple palindromic octic with four dominant roots such that none of its conjugate ratios are roots of unity (so that H1 and H2 are satisfied). Then, the associated Galois group is isomorphic to either $S_4 \times C_2$ or $A_4 \times C_2$.*

Proof. Let $\alpha, \overline{\alpha}, \beta, \overline{\beta}$ be the four dominant roots of f. As before, we can assume without loss of generality that these roots lie on a circle $|z| = R$ with $R > 1$. The Galois group of the splitting field of f is necessarily a subgroup of the group of automorphisms G of the set $\{\alpha^{\pm 1}, \overline{\alpha}^{\pm 1}, \beta^{\pm 1}, \overline{\beta}^{\pm 1}\} \subset \mathbb{C}$. Further, the Galois group necessarily contains permutations

[1] The author was unable to access Kirby's article in order to verify this attribution.

that fix the relations $xx^{-1} = 1$ for x in the set and $\alpha\alpha^{-1} - \beta\beta^{-1} = 0$. We note that G also contains the *inversion map* $x \mapsto x^{-1}$. The action of G on the roots induces an action of (a subgroup of) S_4 on the set of C_2 orbits $\{x, x^{-1}\}$, which we lift to an action of (a subgroup of) S_4 on the roots.

It is easily verified that the transposition (12) induces one of the maps

$$(\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\overline{\alpha}, \alpha, \beta, \overline{\beta}), \quad \text{or} \quad (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\overline{\alpha}^{-1}, \alpha^{-1}, \beta^{-1}, \overline{\beta}^{-1}).$$

Both of these maps commute with inversion and we note the latter is obtained from the former by applying the inversion map. Thus, without loss of generality, we take (12) as the former map.

In fact, every transposition can be lifted and commutes with inversion. Hence $G = S_4 \times C_2$. The transformations are as follows:

$$\begin{aligned}
&(12)\colon (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\overline{\alpha}, \alpha, \beta, \overline{\beta})\\
&(13)\colon (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\beta^{-1}, \overline{\alpha}, \alpha^{-1}, \overline{\beta})\\
&(14)\colon (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\overline{\beta}^{-1}, \overline{\alpha}, \beta, \alpha^{-1})\\
&(23)\colon (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\alpha, \beta^{-1}, \overline{\alpha}^{-1}, \overline{\beta})\\
&(24)\colon (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\alpha, \overline{\beta}^{-1}, \beta, \overline{\alpha}^{-1})\\
&(34)\colon (\alpha, \overline{\alpha}, \beta, \overline{\beta}) \mapsto (\alpha, \overline{\alpha}, \overline{\beta}, \beta).
\end{aligned}$$

By way of explanation, these transformations are deduced as follows. First, when choosing a lift of a transposition from the set of C_2-orbits to the set of roots, one can invert an even number of the orbit pairs $\{x, x^{-1}\}$. Second, the choice of inverting none or all four orbit pairs differs only by the inversion map. Similarly, the two options when inverting two of the orbit pairs differ only by inversion. A transposition σ necessarily preserves $\alpha\overline{\alpha} - \beta\overline{\beta} = 0$. Some careful accounting shows that permutations such as $\sigma(\overline{\alpha}) = \beta$ (or $\sigma(\overline{\alpha}) = \overline{\beta}$) lead to $\alpha\beta - \overline{\alpha\beta} = 0$ (or $\alpha\overline{\beta} - \overline{\alpha}\beta = 0$) and so $\alpha\beta \in \mathbb{R}$ (or $\alpha\overline{\beta} \in \mathbb{R}$). Such conclusions contradict our assumptions on the conjugate ratios of f. Similar arguments lead to the conclusion that there is a unique (that is to say, unique up to inversion) automorphism of the roots that preserves the aforementioned relations. Hence the Galois group is a subgroup of $S_4 \times C_2$.

We make the following useful observations. Firstly, the Galois group is a transitive subgroup of $S_4 \times C_2$ because f is irreducible. Secondly, since the action of C_2 on the roots is *free* (given $g, h \in C_2$ and a root x with $gx = hx$ then necessarily $g = h$), the transitive subgroups of $S_4 \times C_2$ are precisely the direct product of C_2 and a transitive subgroup of S_4. Finally, we note that (12)(34), representing complex conjugation, is certainly an element of the Galois group.

The transitive subgroups of S_4 are isomorphic to S_4, A_4, D_4, C_4, and K_4. Because C_4 and K_4 are Abelian groups, (12)(34) either lies in the centre of each group or the subgroup does not contain (12)(34). In the former case we deduce that the splitting field is a Kroneckerian field (by Corollary 9) and so the splitting field is either CM or totally real. The field cannot be totally real by our assumption that f has non-real roots. The field is also not a CM-field for otherwise the conjugate ratios are necessarily roots of unity by Theorem 10.

We now focus on eliminating the possibility that the Galois group of f is $D_4 \times C_2$. There are three conjugate subgroups of S_4 that are isomorphic to D_4:

$$\begin{aligned}
D_{4,0} &= \{e, (12), (34), (12)(34), (13)(24), (14)(23), (1324), (1423)\} = \langle (1324), (12)\rangle,\\
D_{4,1} &= \{e, (13), (24), (13)(24), (12)(34), (14)(23), (1234), (1432)\} = \langle (1234), (13)\rangle, \text{ and}\\
D_{4,2} &= \{e, (14), (23), (14)(23), (12)(34), (13)(24), (1243), (1342)\} = \langle (1243), (14)\rangle.
\end{aligned}$$

Note (12)(34) lies in the centre of $D_{4,0}$ and so we can once again employ Theorem 10 to deduce that $D_{4,0} \times C_2$ cannot be the Galois group of the splitting field of f. Assume, for a contradiction, that $D_{4,1} \times C_2$ is the Galois group of the splitting field of f. It is easily verified that $\alpha/\beta + \overline{\alpha/\beta}$ is invariant under the action of $D_{4,1} \times C_2$. Hence $\alpha/\beta + \overline{\alpha/\beta} \in \mathbb{Q}$. Since $\alpha/\beta \notin \mathbb{Q}$, we deduce that

$$(x - \alpha/\beta)(x - \overline{\alpha/\beta}) = x^2 - (\alpha/\beta + \overline{\alpha/\beta})x + 1 \in \mathbb{Q}[x].$$

Thus α/β satisfies a quadratic monic polynomial; moreover, since α/β is an algebraic integer it follows that $\alpha/\beta + \overline{\alpha/\beta} \in \mathbb{Z}$. Since $|\alpha/\beta| = 1$, we find $\alpha/\beta + \overline{\alpha/\beta} \in \{\pm 2, \pm 1, 0\}$. Each of the roots of the five possible polynomials are roots of unity, which contradicts our assumption on the conjugate ratios. Mutatis mutandis, one eliminates the possibility that the Galois group is $D_{4,2} \times C_2$ by similar consideration of $\alpha/\overline{\beta} + \overline{\alpha}/\beta$.

Thus the Galois group of the splitting field of f is either $S_4 \times C_2$ or $A_4 \times C_2$, as required. ◀

▶ Remark 21. It is not possible to strengthen the above result: the Galois group of the palindrome $x^8 + x^7 - x^6 + x^5 + 5x^4 + x^3 - x^2 + x + 1$ is $S_4 \times C_2$, whilst the Galois group of the palindrome $x^8 + x^7 - 3x^6 + x^5 + 9x^4 + x^3 - 3x^2 + x + 1$ is $A_4 \times C_2$. Both polynomials satisfy the assumptions in Theorem 20. For the avoidance of doubt, the converse of the statement in Theorem 20 is not true: the Galois group of the palindrome $x^8 + x^6 + 6x^5 + 9x^4 + 6x^3 + x^2 + 1$, which possesses a single complex-conjugate pair of dominant roots, is $S_4 \times C_2$.

5 Directions for Future Research

Our main contribution to the state of the art is a new proof of Theorem 1: the Skolem Problem is decidable for the class of reversible integer linear recurrence sequences of order at most seven. The benefit of our approach (by comparison to the case analysis in [19]) is the potential for applications to related decision problems. In this section we suggest directions for future research; in particular, variations on the Positivity and Skolem Problems for linear recurrence sequences.

5.1 The Simple Reversible Positivity Problem

The *Positivity Problem* asks to decide whether the terms in an integer linear recurrence sequence are all non-negative. Ouaknine and Worrell [26] demonstrated that the Positivity Problem is decidable for *simple* linear recurrence sequences (those whose characteristic polynomials have no repeated roots) of order at most nine. The proofs therein very much depend on an approach via Baker's Theorem for linear forms in logarithms. Those authors identify the class of non-degenerate linear recurrence sequences of order ten that are not amenable to said approach: the characteristic polynomials in this class have one dominant real root, four complex-conjugate pairs of dominant roots, and one non-dominant root.

Consider the family of monic polynomials in $\mathbb{Z}[x]$ of degree ten, with constant coefficient ± 1, and the above distribution of roots. Suppose that $f \in \mathbb{Z}[x]$ is a polynomial in this class. We immediately find that f is irreducible since each dominant root of f is necessarily conjugate to the single non-dominant root. The roots of f lie on two concentric circles centred at the origin and so we can invoke Theorem 8 to derive a contradiction. Thus the obstruction to decidability falls away under our extra assumptions and so we extend the result in [26] in this restricted setting. In summary, the *Simple Reversible Positivity Problem* is decidable for recurrences of order at most ten and so we have proved Corollary 3.

▶ Remark 22. Let us give an alternative proof for Corollary 3; this alternative proof does not invoke Theorem 8.

Observe that any candidate polynomial in our discussion is irreducible. We can invoke standard results for irreducible polynomials with many dominant roots. For example, versions of the following lemma are found in [32, 10].

▶ **Lemma 23.** *Suppose that α is an algebraic number with Galois conjugates β and γ satisfying $\alpha^2 = \beta\gamma$. Then the conjugate ratio α/β is a root of unity.*

Now suppose that a polynomial $f \in \mathbb{Z}[x]$ of degree ten has the aforementioned distribution of roots and is the characteristic polynomial of a non-degenerate linear recurrence sequence. Since f is necessarily irreducible and has a dominant positive root, we can invoke Lemma 23. We deduce that f is the characteristic polynomial of a degenerate recurrence sequence, a contradiction. Thus the obstacle to deciding positivity of simple linear recurrence sequences at order ten falls away under our additional assumption of reversibility: there are no sequences in the aforementioned class. As an aside, Lemma 23 is used by Dubickas and Smyth [7] as a stepping-stone towards Theorem 8.

We propose that such approaches as outlined above could lead to further decidability results for variants of the Positivity Problem. The key observations on the characteristic polynomials were: irreducibility and a dominant positive root. Here we can make the latter assumption without loss of generality. Indeed, let us recall the following classical consequence of Pringsheim's Theorem in complex analysis that is pertinent to deciding positivity.

▶ **Lemma 24.** *Suppose that a non-zero real-valued linear recurrence sequence $\langle X_n \rangle_n$ has no positive dominant characteristic root. Then the cardinalities of the sets $\{n \in \mathbb{N} : X_n > 0\}$ and $\{n \in \mathbb{N} : X_n < 0\}$ are both infinite.*

5.2 The Skolem Problem and unit-norm roots

In this paper we invoke results such as Theorem 8 in order to reduce the reversible Skolem Problem at orders five, six, and seven, to decidable instances of the Skolem Problem (Theorem 2). It is interesting to speculate that techniques involving identities between roots (the Galois theory underlying Theorem 8) have further applications in establishing decidability results for linear recurrence sequences. Indeed, the general version of Theorem 8 (see [7, Theorem 2.1]) considers not only algebraic integers that are units, but also algebraic numbers that are *unit-norms*. An algebraic number α is a *unit-norm* if the minimal polynomial of α is of the form $a_d x^d - a_{d-1}x^{d-1} - \cdots - a_1 x - a_0 \in \mathbb{Z}[x]$ such that $|a_d| = |a_0|$. So the unit-norm algebraic integers are the units.

We can strengthen the statement in Proposition 12 by invoking [7, Theorem 2.1]: we deduce there is no polynomial $a_5x^5 - a_4x^4 - a_3x^3 - a_2x^2 - a_1x \pm a_5 \in \mathbb{Z}[x]$ that satisfies hypotheses H1 and H2. Thus not only do we settle decidability of the reversible Skolem Problem at order five, but also decidability of the Skolem Problem at order five for the class of rational-valued linear recurrence sequences that satisfy a relation of the form

$$X_{n+5} = a_4X_{n+4} + a_3X_{n+3} + a_2X_{n+2} + a_1X_{n+1} \pm X_n$$

with $a_1, a_2, a_3, a_4 \in \mathbb{Q}$. Thus we have established Corollary 4.

References

1 S. Akshay, Nikhil Balaji, and Nikhil Vyas. Complexity of Restricted Variants of Skolem and Related Problems. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78:1–78:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2017.78.

2 M. Baake. A brief guide to reversing and extended symmetries of dynamical systems. In *Ergodic theory and dynamical systems in their interactions with arithmetics and combinatorics*, volume 2213 of *Lecture Notes in Math.*, pages 117–135. Springer, Cham, 2018.

3 Vincent D. Blondel and Natacha Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and its Applications*, 351-352:91–98, 2002. Fourth Special Issue on Linear Systems and Control. doi:10.1016/S0024-3795(01)00466-9.

4 Christos Christopoulos and James McKee. Galois theory of Salem polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 148(1):47–54, 2010. doi:10.1017/S0305004109990284.

5 Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics.* Springer-Verlag, Berlin, 1993.

6 Ryan C. Daileda. Algebraic integers on the unit circle. *Journal of Number Theory*, 118(2):189–191, June 2006. doi:10.1016/j.jnt.2005.09.002.

7 A. Dubickas and C. J. Smyth. On the Remak height, the Mahler measure and conjugate sets of algebraic numbers lying on two circles. *Proc. Edinb. Math. Soc. (2)*, 44(1):1–17, 2001. doi:10.1017/S001309159900098X.

8 Graham Everest, Alf van der Poorten, Igor Shparlinski, and Thomas Ward. *Recurrence sequences*, volume 104 of *Mathematical Surveys and Monographs.* Amer. Math. Soc., Providence, RI, 2003.

9 P. Fatou. Sur les séries entières à coefficients entiers. *Comptes Rendus Acad. Sci. Paris*, 138(130):342–344, 1904.

10 Ronald Ferguson. Irreducible polynomials with many roots of equal modulus. *Acta Arith.*, 78(3):221–225, 1997. doi:10.4064/aa-78-3-221-225.

11 Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. Skolem's problem–on the border between decidability and undecidability. Technical report, Turku Centre for Computer Science, 2005.

12 David Joyner. Zeros of some self-reciprocal polynomials. In *Excursions in harmonic analysis. Volume 1*, Appl. Numer. Harmon. Anal., pages 329–348. Birkhäuser/Springer, New York, 2013. doi:10.1007/978-0-8176-8376-4_17.

13 Anatole Katok and Boris Hasselblatt. *Introduction to the Modern Theory of Dynamical Systems.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995. doi:10.1017/CBO9780511809187.

14 G. Kenison, R. Lipton, J. Ouaknine, and J. Worrell. On the Skolem Problem and prime powers. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '20.* ACM, 2020. doi:10.1145/3373207.3404036.

15 David Kirby. Integer matrices of finite order. *Rend. Mat.*, 2(6):403–408, 1969.

16 L. Kronecker. Zwei Sätze über Gleichungen mit ganzzahligen Coefficienten. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1857(53):173–175, January 1857. doi:10.1515/crll.1857.53.173.

17 J S W Lamb. Reversing symmetries in dynamical systems. *Journal of Physics A: Mathematical and General*, 25(4):925–937, February 1992. doi:10.1088/0305-4470/25/4/028.

18 Christer Lech. A note on recurring series. *Arkiv för Matematik*, 2:417–421, 1953.

19 Richard J. Lipton, Florian Luca, Joris Nieuwveld, Joël Ouaknine, David Purser, and James Worrell. On the Skolem Problem and the Skolem Conjecture. To appear in the Symposium on Logic in Computer Science (LICS), 2022.

20 Florian Luca, Joël Ouaknine, and James Worrell. Universal Skolem Sets. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–6. IEEE, 2021. doi:10.1109/LICS52264.2021.9470513.

21 C. R. MacCluer and Charles J. Parry. Units of modulus 1. *Journal of Number Theory*, 7(4):371–375, November 1975. doi:10.1016/0022-314x(75)90040-2.

22 K. Mahler. Eine arithmetische Eigenschaft der Taylor-koeffizienten rationaler Funktionen. *Proc. Akad. Wet. Amst.*, 38:50–69, 1935.

23 K. Mahler and J. Cassels. On the Taylor coefficients of rational functions. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52(1):39–48, 1956.

24 Maurice Mignotte, Tarlok Shorey, and Robert Tijdeman. The distance between terms of an algebraic recurrence sequence. *Journal für die Reine und Angewandte Mathematik*, pages 63–76, 1984.

25 Joël Ouaknine and James Worrell. Decision problems for linear recurrence sequences. In *Reachability problems*, volume 7550 of *Lecture Notes in Computer Science*, pages 21–28. Springer, Heidelberg, 2012.

26 Joël Ouaknine and James Worrell. On the Positivity Problem for Simple Linear Recurrence sequences,. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 318–329, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

27 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, April 2015.

28 Igor Rivin. Walks on groups, counting reducible matrices, polynomials, and surface and free group automorphisms. *Duke Mathematical Journal*, 142(2):353–379, 2008.

29 Igor Rivin. Large Galois groups with applications to Zariski density, 2015. arXiv:1312.3009.

30 A. Schinzel. *Polynomials with Special Regard to Reducibility*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2000. doi:10.1017/CBO9780511542916.

31 Thoralf Skolem. Ein Verfahren zur Behandlung gewisser exponentialer Gleichungen und diophantischer Gleichungen. *8de Skand. Mat. Kongress, Stockholm (1934)*, pages 163–188, 1934.

32 C. Smyth. Conjugate algebraic numbers on conics. *Acta Arithmetica*, 40(4):333–346, 1982.

33 I. Stewart and D. Tall. *Algebraic number theory and Fermat's last theorem*. CRC Press, Boca Raton, FL, fourth edition, 2016.

34 The PARI Group, Univ. Bordeaux. *PARI/GP version 2.13.3*, 2021. available from http://pari.math.u-bordeaux.fr/.

35 Nikolai Vereshchagin. Occurrence of zero in a linear recursive sequence. *Mathematical notes of the Academy of Sciences of the USSR*, 38(2):609–615, August 1985.

36 Paulo Viana and Paula Murgel Veloso. Galois theory of reciprocal polynomials. *The American Mathematical Monthly*, 109(5):466–471, 2002. doi:10.1080/00029890.2002.11919875.

The Complexity of Computing Optimum Labelings for Temporal Connectivity

Nina Klobas
Department of Computer Science, Durham University, UK

George B. Mertzios
Department of Computer Science, Durham University, UK

Hendrik Molter
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Israel

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
Computer Engineering & Informatics Department, University of Patras, Greece

Abstract

A graph is temporally connected if there exists a strict temporal path, i.e., a path whose edges have *strictly increasing* labels, from every vertex u to every other vertex v. In this paper we study *temporal design* problems for undirected temporally connected graphs. The basic setting of these optimization problems is as follows: given a connected undirected graph G, what is the smallest number $|\lambda|$ of time-labels that we need to add to the edges of G such that the resulting temporal graph (G, λ) is temporally connected? As it turns out, this basic problem, called Minimum Labeling (ML), can be optimally solved in polynomial time. However, exploiting the temporal dimension, the problem becomes more interesting and meaningful in its following variations, which we investigate in this paper. First we consider the problem Min. Aged Labeling (MAL) of temporally connecting the graph when we are given an upper-bound on the allowed *age* (i.e., maximum label) of the obtained temporal graph (G, λ). Second we consider the problem Min. Steiner Labeling (MSL), where the aim is now to have a temporal path between any pair of "important" vertices which lie in a subset $R \subseteq V$, which we call the *terminals*. This relaxed problem resembles the problem Steiner Tree in static (i.e., non-temporal) graphs. However, due to the requirement of *strictly* increasing labels in a temporal path, Steiner Tree is *not* a special case of MSL. Finally we consider the age-restricted version of MSL, namely Min. Aged Steiner Labeling (MASL). Our main results are threefold: we prove that (i) MAL becomes NP-complete on undirected graphs, while (ii) MASL becomes W[1]-hard with respect to the number $|R|$ of terminals. On the other hand we prove that (iii) although the age-unrestricted problem MSL remains NP-hard, it is in FPT with respect to the number $|R|$ of terminals. That is, adding the age restriction, makes the above problems *strictly harder* (unless P=NP or W[1]=FPT).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Discrete mathematics

Keywords and phrases Temporal graph, graph labeling, foremost temporal path, temporal connectivity, Steiner Tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.62

Related Version *Full Version*: https://arxiv.org/abs/2202.05880

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.
Hendrik Molter: Supported by the ISF, grant No. 1070/20.
Paul G. Spirakis: Supported by the NeST initiative of the School of EEE and CS at the University of Liverpool and by the EPSRC grant EP/P02002X/1.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 62; pp. 62:1–62:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A temporal (or dynamic) graph is a graph whose underlying topology is subject to discrete changes over time. This paradigm reflects the structure and operation of a great variety of modern networks; social networks, wired or wireless networks whose links change dynamically, transportation networks, and several physical systems are only a few examples of networks that change over time [23, 33, 35]. Inspired by the foundational work of Kempe et al. [25], we adopt here a simple model for temporal graphs, in which the vertex set remains unchanged while each edge is equipped with a set of integer time-labels.

▶ **Definition 1** (temporal graph [25]). *A* temporal graph *is a pair* (G, λ)*, where* $G = (V, E)$ *is an underlying (static) graph and* $\lambda : E \to 2^{\mathbb{N}}$ *is a* time-labeling *function which assigns to every edge of* G *a set of discrete time-labels.*

Here, whenever $t \in \lambda(e)$, we say that the edge e is *active* or *available* at time t. Throughout the paper we may refer to "time-labels" simply as "labels" for brevity. Furthermore, the *age* (or *lifetime*) $\alpha(G, \lambda)$ of the temporal graph (G, λ) is the largest time-label used in it, i.e., $\alpha(G, \lambda) = \max\{t \in \lambda(e) : e \in E\}$. One of the most central notions in temporal graphs is that of a *temporal path* (or *time-respecting path*) which is motivated by the fact that, due to causality, entities and information in temporal graphs can "flow" only along sequences of edges whose time-labels are strictly increasing, or at least non-decreasing.

▶ **Definition 2** (temporal path). *Let* (G, λ) *be a temporal graph, where* $G = (V, E)$ *is the underlying static graph. A* temporal path *in* (G, λ) *is a sequence* $(e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)$*, where* $(e_1, e_2, \ldots, e_k)$ *is a path in* G*,* $t_i \in \lambda(e_i)$ *for every* $i = 1, 2, \ldots, k$*, and* $t_1 < t_2 < \ldots < t_k$*.*

A vertex v is *temporally reachable* (or *reachable*) from vertex u in (G, λ) if there exists a temporal path from u to v. If every vertex v is reachable by every other vertex u in (G, λ), then (G, λ) is called *temporally connected*. Note that, for every temporally connected temporal graph (G, λ), we have that its age is at least as large as the diameter d_G of the underlying graph G. Indeed, the largest label used in any temporal path between two anti-diametrical vertices cannot be smaller than d_G. Temporal paths have been introduced by Kempe et al. [25] for temporal graphs which have only one label per edge, i.e., $|\lambda(e)| = 1$ for every edge $e \in E$, and this notion has later been extended by Mertzios et al. [28] to temporal graphs with multiple labels per edge. Furthermore, depending on the particular application, both variations of temporal paths with non-decreasing [6, 25, 26] and with strictly increasing [15, 28] labels have been studied. In this paper we focus on temporal paths with *strictly increasing* labels. Due to the very natural use of temporal paths in various contexts, several path-related notions, such as temporal analogues of distance, diameter, reachability, exploration, and centrality have also been studied [1–3, 6, 8, 10, 11, 13, 15–18, 20, 26, 28, 32, 34, 36].

Furthermore, some non-path temporal graph problems have been recently introduced too, including for example temporal variations of maximal cliques [7, 37], vertex cover [4, 21], vertex coloring [31], matching [29], and transitive orientation [30]. Motivated by the need of restricting the spread of epidemic, Enright et al. [15] studied the problem of removing the smallest number of time-labels from a given temporal graph such that every vertex can only temporally reach a limited number of other vertices. Deligkas et al. [12] studied the problem of accelerating the spread of information for a set of sources to all vertices in a temporal graph, by only using delaying operations, i.e., by shifting specific time-labels to a later time slot. The problems studied in [12] are related but orthogonal to our temporal connectivity problems. Various other temporal graph modification problems have been also studied, see for example [6, 11, 13, 16, 34].

The time-labels of an edge e in a temporal graph indicate the discrete units of time (e.g., days, hours, or even seconds) in which e is active. However, in many real dynamic systems, e.g., in synchronous mobile distributed systems that operate in discrete rounds, or in unstable chemical or physical structures, maintaining an edge over time requires energy and thus comes at a cost. One natural way to define the *cost* of the whole temporal graph (G, λ) is the *total number* of time-labels used in it, i.e., the total cost of (G, λ) is $|\lambda| = \sum_{e \in E} |\lambda_e|$.

In this paper we study *temporal design* problems of undirected temporally connected graphs. The basic setting of these optimization problems is as follows: given an undirected graph G, what is the smallest number $|\lambda|$ of time-labels that we need to add to the edges of G such that (G, λ) is temporally connected? As it turns out, this basic problem can be optimally solved in polynomial time, thus answering to a conjecture made in [2]. However, exploiting the temporal dimension, the problem becomes more interesting and meaningful in its following variations, which we investigate in this paper. First we consider the problem variation where we are given along with the input also an upper bound of the allowed *age* (i.e., maximum label) of the obtained temporal graph (G, λ). This age restriction is sensible in more pragmatic cases, where delaying the latest arrival time of any temporal path incurs further costs, e.g., when we demand that all agents in a safety-critical distributed network are synchronized as quickly as possible, and with the smallest possible number of communications among them. Second we consider problem variations where the aim is to have a temporal path between any pair of "important" vertices which lie in a subset $R \subseteq V$, which we call the *terminals*. For a detailed definition of our problems we refer to Section 2.

Here it is worth noting that the latter relaxation of temporal connectivity resembles the problem Steiner Tree in static (i.e., non-temporal) graphs. Given a connected graph $G = (V, E)$ and a set $R \subseteq V$ of terminals, Steiner Tree asks for a smallest-sized subgraph of G which connects all terminals in R. Clearly, the smallest subgraph sought by Steiner Tree is a tree. As it turns out, this property does not carry over to the temporal case. Consider for example an arbitrary graph G and a terminal set $R = \{a, b, c, d\}$ such that G contains an induced cycle on four vertices a, b, c, d; that is, G contains the edges ab, bc, cd, da but not the edges ac or bd. Then, it is not hard to check that only way to add the smallest number of time-labels such that all vertices of R are temporally connected is to assign one label to each edge of the cycle on a, b, c, d, e.g., $\lambda(ab) = \lambda(cd) = 1$ and $\lambda(bc) = \lambda(cd) = 2$. The main underlying reason for this difference with the static problem Steiner Tree is that temporal connectivity is *not transitive* and *not symmetric:* if there exists temporal paths from u to v, and from v to w, it is not a priori guaranteed that a temporal path from v to u, or from u to w exists.

Temporal network design problems have already been considered in previous works. Mertzios et al. [28] proved that it is APX-hard to compute a minimum-cost labeling for temporally connecting an input *directed* graph G, where the age of the graph is upper-bounded by the diameter of G. This hardness reduction was strongly facilitated by the careful placement of the edge directions in the constructed instance, in which every vertex was reachable in the static graph by only constantly many vertices. Unfortunately this cannot happen in an undirected connected graph, where every vertex is reachable by all other vertices. Later, Akrida et al. [2] proved that it is also APX-hard to *remove* the largest number of time-labels from a given temporally connected (undirected) graph (G, λ), while still maintaining temporal connectivity. In this case, although there are no edge directions, the hardness reduction was strongly facilitated by the careful placement of the initial time-labels of λ in the input temporal graph, in which every pair of vertices could be connected by only a few different temporal paths, among which the solution had to choose. Unfortunately this cannot happen when the goal is to add time-labels to an undirected connected graph, where there are potentially multiple ways to temporally connect a pair of vertices (even if we upper-bound the largest time-label by the diameter).

Summarizing, the above technical difficulties seem to be the reason why the problem of *adding* the minimum number of time-labels with an age-restriction to an *undirected* graph to achieve temporal connectivity remained open until now for the last decade. In this paper we overcome these difficulties by developing a hardness reduction from a variation of the problem Max XOR SAT (see Theorem 12 in Section 3) where we manage to add the appropriate (undirected) edges among the variable-gadgets such that simultaneously (i) the distance between any two vertices from different variable gadgets remains small (constant) and (ii) there is no shortest path between two vertices of the *same* variable gadget that leaves this gadget.

Our contribution and road-map. In the first part of our paper, in Section 3, we present our results on Min. Aged Labeling (MAL). This problem is the same as ML, with the additional restriction that we are given along with the input an upper bound on the allowed *age* of the resulting temporal graph (G, λ). Using a technically involved reduction from a variation of Max XOR SAT, we prove that MAL is NP-complete on undirected graphs, even when the required maximum age is equal to the diameter d_G of the input static graph G.

In the second part of our paper, in Section 4, we present our results on the Steiner-tree versions of the problem, namely on Min. Steiner Labeling (MSL) and Min. Aged Steiner Labeling (MASL). The difference of MSL from ML is that, here, the goal is to have a temporal path between any pair of "important" vertices which lie in a given subset $R \subseteq V$ (the *terminals*). In Section 4.1 we prove that MSL is NP-complete by a reduction from Vertex Cover, the correctness of which requires showing structural properties of MSL. Here it is worth recalling that, as explained above, the classical problem Steiner Tree on static graphs is *not* a special case of MSL, due to the requirement of strictly increasing labels in a temporal path. Furthermore, we would like to emphasize here that, as temporal connectivity is neither transitive nor symmetric, a straightforward NP-hardness reduction from Steiner Tree to MSL does not seem to exist. For example, as explained above, in a graph that contains a C_4 with its four vertices as terminals, labeling a Steiner tree is sub-optimal for MSL.

In Section 4.2 we provide a fixed-parameter tractable (FPT) algorithm for MSL with respect to the number $|R|$ of terminal vertices, by providing a parameterized reduction to Steiner Tree. The proof of correctness of our reduction, which is technically quite involved, is of independent interest, as it proves crucial graph-theoretical properties of minimum temporal Steiner labelings. In particular, for our algorithm we prove (see Lemma 14) that, for any undirected graph G with a set R of terminals, there always exists at least one minimum temporal Steiner labeling (G, λ) which labels edges either from (i) a tree or from (ii) a tree with one extra edge that builds a C_4.

In Section 4.3 we prove that MASL is W[1]-hard with respect to the number $|R|$ of terminals. Our results actually imply the stronger statement that MASL is W[1]-hard even with respect to the number of time-labels of the solution (which is a larger parameter than the number $|R|$ of terminals).

Finally, we complete the picture by providing some auxiliary results in our preliminary Section 2. More specifically, in Section 2.1 we prove that ML can be solved in polynomial time, and in Section 2.2 we prove that the analogue minimization versions of ML and MAL on directed acyclic graphs are solvable in polynomial time.

Due to space constraints, proofs of results marked with $\star$ are (partially) deferred to a full version on arXiv [27].

2 Preliminaries and notation

Given a (static) undirected graph $G = (V, E)$, an edge between two vertices $u, v \in V$ is denoted by uv, and in this case the vertices u, v are said to be *adjacent* in G. If the graph is directed, we will use the ordered pair (u, v) (resp. (v, u)) to denote the oriented edge from u to v (resp. from v to u). The *age* of a temporal graph (G, λ) is denoted by $\alpha(G, \lambda) = \max\{t \in \lambda(e) : e \in E\}$. A temporal path $(e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)$ from vertex u to vertex v is called *foremost*, if it has the smallest arrival time t_k among all temporal paths from u to v. Note that there might be another temporal path from u to v that uses fewer edges than a foremost path. A temporal graph (G, λ) is *temporally connected* if, for every pair of vertices $u, v \in V$, there exists a temporal path (see Definition 2) P_1 from u to v and a temporal path P_2 from v to u. Furthermore, given a set of terminals $R \subseteq V$, the temporal graph (G, λ) is *R-temporally connected* if, for every pair of vertices $u, v \in R$, there exists a temporal path from u to v and a temporal path from v to u; note that P_1 and P_2 can also contain vertices from $V \setminus R$. Now we provide our formal definitions of our four decision problems.

MIN. LABELING (ML)

Input: A static graph $G = (V, E)$ and a $k \in \mathbb{N}$.
Question: Does there exist a temporally connected temporal graph (G, λ), where $|\lambda| \leq k$?

MIN. AGED LABELING (MAL)

Input: A static graph $G = (V, E)$ and two integers $a, k \in \mathbb{N}$.
Question: Does there exist a temporally connected temporal graph (G, λ), where $|\lambda| \leq k$ and $\alpha(\lambda) \leq a$?

MIN. STEINER LABELING (MSL)

Input: A static graph $G = (V, E)$, a subset $R \subseteq V$ and a $k \in \mathbb{N}$.
Question: Does there exist a temporally R-connected temporal graph (G, λ), where $|\lambda| \leq k$?

MIN. AGED STEINER LABELING (MASL)

Input: A static graph $G = (V, E)$, a subset $R \subseteq V$, and two integers $a, k \in \mathbb{N}$.
Question: Does there exist a temporally R-connected temporal graph (G, λ), where $|\lambda| \leq k$ and $\alpha(\lambda) \leq a$?

Note that, for both problems MAL and MASL, whenever the input age bound a is strictly smaller than the diameter d of G, the answer is always NO. Thus, we always assume in the remainder of the paper that $a \geq d$, where d is the diameter of the input graph G. For simplicity of the presentation, we denote next by $\kappa(G, d)$ the smallest number k for which (G, k, d) is a YES instance for MAL.

▶ **Observation 3 ($\star$).** *For every graph G with n vertices and diameter d, we have that $\kappa(G, d) \leq n(n - 1)$.*

The next lemma shows that the upper bound of Observation 3 is asymptotically tight as, for cycle graphs C_n with diameter d, we have that $\kappa(C_n, d) = \Theta(n^2)$.

▶ **Lemma 4 ($\star$).** *Let C_n be a cycle on n vertices, where $n \neq 4$, and let d be its diameter. Then*

$$\kappa(C_n, d) = \begin{cases} d^2, & \text{when } n = 2d \\ 2d^2 + d, & \text{when } n = 2d + 1. \end{cases}$$

2.1 A polynomial-time algorithm for ML

As a first warm-up, we study the problem ML, where no restriction is imposed on the maximum allowed age of the output temporal graph. It is already known by Akrida et al. [2] that any undirected graph can be made temporally connected by adding at most $2n-3$ time-labels, while for trees $2n-3$ labels are also necessary. Moreover, it was conjectured that every graph needs at least $2n-4$ time-labels [2]. Here we prove their conjecture true by proving that, if G contains (resp. does not contain) the cycle C_4 on four vertices as a subgraph, then (G,k) is a YES instance of ML if and only if $k \geq 2n-4$ (resp. $k \geq 2n-3$). The proof is done via a reduction to the gossip problem [9] (for a survey on gossiping see also [22]).

The related problem of achieving temporal connectivity by assigning to every edge of the graph at most one time-label, has been studied by Göbel et al. [19], where the relationship with the gossip problem has also been drawn. Contrary to ML, this problem is NP-hard [19]. That is, the possibility of assigning two or more labels to an edge makes the problem computationally much easier. Indeed, in a C_4-free graph with n vertices, an optimal solution to ML consists in assigning in total $2n-3$ time-labels to the $n-1$ edges of a spanning tree. In such a solution, one of these $n-1$ edges receives one time-label, while each of the remaining $n-2$ edges receives two time-labels. Similarly, when the graph contains a C_4, it suffices to span the graph with four trees tooted at the vertices of the C_4, where each of the edges of the C_4 receives one time-label and each edge of the four trees receives two labels. That is, a graph containing a C_4 can be temporally connected using $2n-4$ time-labels.

In the gossip problem we have n agents from a set A. At the beginning, every agent $x \in A$ holds its own secret. The goal is that each agent eventually learns the secret of every other agent. This is done by producing a sequence of unordered pairs (x,y), where $x,y \in A$ and each such pair represents one phone call between the agents involved, during which the two agents exchange all the secrets they currently know.

The above gossip problem is naturally connected to ML. The only difference between the two problems is that, in gossip, all calls are non-concurrent, while in ML we allow concurrent temporal edges, i.e., two or more edges can appear at the same time slot t. Therefore, in order to transfer the known results from gossip to ML, it suffices to prove that in ML we can equivalently consider solutions with non-concurrent edges.

▶ **Theorem 5** ($\star$). *Let $G=(V,E)$ be a connected graph. Then the smallest $k \in \mathbb{N}$ for which (G,k) is a YES instance of ML is:*

$$k = \begin{cases} 2n-4, & \text{if } G \text{ contains } C_4 \text{ as a subgraph,} \\ 2n-3, & \text{otherwise.} \end{cases}$$

2.2 A polynomial-time algorithm for directed acyclic graphs

As a second warm-up, we show that the minimization analogues of ML and MAL on directed acyclic graphs (DAGs) are solvable in polynomial time. More specifically, for the minimization analogue of ML we provide an algorithm which, given a DAG $G=(V,A)$ with diameter d_G, computes a temporal labeling function λ which assigns the smallest possible number of time-labels on the arcs of G with the following property: for every two vertices $u,v \in V$, there exists a directed temporal path from u to v in (G,λ) if and only if there exists a directed path from u to v in G. Moreover, the age $\alpha(G,\lambda)$ of the resulting temporal graph is *equal* to d_G. Therefore, this immediately implies a polynomial-time algorithm for the minimization analogue of MAL on DAGs. For notation uniformity, we call these minimization problems $\text{ML}_{directed}$ and $\text{MAL}_{directed}$, respectively.

▶ **Theorem 6** (⋆). *Let $G = (V, E)$ be a DAG with n vertices and m arcs. Then $ML_{directed}(G)$ and $MAL_{directed}(G)$ can be both computed in $O(n(n + m))$ time.*

3 MAL is NP-complete

In this section we prove that it is NP-hard to determine the number of labels in an optimal labeling of a static, undirected graph G, where the age, i.e., the maximum label used, is not larger than the diameter of the input graph.

To prove this we provide a reduction from the NP-hard problem Monotone Max XOR(3) (or MonMaxXOR(3) for short). This is a special case of the classical Boolean satisfiability problem, where the input formula ϕ consists of the conjunction of *monotone* XOR clauses of the form $(x_i \oplus x_j)$, i.e., variables x_i, x_j are non-negated. If each variable appears in exactly r clauses, then ϕ is called a *monotone* Max XOR(r) formula. A clause $(x_i \oplus x_j)$ is *XOR-satisfied* (or simply *satisfied*) if and only if $x_i \neq x_j$. In Monotone Max XOR(r) we are trying to find a truth assignment τ of ϕ which satisfies the maximum number of clauses. As it can be easily checked, MonMaxXOR(3) encodes the problem Max-Cut on cubic graphs, which is known to be NP-hard [5]. Therefore we conclude the following.

▶ **Theorem 7** ([5]). *MonMaxXOR(3) is NP-hard.*

Now we explain our reduction from MonMaxXOR(3) to the problem Minimum Aged Labeling (MAL), where the input static graph G is undirected and the desired age of the output temporal graph is the diameter d of G . Let ϕ be a monotone Max XOR(3) formula with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$. Note that $m = \frac{3}{2}n$, since each variable appears in exactly 3 clauses. From ϕ we construct a static undirected graph G_ϕ with diameter $d = 10$, and prove that there exists a truth assignment τ which satisfies at least k clauses in ϕ, if and only if there exists a labeling λ_ϕ of G_ϕ, with $|\lambda_\phi| \leq \frac{13}{2}n^2 + \frac{99}{2}n - 8k$ labels and with age $\alpha(G, \lambda) \leq 10$.

High-level construction

For each variable x_i, $1 \leq i \leq n$, we construct a variable gadget X_i that consists of a "starting" vertex s_i and three "ending" vertices t_i^ℓ (for $\ell \in \{1, 2, 3\}$); these ending vertices correspond to the appearances of x_i in three clauses of ϕ. In an optimum labeling $\lambda(\phi)$, in each variable gadget there are exactly two labelings that temporally connect starting and ending vertices, which correspond to the True or False truth assignment of the variable in the input formula ϕ. For every clause $(x_i \oplus x_j)$ we identifying corresponding ending vertices of X_i and X_j (as well as some other auxiliary vertices and edges). Whenever $(x_i \oplus x_j)$ is satisfied by a truth assignment of ϕ, the labels of the common edges of X_i and X_j in an optimum labeling coincide (thus using few labels); otherwise we need additional labels for the common edges of X_i and X_j.

Detailed construction of G_ϕ

For each variable x_i from ϕ we create a variable gadget X_i, that consists of a *base* BX_i on 11 vertices, $BX_i = \{s_i, a_i, b_i, c_i, d_i, e_i, \overline{a_i}, \overline{b_i}, \overline{c_i}, \overline{d_i}, \overline{e_i}\}$, and three *forks* F^1X_i, F^2X_i, F^3X_i, each on 9 vertices, $F^\ell X_i = \{t_i^\ell, f_i^\ell, g_i^\ell, h_i^\ell, m_i^\ell, \overline{f_i}^\ell, \overline{g_i}^\ell, \overline{h_i}^\ell, \overline{m_i}^\ell\}$, where $\ell \in \{1, 2, 3\}$. Vertices in the base BX_i are connected in the following way: there are two paths of length 5: $s_i a_i b_i c_i d_i e_i$ and $s_i \overline{a_i} \overline{b_i} \overline{c_i} \overline{d_i} \overline{e_i}$, and 5 extra edges of form $y_i \overline{y_i}$, where $y \in \{a, b, c, d, e\}$. Vertices in each fork $F^\ell X_i$ (where $\ell \in \{1, 2, 3\}$) are connected in the following way: there are two paths of length

4: $t_i^\ell m_i^\ell h_i^\ell g_i^\ell f_i^\ell$ and $t_i^\ell \overline{m_i}^\ell \overline{h_i}^\ell \overline{g_i}^\ell \overline{f_i}^\ell$, and 4 extra edges of form $y_i\overline{y_i}^\ell$, where $y \in \{m, h, g, f\}$. The base BX_i of the variable gadget X_i is connected to each of the three forks $F^\ell X_i$ via two edges $e_i f_i^\ell$ and $\overline{e_i}\overline{f_i}^\ell$, where $\ell \in \{1, 2, 3\}$. For an illustration see Figure 1.

For an easier analysis we fix the following notation. The vertex $s_i \in BX_i$ is called a *start vertex* of X_i, vertices t_i^ℓ ($\ell \in \{1, 2, 3\}$) are called *ending vertices* of X_i, a path connecting s_i, t_i^ℓ that passes through vertices $a_i b_i c_i d_i e_i f_i^\ell g_i^\ell h_i^\ell m_i^\ell$ (resp. $\overline{a_i}\overline{b_i} \dots \overline{m_i}^\ell$) is called the *left* (resp. *right*) s_i, t_i^ℓ-path. The left (resp. right) s_i, t_i^ℓ-path is a disjoint union of the left (resp. right) path on vertices of the base BX_i of X_i, an edge of form $e_i f_i^\ell$ (resp. $\overline{e_i}\overline{f_i}^\ell$) called the left (resp. right) *bridge edge* and the left (resp. right) path on vertices of the ℓ-th fork $F^\ell X_i$ of X_i. The edges $y_i\overline{y_i}$, where $y \in \{a, b, c, d, e, f^\ell, g^\ell, h^\ell, m^\ell\}$, $\ell \in \{1, 2, 3\}$, are called *connecting edges*.

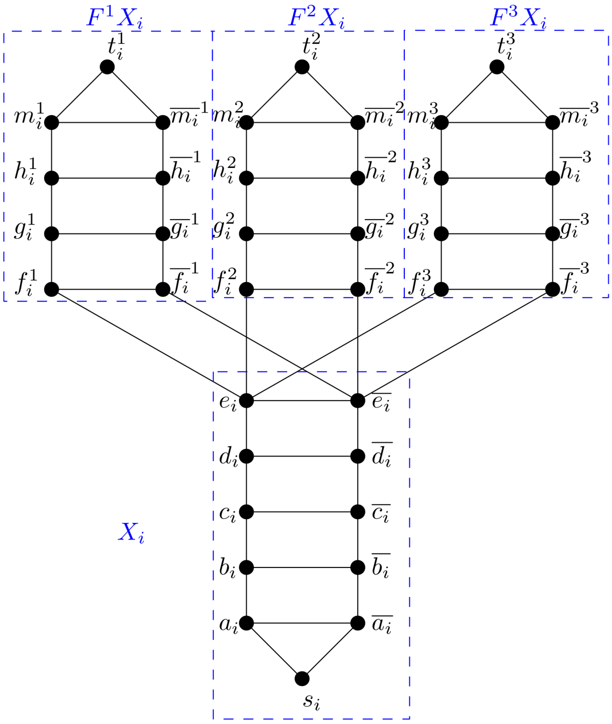


Figure 1 An example of a variable gadget X_i in G_ϕ, corresponding to the variable x_i from ϕ.

Connecting variable gadgets

There are two ways in which we connect two variable gadgets, depending whether they appear in the same clause in ϕ or not.

1. Two variables x_i, x_j do not appear in any clause together. In this case we add the following edges between the variable gadgets X_i and X_j:
   - from e_i (resp. $\overline{e_i}$) to $f_j^{\ell'}$ and $\overline{f_j}^{\ell'}$, where $\ell' \in \{1, 2, 3\}$,
   - from e_j (resp. $\overline{e_j}$) to f_i^ℓ and $\overline{f_i}^\ell$, where $\ell \in \{1, 2, 3\}$,
   - from d_i (resp. $\overline{d_i}$) to d_j and $\overline{d_j}$.

   We call these edges the *variable edges*. For an illustration see Figure 2.

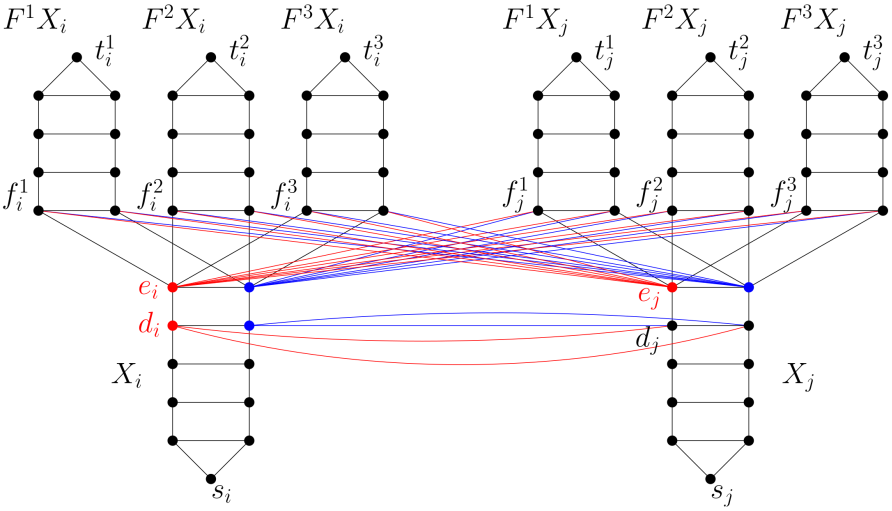


Figure 2 An example of two non-intersecting variable gadgets and variable edges among them.

2. Let $C = (x_i \oplus x_j)$ be a clause of ϕ, that contains the r-th appearance of the variable x_i and r'-th appearance of the variable x_j. In this case we identify the r-th fork F^rX_i of X_i with the r'-th fork $F^{r'}X_j$ of X_j in the following way:
   - $t_i^r = t_j^{r'}$,
   - $\{f_i^r, g_i^r, h_i^r, m_i^r\} = \{\overline{f_j}^{r'}, \overline{g_j}^{r'}, \overline{h_j}^{r'}, \overline{m_j}^{r'}\}$ respectively, and
   - $\{\overline{f_i}^{r}, \overline{g_i}^{r}, \overline{h_i}^{r}, \overline{m_i}^{r}\} = \{f_j^{r'}, g_j^{r'}, h_j^{r'}, m_j^{r'}\}$ respectively.

   Besides that we add the following edges between the variable gadgets X_i and X_j:
   - from e_i (resp. $\overline{e_i}$) to $f_j^{\ell'}$ and $\overline{f_j}^{\ell'}$, where $\ell' \in \{1,2,3\} \setminus \{r'\}$,
   - from e_j (resp. $\overline{e_j}$) to f_i^{ℓ} and $\overline{f_i}^{\ell}$, where $\ell \in \{1,2,3\} \setminus \{r\}$,
   - from d_i (resp. $\overline{d_i}$) to d_j and $\overline{d_j}$.

   For an illustration see Figure 3.

This finishes the construction of G_ϕ. Before continuing with the reduction, we prove the following structural property of G_ϕ.

▶ **Lemma 8** ($\star$). *The diameter d_ϕ of G_ϕ is* 10.

▶ **Theorem 9** ($\star$). *If $OPT_{\text{MonMaxXOR(3)}}(\phi) \geq k$ then $OPT_{MAL}(G_\phi, d_\phi) \leq \frac{13}{2}n^2 + \frac{99}{2}n - 8k$, where n is the number of variables in the formula ϕ.*

Before proving the statement in the other direction, we have to show some structural properties. Let us fix the following notation. If a labeling λ_ϕ labels all left (resp. right) paths of the variable gadget X_i (i.e., both bottom-up from s_i to t_i^1, t_i^2, t_i^3 and top-down from t_i^1, t_i^2, t_i^3 to s_i with labels $1, 2 \ldots, 10$ in this order), then we say that the variable gadget X_i is *left-aligned* (resp. *right-aligned*) in the labeling λ_ϕ. Note, if at least one edge on any of these left (resp. right) paths of X_i is not labeled with the appropriate label between 1 and 10, then the variable gadget is *not* left-aligned (resp. *not* right-aligned). Every temporal path from s_i to t_i^ℓ (resp. from t_i^ℓ to s_i) of length 10 in X_i is called an *upward path* (resp. a *downward path*) in X_i. Any part of an upward (resp. downward) path is called a *partial* upward (resp. downward) path. Note that, for any $\ell, \ell' \in \{1,2,3\}$, $\ell \neq \ell'$, a temporal path from t_i^ℓ to $t_i^{\ell'}$ of length 10 is the union of a partial downward path on the fork F_i^ℓ and a partial upward path on $F_i^{\ell'}$. Moreover, note that these two partial downward/upward paths

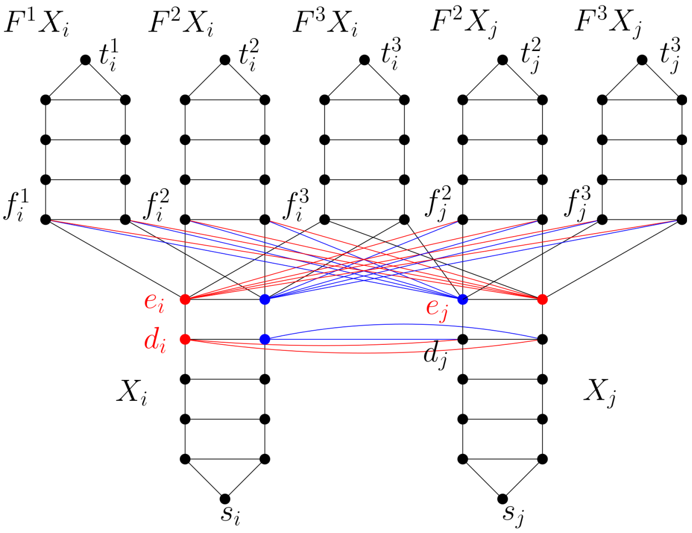


Figure 3 An example of two intersecting variable gadgets X_i, X_j corresponding to variables x_i, x_j, that appear together in some clause in ϕ, where it is the third appearance of x_i and the first appearance of x_j.

must be either both parts of a left temporal path or both parts of a right temporal path between s_i and $t_i^{\ell}, t_i^{\ell'}$. The following technical lemma will allow us to prove the correctness of our reduction.

▶ **Lemma 10** ($\star$). *Let λ_ϕ be a minimum labeling of G_ϕ. Then λ_ϕ can be modified in polynomial time to a minimum labeling of G_ϕ in which each variable gadget X_i is either left-aligned or right-aligned.*

▶ **Theorem 11** ($\star$). *If $OPT_{\text{MAL}}(G_\phi, d_\phi) \leq \frac{13}{2}n^2 + \frac{99}{2}n - 8k$ then $OPT_{\text{MonMaxXOR(3)}}(\phi) \geq k$, where n is the number of variables in the formula ϕ.*

Since MAL is clearly in NP, the next theorem follows directly by Theorems 7, 9, and 11.

▶ **Theorem 12.** *MAL is NP-complete on undirected graphs, even when the required maximum age is equal to the diameter of the input graph.*

4 The Steiner-Tree variations of the problem

In this section we investigate the computational complexity of the Steiner-Tree variations of the problem, namely MSL and MASL. First, we prove in Section 4.1 that the age-unrestricted problem MSL remains NP-hard, using a reduction from Vertex Cover. In Section 4.2 we prove that this problem is in FPT, when parameterized by the number $|R|$ of terminals. Finally, using a parameterized reduction from Multicolored Clique, we prove in Section 4.3 that the age-restricted version MASL is W[1]-hard with respect to $|R|$, even if the maximum allowed age is a constant.

4.1 MSL is NP-complete

▶ **Theorem 13** ($\star$). *MSL is NP-complete.*

Proof sketch. MSL is clearly contained in NP. To prove that the MSL is NP-hard we provide a polynomial-time reduction from the NP-complete Vertex Cover problem [24].

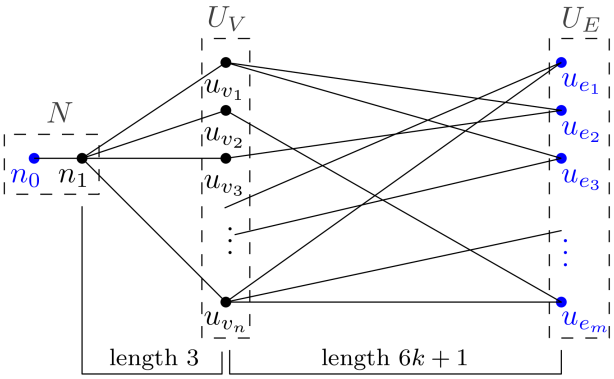


Figure 4 An example of construction of the input graph for MSL.

Vertex Cover
Input: A static graph $G = (V, E)$, a positive integer k.
Question: Does there exist a subset of vertices $S \subseteq V$ such that $|S| = k$ and $\forall e \in E, e \cap S \neq \emptyset$.

Let (G, k) be an input of the Vertex Cover problem and denote $|V(G)| = n, |E(G)| = m$. We assume w.l.o.g. that G does not admit a vertex cover of size $k-1$. We construct (G^*, R^*, k^*), the input of MSL using the following procedure. The vertex set $V(G^*)$ consists of the following vertices:

- two starting vertices $N = \{n_0, n_1\}$,
- a “vertex-vertex” corresponding to every vertex of G: $U_V = \{u_v | v \in V(G)\}$,
- an “edge-vertex” corresponding to every edge of G: $U_E = \{u_e | e \in E(G)\}$,
- $2n + 12m \cdot k$ “dummy” vertices.

The edge set $E(G^*)$ consists of the following edges:

- an edge between starting vertices, i.e., $n_0 n_1$,
- a path of length 3 between a starting vertex n_1 and every vertex-vertex $u_v \in U_V$ using 2 dummy vertices, and
- for every edge $e = vw \in E(G)$ we connect the corresponding edge-vertex u_e with the vertex-vertices u_v and u_w, each with a path of length $6k+1$ using $6k$ dummy vertices.

We set $R^* = \{n_0\} \cup U_E$ and $k^* = 6k + 2m(6k+1) + 1$. This finishes the construction. It is not hard to see that this construction can be performed in polynomial time. For an illustration see Figure 4. Note that any two paths in G^* can intersect only in vertices from $N \cup U_V \cup U_E$ and not in any of the dummy vertices. At the end G^* is a graph with $3n + m(12k+1) + 2$ vertices and $1 + 3n + 2m(6k+1)$ edges.

In the full proof we prove that (G, k) is a YES instance of the Vertex Cover if and only if (G^*, R^*, k^*) is a YES instance of the MSL. ◀

4.2 An FPT-algorithm for MSL with respect to the number of terminals

In this section we provide an FPT-algorithm for MSL, parameterized by the number $|R|$ of terminals. The algorithm is based on a crucial structural property of minimum solutions for MSL: there always exists a minimum labeling λ that labels the edges of a subtree of the input graph (where every leaf is a terminal vertex), and potentially one further edge that forms a C_4 with three edges of the subtree.

Intuitively speaking, we can use an FPT-algorithm for Steiner Tree parameterized by the number of terminals [14] to reveal a subgraph of the MSL instance that we can optimally label using Theorem 5. Since the number of terminals in the created Steiner Tree instance is larger than the number of terminals in the MSL instance by at most a constant, we obtain an FPT-algorithm for MSL parameterized by the number of terminals.

▶ **Lemma 14** ($\star$). *Let $G = (V, E)$ be a graph, $R \subseteq V$ a set of terminals, and k be an integer such that (G, R, k) is a YES instance of MSL and $(G, R, k-1)$ is a NO instance of MSL.*

- *If k is odd, then there is a labeling λ of size k for G such that the edges labeled by λ form a tree, and every leaf of this tree is a vertex in R.*
- *If k is even, then there is a labeling λ of size k for G such that the edges labeled by λ form a graph that is a tree with one additional edge that forms a C_4, and every leaf of the tree is a vertex in R.*

The main idea for the proof of Lemma 14 is as follows. Given a solution labeling λ, we fix one terminal r^* and then (i) we consider the minimum subtree in which r^* can reach all other terminal vertices and (ii) we consider the minimum subtree in which all other terminal vertices can reach r^*. Intuitively speaking, we want to label the smaller one of those subtrees using Theorem 5 and potentially adding an extra edge to form a C_4; we then argue that the obtained labeling does not use more labels than λ. To do that, and to detect whether it is possible to add an edge to create a C_4, we make a number of modifications to the trees until we reach a point where we can show that our solution is correct.

Having Lemma 14, we can now give our algorithm for MSL. As mentioned before, it uses an FPT-algorithm for Steiner Tree parameterized by the number of terminals [14] as a subroutine.

▶ **Theorem 15** ($\star$). *MSL is in FPT when parameterized by the number of terminals.*

4.3 Parameterized Hardness of MASL

Note that, since MASL generalizes both MSL and MAL, NP-hardness of MASL is already implied by both Theorems 12 and 13. In this section, we prove that MASL is W[1]-hard when parameterized by the number $|R|$ of the terminals, even if the restriction a on the age is a constant. To this end, we provide a parameterized reduction from Multicolored Clique. This, together with Theorem 15, implies that MASL is strictly harder than MSL (parameterized by the number $|R|$ of terminals), unless FPT=W[1].

▶ **Theorem 16** ($\star$). *MASL is W[1]-hard when parameterized by the number $|R|$ of the terminals, even if the restriction a on the age is a constant.*

Note here that, in the constructed instance of MASL in the proof of Theorem 16, the number of labels is also upper-bounded by a function of the number of colors in the instance of Multicolored Clique. Therefore the proof of Theorem 16 implies also the next result, which is even stronger (since in every solution of MASL the number of time-labels is lower-bounded by a function of the number $|R|$ of terminals).

▶ **Corollary 17.** *MASL is W[1]-hard when parameterized by the number k of time-labels, even if the restriction a on the age is a constant.*

5 Concluding remarks

Several open questions arise from our results. As we pointed out in Lemma 4, $\kappa(C_n, d) = \Theta(n^2)$, while $\kappa(G, d) = O(n^2)$ for every graph G by Observation 3. For which graph classes $\mathcal{G}$ do we have $\kappa(G, d) = o(n^2)$ (resp. $\kappa(G, d) = O(n)$) for every $G \in \mathcal{G}$?

As we proved in Theorem 12, MAL is NP-complete when the upper age bound is equal to the diameter d of the input graph G. In other words, it is NP-hard to compute $\kappa(G, d)$. On the other hand, $\kappa(G, 2r)$ can be easily computed in polynomial time, where r is the *radius* of G. Indeed, using the results of Section 2.1, it easily follows that, if G contains (resp. does not contain) a C_4 then $\kappa(G, 2r) = 2n - 4$ (resp. $\kappa(G, 2r) = 2n - 3$). For which values of an upper age bound a, where $d \leq a \leq 2r$, can $\kappa(G, a)$ computed efficiently? In particular, can $\kappa(G, d + 1)$ or $\kappa(G, 2r - 1)$ be computed in polynomial time for every undirected graph G?

With respect to parameterized algorithmics, is MAL FPT with respect to the number k of time-labels?

References

1 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.

2 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.

3 Eleni C. Akrida, George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, Paul G. Spirakis, and Viktor Zamaraev. How fast can we reach a target vertex in stochastic temporal graphs? In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 132, pages 131:1–131:14, 2019.

4 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 148:1–148:14, 2018.

5 Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. In *Proceedings of the 3rd Italian Conference on Algorithms and Complexity (CIAC)*, pages 288–298, 1997.

6 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 149:1–149:14, 2016.

7 Matthias Bentert, Anne-Sophie Himmel, Hendrik Molter, Marco Morik, Rolf Niedermeier, and René Saitenmacher. Listing all maximal k-plexes in temporal graphs. *ACM Journal of Experimental Algorithmics*, 24(1):13:1–13:27, 2019.

8 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.

9 Richard T. Bumby. A problem with telephones. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):13–18, 1981.

10 Sebastian Buß, Hendrik Molter, Rolf Niedermeier, and Maciej Rymar. Algorithmic aspects of temporal betweenness. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2084–2092, 2020.

11 Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *Journal of Computer and System Sciences*, 121:1–17, 2021.

12 Argyrios Deligkas, Eduard Eiben, and George Skretas. Minimizing reachability times on temporal graphs via shifting labels. *CoRR*, abs/2112.08797, 2021. arXiv:2112.08797.

13 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. In *Proceedings of the 34th Conference on Artificial Intelligence (AAAI)*, pages 9810–9817, 2020.

14 S.E. Dreyfus and R.A. Wagner. The steiner problem in graphs. *Networks*, 1:195–207, 1971.

15 Jessica 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.

16 Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021.

17 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 444–455, 2015.

18 Thomas Erlebach and Jakob T. Spooner. Faster exploration of degree-bounded temporal graphs. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 36:1–36:13, 2018.

19 F. Göbel, J.Orestes Cerdeira, and H.J. Veldman. Label-connected graphs and the gossip problem. *Discrete Mathematics*, 87(1):29–40, 1991.

20 Roman Haag, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Feedback edge sets in temporal graphs. *Discrete Applied Mathematics*, 307:65–78, 2022.

21 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Proceedings of the 36th Conference on Artificial Intelligence (AAAI)*, 2022. To appear.

22 Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

23 Petter Holme and Jari Saramäki. *Temporal network theory*, volume 2. Springer, 2019.

24 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

25 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.

26 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4090–4096, 2021.

27 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. *CoRR*, abs/2202.0588, 2022. `arXiv:2202.0588`.

28 George B. Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 657–668, 2013.

29 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.

30 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.

31 George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. *Journal of Computer and System Sciences*, 120:97–115, 2021.

32 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.

33 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, January 2018.

34 Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS '21)*, pages 76:1–76:15, 2021.

35 Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. Graph metrics for temporal networks. In *Temporal Networks*. Springer, 2013.

36 Suhas Thejaswi, Juho Lauri, and Aristides Gionis. Restless reachability in temporal graphs. *CoRR*, abs/2010.08423, 2021. `arXiv:2010.08423`.

37 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

Beyond Value Iteration for Parity Games: Strategy Iteration with Universal Trees

Zhuan Khye Koh
Department of Mathematics, London School of Economics and Political Science, United Kingdom

Georg Loho
Discrete Mathematics and Mathematical Programming, University of Twente, The Netherlands

Abstract

Parity games have witnessed several new quasi-polynomial algorithms since the breakthrough result of Calude et al. (STOC 2017). The combinatorial object underlying these approaches is a *universal tree*, as identified by Czerwiński et al. (SODA 2019). By proving a quasi-polynomial lower bound on the size of a universal tree, they have highlighted a barrier that must be overcome by all existing approaches to attain polynomial running time. This is due to the existence of worst case instances which force these algorithms to explore a large portion of the tree.

As an attempt to overcome this barrier, we propose a strategy iteration framework which can be applied on any universal tree. It is at least as fast as its value iteration counterparts, while allowing one to take bigger leaps in the universal tree. Our main technical contribution is an efficient method for computing the least fixed point of 1-player games. This is achieved via a careful adaptation of shortest path algorithms to the setting of ordered trees. By plugging in the universal tree of Jurdziński and Lazić (LICS 2017), or the Strahler universal tree of Daviaud et al. (ICALP 2020), we obtain instantiations of the general framework that take time $O(mn^2 \log n \log d)$ and $O(mn^2 \log^3 n \log d)$ respectively per iteration.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Logic and verification; Mathematics of computing → Discrete mathematics

Keywords and phrases parity games, strategy iteration, value iteration, progress measure, universal trees

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.63

Related Version *Full Version*: https://arxiv.org/abs/2108.13338

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 757481–ScaleOpt).

Acknowledgements We are grateful for the helpful comments and support by László A. Végh. We would also like to thank Xavier Allamigeon, Nathanaël Fijalkow and Marcin Jurdziński for inspiring discussions.

1 Introduction

A *parity game* is an infinite duration game between two players Even and Odd. It takes place on a sinkless directed graph $G = (V, E)$ equipped with a *priority* function $\pi : V \to \{1, 2, \ldots, d\}$. Let $n = |V|$ and $m = |E|$. The node set V is partitioned into $V_0 \sqcup V_1$ such that nodes in V_0 and V_1 are owned by Even and Odd respectively. The game starts when a token is placed on a node. In each turn, the owner of the current node moves the token along an outgoing arc to the next node, resulting in an infinite walk. If the highest priority occurring infinitely often in this walk is even, then Even wins. Otherwise, Odd wins.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 63; pp. 63:1–63:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

By the positional determinacy of parity games [8], there exists a partition of V into two subsets from which Even and Odd can force a win respectively. The main algorithmic problem of parity games is to determine this partition, or equivalently, to decide the winner given a starting node. This is a notorious problem that lies in NP $\cap$ co-NP [9], and also in UP $\cap$ co-UP [14], with no known polynomial algorithm to date.

Due to its intriguing complexity status, as well as its fundamental role in automata theory and logic [9, 18], parity games have been intensely studied over the past three decades. Prior to 2017, algorithms for solving parity games, e.g. [27, 15, 26, 4, 24, 17, 25, 20, 2], are either exponential or subexponential. In a breakthrough result, Calude et al. [5] gave the first quasi-polynomial algorithm. Since then, many other quasi-polynomial algorithms [10, 16, 19, 22, 3] have been developed. Most of them have been unified by Czerwiński et al. [6] via the concept of a *universal tree*. A universal tree is an ordered tree into which every ordered tree of a certain size can be isomorphically embedded. They proved a quasi-polynomial lower bound on the size of a universal tree.

Value iteration. The starting point of this paper is the classic *progress measure* algorithm [15, 16] for solving parity games. It belongs to a broad class of algorithms called *value iteration* – a well-known method for solving more general games on graphs such as mean payoff games and stochastic games. In value iteration, every node v in G is assigned a value $\mu(v) \in \mathcal{V}$ from some totally ordered set $\mathcal{V}$, and the values are locally improved until we reach the *least fixed point* of a set of operators associated with the game. The set $\mathcal{V}$ is called the *value domain*, which is usually a bounded set of real numbers or integers. For the progress measure algorithm, its value domain is the set of leaves $L(T)$ in a universal tree T. As the values are monotonically improved, the running time is proportional to $|L(T)|$. The first progress measure algorithm of Jurdziński [15] uses a perfect n-ary tree, which runs in exponential time. Its subsequent improvement by Jurdziński and Lazić [16] uses a quasi-polynomial-sized tree, which runs in $n^{\log(d/\log n)+O(1)}$ time.

Despite having good theoretical efficiency, the progress measure algorithm is not robust against its worst-case behaviour. In fact, it is known to realize its worst-case running time on very simple instances. As an example, let (G, π) be an arbitrary instance with maximum priority d, with d being even. For a small odd constant k, if we add two nodes of priority k as shown in Figure 1, then the progress measure algorithm realizes its worst-case running time. This is because the values of those nodes are updated superpolynomially many times.

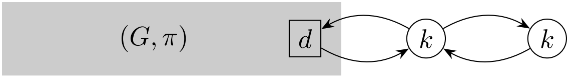


Figure 1 A worst-case construction for the progress measure algorithm. Nodes in V_0 and V_1 are drawn as squares and circles, respectively.

Strategy iteration. A different but related method for solving games on graphs is *strategy iteration*. For a parity game (G, π), a *(positional) strategy* τ for a player (say Odd) is a choice of an outgoing arc from every node in V_1. Removing the unchosen outgoing arcs from every node in V_1 results in a *strategy subgraph* $G_\tau \subseteq G$. A general framework for strategy iteration is given, e.g., in [12]. Following that exposition, to rank the strategies for Odd, one fixes a suitable value domain $\mathcal{V}$ and associates a valuation $\mu : V \to \mathcal{V}$ to each strategy. This induces a partial order over the set of strategies for Odd. Note that most valuations used in the literature can be thought of as fixed points of a set of operators associated with the

1-player game (G_τ, π) for Even. In every iteration, the algorithm maintains a strategy τ for Odd and its corresponding valuation $\mu : V \to \mathcal{V}$. Based on a *pivot rule*, it modifies τ to a better strategy τ', and updates μ to the valuation μ' of τ'. Note that $\mu' \geq \mu$. This process is repeated until we reach the optimal strategy for Odd.

Originally introduced by Hoffman and Karp for stochastic games [13], variants of strategy iteration for parity games have been developed [23, 26, 4, 24]. They usually perform well in practice, but tedious constructions of their worst case (sub)exponential complexity are known [11]. Motivated by the construction of small universal trees [16, 7], a natural question is whether there exists a strategy iteration algorithm with value domain $L(T)$ for a universal tree T. It is not hard to see that with value domain $L(T)$, unfortunately, the fixed point of a 1-player game (G_τ, π) may not be unique. Moreover, in a recent thesis [21], Ohlmann showed that a valuation that is fit for strategy iteration cannot be defined using $L(T)$.

Our contribution. We show that an adaptation of strategy iteration with value domain $L(T)$ is still possible. To circumvent the impossibility result of Ohlmann [21], we slightly alter the strategy iteration framework as follows. After pivoting to a strategy τ' in an iteration, we update the current node labeling μ to the least fixed point of $(G_{\tau'}, \pi)$ that is *pointwise at least* μ. In other words, we force μ to increase (whereas this happens automatically in the previous framework). Since the fixed point of a 1-player game may not be unique, this means that we may encounter a strategy more than once during the course of the algorithm. The motivation of our approach comes from tropical geometry, as discussed in the full version.

To carry out each iteration efficiently, we give a combinatorial method for computing the least fixed point of 1-player games with value domain $L(T)$. It relies on adapting the classic techniques of label-correcting and label-setting from the shortest path problem to the setting of ordered trees. When T is instantiated as a specific universal tree constructed in the literature, we obtain the following running times:

- The universal tree of Jurdziński and Lazić [16] takes $O(mn^2 \log n \log d)$.
- The Strahler universal tree of Daviaud et al. [7] takes $O(mn^2 \log^3 n \log d)$.
- The perfect n-ary tree of height $d/2$ takes $O(d(m + n \log n))$.

The total number of strategy iterations is trivially bounded by $n|L(T)|$, the same bound for the progress measure algorithm. Whereas we do not obtain a strict improvement over previous running time bounds, it is conceivable that our algorithm would terminate in fewer iterations than the progress measure algorithm on most examples. Moreover, our framework provides large flexibility in the choice of pivot rules. Identifying a pivot rule that may provide strictly improved (and possibly even polynomial) running time is left for future research.

Computing the least fixed point of 1-player games. Let (G_τ, π) be a 1-player game for Even, and μ^* be its least fixed point with value domain $L(T)$ for some universal tree T. Starting from $\mu(v) = \min L(T)$ for all $v \in V$, the progress measure algorithm successively lifts the label of a node based on the labels of its out-neighbours until μ^* is reached. However, this is not polynomial in general, even on 1-player games. So, instead of approaching μ^* from below, we approach it from above. This is reminiscent of shortest path algorithms, where node labels form upper bounds on the shortest path distances throughout the algorithm. In a label-correcting method like the Bellman–Ford algorithm, to compute shortest paths to a target node t, the label at t is initialized to 0, while the label at all other nodes is initialized to $+\infty$. By iteratively checking if an arc violates feasibility, the node labels are monotonically decreased. We refer to Ahuja et al. [1] for an overview on label-correcting and label-setting techniques for computing shortest paths.

In our setting, the role of the target node t is replaced by a (potentially empty) set of *even* cycles in G_τ. A cycle is said to be *even* if its maximum priority is even. However, this set is not known to us a priori. To overcome this issue, we define *base nodes* as candidate target nodes. A node $w \in V$ is a *base node* if it *dominates* an even cycle in G_τ, that is, it is a node with the highest priority in the cycle. Note that $\pi(w)$ is even.

To run a label-correcting method, we need to assign initial labels ν to the nodes in G_τ. The presumably obvious choice is to set $\nu(w) \leftarrow \min L(T)$ if w is a base node, and $\nu(w) \leftarrow \top$ otherwise, where $\top$ is bigger than every element in $L(T)$ ($\top$ is analogous to $+\infty$ for real numbers). However, this only works when T is a perfect n-ary tree. For a more complicated universal tree, the number of children at each internal vertex of T is not the same. Hence, it is possible to have $\nu(w) < \mu^*(w)$ for a base node w. We also cannot make $\nu(w)$ too large, as otherwise we may converge to a fixed point that is not pointwise minimal.

To correctly initialize $\nu(w)$ for a base node w, let us consider the cycles dominated by w in G_τ. Every such cycle C induces a subgame (C, π) on which Even wins because C is even. The least fixed point of (C, π) consists of leaves of an ordered tree T_C of height $j := \pi(w)/2$. Initializing $\nu(w)$ essentially boils down to finding such a cycle C with the "narrowest" T_C. To this end, let $\mathcal{T}_j$ be the set of *distinct* subtrees of height j of our universal tree T. We will exploit the fact that $\mathcal{T}_j$ is a poset with respect to the partial order of embeddability. In particular, let $\mathcal{C}_j$ be a set of chains covering $\mathcal{T}_j$, and fix a chain $\mathcal{C}_j^k$ in $\mathcal{C}_j$. We define the *width* of a cycle C as the "width" of the smallest tree in $\mathcal{C}_j^k$ into which T_C is embeddable. Then, we show that our problem reduces to finding a minimum width cycle dominated by w in G_τ.

To solve the latter problem, we construct an arc-weighted *auxiliary digraph* D on the set of base nodes. Every arc uv in D represents a path from base node u to base node v in G_τ, in such a way that minimum bottleneck cycles in D correspond to minimum width cycles in G_τ. It follows that the desired cycle C can be obtained by computing a minimum bottleneck cycle in D containing w. After getting C, we locate the corresponding subtree T' of T into which T_C is embeddable. Then, the label at w is initialized as $\nu(w) \leftarrow \min L(T')$.

With these initial labels, we show that a generic label-correcting procedure returns the desired least fixed point μ^* in $O(mn)$ time. The overall running time of this label-correcting method is dominated by the initialization phase, whose running time is proportional to the size of the chain cover $\mathcal{C}_j$. We prove that the quasi-polynomial universal trees constructed in the literature [16, 7] admit small chain covers. Using this result, we then give efficient implementations of our method for these trees.

In the full version of the paper, we also develop a label-setting method for computing μ^*, which is faster but only applicable when T is a perfect n-ary tree. Unlike the label-correcting approach, in a label-setting method such as Dijkstra's algorithm, the label of a node is fixed in each iteration. In the shortest path problem, Dijkstra's algorithm selects a node with the smallest label to be fixed in every iteration. When working with labels given by the leaves of a universal tree, this criterion does not work anymore. Let H be the subgraph of G_τ obtained by deleting all the base nodes. For $p \in \mathbb{N}$, let H_p be the subgraph of H induced by the nodes with priority at most p. We construct a suitable potential function by interlacing each node label with a tuple that encodes the topological orders in $H_2, H_4, \ldots$. In every iteration, a node with the smallest potential is selected, and its label is fixed.

Paper organization. In Section 2, we introduce notation and provide the necessary preliminaries on parity games and universal trees. Section 3 contains our strategy iteration framework based on universal trees. In Section 4, we give a label-correcting method for computing the least fixed point of 1-player games. The label-setting method, on the other hand, is given in the full version. Missing proofs can also be found in the full version.

2 Preliminaries on Parity Games and Universal Trees

For $d \in \mathbb{N}$, let $[d] = \{1, 2, \ldots, d\}$. For a graph G, we use $V(G)$ as its vertex set, and $E(G)$ as its edge set. A parity game instance is given by (G, π), where $G = (V, E)$ is a sinkless directed graph with $V = V_0 \sqcup V_1$, and $\pi : V \to [d]$ is a priority function. Without loss of generality, we may assume that d is even. In this paper, we are only concerned with positional strategies. A *strategy* for Odd is a function $\tau : V_1 \to V$ such that $v\tau(v) \in E$ for all $v \in V_1$. Its *strategy subgraph* is $G_\tau = (V, E_\tau)$, where $E_\tau := \{vw \in E : v \in V_0\} \cup \{v\tau(v) : v \in V_1\}$. A strategy for Even and its strategy subgraph are defined analogously. We always denote a strategy for Even as σ, and a strategy for Odd as τ. If we fix a strategy τ for Odd, the resulting instance (G_τ, π) is a *1-player game* for Even.

For the sake of brevity, we overload the priority function π as follows. Given a subgraph $H \subseteq G$, let $\pi(H)$ be the highest priority in H. The subgraph H is said to be *even* if $\pi(H)$ is even, and *odd* otherwise. For a fixed π, we denote by $\Pi(H)$ the set of nodes with the highest priority in H. If $v \in \Pi(H)$, we say that v *dominates* H. For $p \in [d]$, H_p refers to the subgraph of H induced by nodes with priority at most p. For a node v, let $\delta^-_H(v)$ and $\delta^+_H(v)$ be the incoming and outgoing arcs of v in H respectively. Similarly, let $N^-_H(v)$ and $N^+_H(v)$ be the in-neighbors and out-neighbors of v in H respectively. When H is clear from context, we will omit it from the subscripts.

The win of a player can be certified by *node labels* from a *universal tree*, as stated in Theorem 3. We give the necessary background for this now.

2.1 Ordered Trees and Universal Trees

An *ordered tree* T is a prefix-closed set of tuples, whose elements are drawn from a linearly ordered set M. The linear order of M lexicographically extends to T. Equivalently, T can be thought of as a rooted tree, whose root we denote by r. Under this interpretation, elements in M correspond to the branching directions at each vertex of T (see Figures 2 and 3 for examples). Every tuple then corresponds to a vertex $v \in V(T)$. This is because the tuple can be read by traversing the unique r-v path in T. Observe that v is an h-tuple if and only if v is at depth h in T. In particular, r is the empty tuple.

In this paper, we always use the terms "vertex" and "edge" when referring to an ordered tree T. The terms "node" and "arc" are reserved for the game graph G.

Given an ordered tree T of height h, let $L(T)$ be the set of leaves in T. For convenience, we assume that every leaf in T is at depth h throughout. The tuple representing a leaf $\xi \in L(T)$ is denoted as $\xi = (\xi_{2h-1}, \xi_{2h-3}, \ldots, \xi_1)$, where $\xi_i \in M$ for all i. We refer to ξ_{2h-1} as the *first* component of ξ, even though it has index $2h - 1$. For a fixed $p \in [2h]$, the *p-truncation* of ξ is $\xi|_p := \begin{cases} (\xi_{2h-1}, \xi_{2h-3}, \ldots, \xi_{p+1}), & \text{if } p \text{ is even} \\ (\xi_{2h-1}, \xi_{2h-3}, \ldots, \xi_p), & \text{if } p \text{ is odd.} \end{cases}$
In other words, the p-truncation of a tuple is obtained by deleting the components with index less than p. Note that a truncated tuple is an ancestor of the untruncated tuple in T.

▶ **Definition 1.** *Given ordered trees T and T', we say that T* embeds into *T' (denoted $T \sqsubseteq T'$) if there exists an* injective *and* order-preserving *homomorphism from T to T' such that leaves in T are mapped to leaves in T'. Formally, this is an injective function $f : V(T) \to V(T')$ which satisfies the following properties:*

1. *For all $u, v \in V(T)$, $uv \in E(T)$ implies $f(u)f(v) \in E(T')$;*
2. *For all $u, v \in V(T)$, $u \le v$ implies $f(u) \le f(v)$.*
3. *$f(u) \in L(T')$ for all $u \in L(T)$.*

We write $T \equiv T'$ if $T \sqsubseteq T'$ and $T' \sqsubseteq T$. Also, $T \sqsubset T'$ if $T \sqsubseteq T'$ and $T \not\equiv T'$.

In the definition above, since f is order-preserving, the children of every vertex in T are mapped to the children of its image injectively such that their order is preserved. As an example, the tree in Figure 3 embeds into the tree in Figure 2. It is easy to verify that $\sqsubseteq$ is a partial order on the set of all ordered trees.

▶ **Definition 2.** *An* (ℓ, h)-universal tree *is an ordered tree T' of height h such that $T \sqsubseteq T'$ for every ordered tree T of height h and with at most ℓ leaves, all at depth exactly h.*

The simplest example of an (ℓ, h)-universal tree is the perfect ℓ-ary tree of height h, which we call a *perfect universal tree*. The linearly ordered set M for this tree can be chosen as $\{0, 1, \ldots, \ell - 1\}$ (see Figure 2 for an example). It has ℓ^h leaves, which grows exponentially with h. Jurdziński and Lazić [16] constructed an (ℓ, h)-universal tree with at most $\ell^{\log h + O(1)}$ leaves, which we call a *succinct universal tree*. In this tree, every leaf ξ corresponds to an h-tuple of binary strings with at most $\lfloor \log(\ell) \rfloor$ bits in total[1]. We use $|\xi|$ and $|\xi_i|$ to denote the total number of bits in ξ and ξ_i respectively. The linearly ordered set M for this tree consists of finite binary strings, where $\varepsilon \in M$ is the empty string (see Figure 3 for an example). For any pair of binary strings $s, s' \in M$ and a bit b, the linear order on M is defined as $0s < \varepsilon < 1s'$ and $bs < bs' \iff s < s'$.

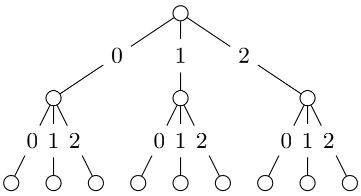


Figure 2 The perfect (3,2)-universal tree.

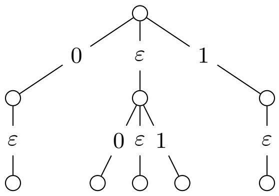


Figure 3 The succinct (3,2)-universal tree.

2.2 Node Labelings from Universal Trees

Let (G, π) be a parity game instance and T be an ordered tree of height $d/2$. We augment the set of leaves with an extra *top* element $\top$, denoted $\bar{L}(T) := L(T) \cup \{\top\}$, such that $\top > v$ for all $v \in V(T)$. We also set $\top|_p := \top$ for all $p \in [d]$. A function $\mu : V \to \bar{L}(T)$ which maps the nodes in G to $\bar{L}(T)$ is called a *node labeling*. For a subgraph H of G, we say that μ is *feasible in H* if there exists a strategy $\sigma : V_0 \to V$ for Even with $v\sigma(v) \in E(H)$ whenever $\delta^+_H(v) \neq \emptyset$, such that the following condition holds for every arc vw in $H \cap G_\sigma$:

- If $\pi(v)$ is even, then $\mu(v)|_{\pi(v)} \geq \mu(w)|_{\pi(v)}$.
- If $\pi(v)$ is odd, then $\mu(v)|_{\pi(v)} > \mu(w)|_{\pi(v)}$ or $\mu(v) = \mu(w) = \top$.

An arc vw which does not satisfy the condition above is called *violated* (with respect to μ). On the other hand, if $\mu(v)$ is the smallest element in $\bar{L}(T)$ such that vw is non-violated, then vw is said to be *tight*. Any arc which is neither tight nor violated is called *loose*. We say that a subgraph is tight if it consists of tight arcs.

In the literature, a node labeling which is feasible in G is also called a *progress measure*. The node labeling given by $\mu(v) = \top$ for all $v \in V$ is trivially feasible in G. However, we are primarily interested in progress measures with minimal top support, i.e. such that the set of nodes having label $\top$ is inclusion-wise minimal.

[1] A slightly looser bound of $\lceil \log \ell \rceil$ was derived in [16, Lemma 1]. It can be strengthened to $\lfloor \log \ell \rfloor$ with virtually no change in the proof.

▶ **Theorem 3** ([15, Corollaries 7–8]). *Given an $(n, d/2)$-universal tree T, let $\mu^* : V \to \bar{L}(T)$ be a node labeling which is feasible in G and has minimal top support. Then, Even wins from $v \in V$ if and only if $\mu^*(v) \neq \top$.*

The above theorem formalizes the following intuition: nodes with smaller labels are more advantageous for Even to play on. Note that if μ is a minimal node labeling which is feasible in G, i.e. μ' is infeasible in G for all $\mu' < \mu$, then there exists a strategy σ for Even such that $v\sigma(v)$ is tight for all $v \in V_0$. The next observation is well-known (see, e.g., [16, Lemma 2]) and follows directly from the definition of feasibility.

▶ **Lemma 4** (Cycle Lemma). *Let μ be a node labeling and C be a cycle such that $\mu(v) \neq \top$ for all $v \in V(C)$. If μ is feasible in C, then C is even. If C is also tight, then $\mu(v) = \mu(w)$ for all $v, w \in \Pi(C)$.*

We assume to have access to the following algorithmic primitive, whose running time we denote by $\gamma(T)$. Its implementation depends on the ordered tree T. For instance, $\gamma(T) = O(d)$ if T is a perfect $(n, d/2)$-universal tree. If T is a succinct $(n, d/2)$-universal tree, Jurdziński and Lazić [16, Theorem 7] showed that $\gamma(T) = O(\log n \log d)$.

Tighten(μ, vw)

Given a node labeling $\mu : V \to \bar{L}(T)$ and an arc $vw \in E$, return the unique element $\xi \in \bar{L}(T)$ such that vw is tight after setting $\mu(v)$ to ξ.

Given a node labeling $\mu : V \to \bar{L}(T)$ and an arc $vw \in E$, let $\mathrm{lift}(\mu, vw)$ be the smallest element $\xi \in \bar{L}(T)$ such that $\xi \geq \mu(v)$ and vw is not violated after setting $\mu(v)$ to ξ. Observe that if vw is violated, $\mathrm{lift}(\mu, vw)$ is given by Tighten(μ, vw). Otherwise, it is equal to $\mu(v)$. Hence, it can be computed in $\gamma(T)$ time.

Let $\mathcal{L}$ be the finite lattice of node labelings mapping V to $\bar{L}(T)$. For a sinkless subgraph $H \subseteq G$, consider the following operators. For every node $v \in V_0$, define $\mathrm{Lift}_v : \mathcal{L} \times V \to \bar{L}(T)$ as $\mathrm{Lift}_v(\mu, u) := \min_{vw \in E(H)} \mathrm{lift}(\mu, vw)$ if $u = v$, and $\mu(u)$ otherwise. For every arc $vw \in E(H)$ where $v \in V_1$, define $\mathrm{Lift}_{vw} : \mathcal{L} \times V \to \bar{L}(T)$ as $\mathrm{Lift}_{vw}(\mu, u) := \mathrm{lift}(\mu, vw)$ if $u = v$, and $\mu(u)$ otherwise. We denote $\mathcal{H}^{\uparrow} = \{\mathrm{Lift}_v : v \in V_0\} \cup \{\mathrm{Lift}_{vw} : v \in V_1\}$ as the operators in H. Since they are inflationary and monotone, for any $\mu \in \mathcal{L}$, the *least* simultaneous fixed point of $\mathcal{H}^{\uparrow}$ that is *pointwise at least* μ exists. It is denoted as $\mu^{\mathcal{H}^{\uparrow}}$. Note that a node labeling is a simultaneous fixed point of $\mathcal{H}^{\uparrow}$ if and only if it is feasible in H. The *progress measure algorithm* [15, 16] is an iterative application of the operators in $\mathcal{G}^{\uparrow}$ to μ to obtain $\mu^{\mathcal{G}^{\uparrow}}$.

3 Strategy Iteration with Tree Labels

In this section, we present a strategy iteration algorithm (Algorithm 1) whose pivots are guided by a universal tree. It takes as input an instance (G, π), a universal tree T, and an initial strategy τ_1 for Odd. Throughout, it maintains a node labeling $\mu : V \to \bar{L}(T)$, initialized as the least simultaneous fixed point of $\mathcal{G}^{\uparrow}_{\tau_1}$. At the start of every iteration, the algorithm maintains a strategy τ for Odd, and a node labeling $\mu : V \to \bar{L}(T)$ which is feasible in G_τ. Furthermore, there are no loose arcs in G_τ with respect to μ. So, every arc in G_τ is either tight (usable by Even in her counterstrategy σ) or violated (not used by Even). Note that our initial node labeling satisfies these conditions with respect to τ_1.

For $v \in V_1$, we call a violated arc $vw \in E$ with respect to μ *admissible* (as it admits Odd to perform an improvement). If there are no admissible arcs in G, then the algorithm terminates. In this case, μ is feasible in G. Otherwise, Odd pivots to a new strategy τ' by

Algorithm 1 Strategy iteration with tree labels: (G, π) instance, T universal tree, τ_1 initial strategy for Odd.

```
procedure StrategyIteration($(G, \pi), T, \tau_1$)
    $\mu(v) \leftarrow \min L(T)\ \forall v \in V$
    $\tau \leftarrow \tau_1$, $\mu \leftarrow \mu^{\mathcal{G}^{\uparrow}_{\tau}}$
    while $\exists$ an admissible arc in $G$ with respect to $\mu$ do
        Pivot to a strategy $\tau'$ by selecting admissible arc(s)        ▷ requires a pivot rule
        $\tau \leftarrow \tau'$, $\mu \leftarrow \mu^{\mathcal{G}^{\uparrow}_{\tau}}$
    return $\tau$, $\mu$
```

switching to admissible arc(s). The choice of which admissible arc(s) to pick is governed by a *pivot rule*. Then, μ is updated to $\mu^{\mathcal{G}^{\uparrow}_{\tau'}}$. Due to the minimality of $\mu^{\mathcal{G}^{\uparrow}_{\tau'}}$, there are no loose arcs in $G_{\tau'}$ with respect to $\mu^{\mathcal{G}^{\uparrow}_{\tau'}}$, so this invariant continues to hold in the next iteration.

The correctness of Algorithm 1 follows from the Knaster–Tarski Theorem. We remark that a strategy τ may occur more than once during the course of the algorithm, as mentioned in the description of strategy iteration in Section 1. This is because the fixed points of $\mathcal{G}^{\uparrow}_{\tau}$ are not necessarily unique. See Figure 4 for an example run with a succinct universal tree.

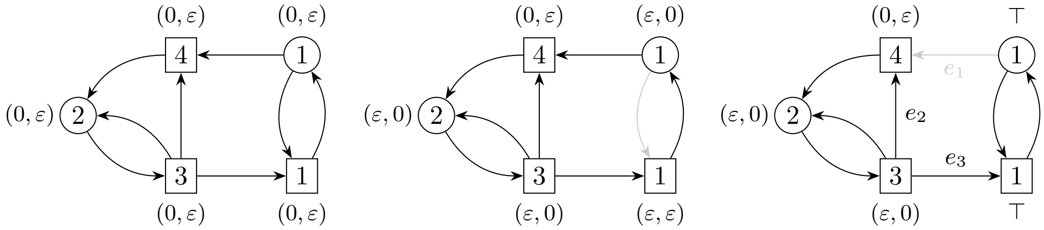


Figure 4 An example run of Algorithm 1 with the succinct (3,2)-universal tree. The left figure depicts a game instance (nodes in V_0 and V_1 are drawn as squares and circles respectively). The next two figures show Odd's strategy and the node labeling at the start of Iteration 1 and 2. Arcs not selected by Odd are greyed out. In the right figure, e_1 is loose, e_2 is tight, and e_3 is violated.

4 Computing the Least Fixed Point of 1-Player Games

Let (G_τ, π) be a 1-player game for Even, and let $\mu \in \mathcal{L}$ be a node labeling such that there are no loose arcs in G_τ. In this section, we develop an efficient method for computing $\mu^{\mathcal{G}^{\uparrow}_{\tau}}$. We know that applying the operators in $\mathcal{G}^{\uparrow}_{\tau}$ to μ is not polynomial in general. So, we will approach $\mu^{\mathcal{G}^{\uparrow}_{\tau}}$ from above instead.

Given a node labeling $\nu : V \to \bar{L}(T)$ and an arc $vw \in E$, let $\mathrm{drop}(\nu, vw)$ be the largest element $\xi \in \bar{L}(T)$ such that $\xi \leq \nu(v)$ and vw is not loose after setting $\nu(v)$ to ξ. Observe that if vw is loose, then $\mathrm{drop}(\nu, vw)$ is given by $\textsc{Tighten}(\nu, vw)$. Otherwise, it is equal to $\nu(v)$. Hence, it can be computed in $\gamma(T)$ time.

We are ready to define the deflationary counterpart of Lift_{vw}. For every arc $vw \in E_\tau$, define the operator $\mathrm{Drop}_{vw} : \mathcal{L} \times V \to \bar{L}(T)$ as $\mathrm{Drop}_{vw}(\nu, u) := \mathrm{drop}(\nu, vw)$ if $u = v$, and $\nu(v)$ otherwise. For a subgraph $H \subseteq G_\tau$, we denote $\mathcal{H}^{\downarrow} = \{\mathrm{Drop}_e : e \in E(H)\}$ as the operators in H. Since they are deflationary and monotone, for any $\nu \in \mathcal{L}$, the *greatest* simultaneous fixed point of $\mathcal{H}^{\downarrow}$ that is *pointwise at most* ν exists. It is denoted as $\nu^{\mathcal{H}^{\downarrow}}$. Note that a node labeling is a simultaneous fixed point of $\mathcal{H}^{\downarrow}$ if and only if there are no loose arcs in H with respect to it.

Our techniques are inspired by the methods of *label-correcting* and *label-setting* for the shortest path problem. In the shortest path problem, we have a designated target node t whose label is initialized to 0. For us, the role of t is replaced by a (potentially empty) set of even cycles in G_τ, which we do not know a priori. So, we define a set of candidates nodes called *base nodes*, whose labels need to be initialized properly.

▶ **Definition 5.** *Given a 1-player game (G_τ, π) for Even, we call $v \in V$ a* base node *if $v \in \Pi(C)$ for some even cycle C in G_τ. Denote $B(G_\tau)$ as the set of base nodes in G_τ.*

The base nodes can be found by recursively decomposing G_τ into strongly connected components. Initially, for each strongly connected component K of G_τ, we delete $\Pi(K)$. If $\pi(K)$ is even and $|V(K)| > 1$, then $\Pi(K)$ are base nodes and we collect them. Otherwise, we ignore them. Then, we are left with a smaller subgraph of G, so we repeat the process. Using Tarjan's strongly connected components algorithm, this procedure takes $O(dm)$ time.

In the next subsection, we develop a label-correcting method for computing $\mu^{\mathcal{G}_\tau^\uparrow}$, and apply it to the quasi-polynomial universal trees constructed in the literature [16, 7]. The label-setting method, which is faster but only applicable to perfect universal trees, is deferred to the full version.

4.1 Label-Correcting Method

The Bellman–Ford algorithm for the shortest path problem is a well-known implementation of the generic label-correcting method [1]. We start by giving its analogue for ordered trees. Algorithm 2 takes as input a 1-player game (G_τ, π) for Even and a node labeling $\nu : V \to \bar{L}(T)$ from some ordered tree T. Like its classical version for shortest paths, the algorithm runs for $n - 1$ iterations. In each iteration, it replaces the tail label of every arc $e \in E_\tau$ by $\mathrm{drop}(\nu, e)$. Clearly, the running time is $O(mn\gamma(T))$. Moreover, if ν' is the returned node labeling, then $\nu' \geq \nu^{\mathcal{G}_\tau^\downarrow}$.

Algorithm 2 Bellman–Ford: (G_τ, π) 1-player game for Even, $\nu : V \to \bar{L}(T)$ node labeling from an ordered tree T.

```
procedure BellmanFord($((G, \pi), \nu)$)
    for $i = 1$ to $n - 1$ do
        for all $vw \in E$ do                          ▷ In any order
            $\nu(v) \leftarrow \mathrm{drop}(\nu, vw)$
    return $\nu$
```

Recall that we have a node labeling $\mu \in \mathcal{L}$ such that G_τ does not have loose arcs, and our goal is to compute $\mu^{\mathcal{G}_\tau^\uparrow}$. We first state a sufficient condition on the input node labeling ν such that Algorithm 2 returns $\mu^{\mathcal{G}_\tau^\uparrow}$. In the shortest path problem, we set $\nu(t) = 0$ at the target node t, and $\nu(v) = \infty$ for all $v \in V \setminus \{t\}$. When working with node labels given by an ordered tree, one has to ensure that the algorithm does not terminate with a fixed point larger than $\mu^{\mathcal{G}_\tau^\uparrow}$, motivating the following definition.

▶ **Definition 6.** *Given a node labeling $\mu \in \mathcal{L}$, the* threshold label *of a base node v is*

$$\hat{\mu}(v) := \min_{\tilde{\mu} \in \mathcal{L}} \{\tilde{\mu}(v) : \tilde{\mu}(v) \geq \mu(v) \text{ and } \tilde{\mu} \text{ is feasible in a cycle dominated by } v\} \quad .$$

The next lemma follows directly from the pointwise minimality of $\mu^{\mathcal{G}_\tau^\uparrow}(v)$.

▶ **Lemma 7.** *Let $\mu \in \mathcal{L}$ be a node labeling such that G_τ does not have loose arcs. For every base node $v \in B(G_\tau)$, we have $\widehat{\mu}(v) \geq \mu^{\mathcal{G}_\tau^\uparrow}(v)$.*

The next theorem shows that if we initialize the base nodes with their corresponding threshold labels, then Algorithm 2 returns $\mu^{\mathcal{G}_\tau^\uparrow}$. Even more, it suffices to have an initial node labeling $\nu \in \mathcal{L}$ such that $\mu^{\mathcal{G}_\tau^\uparrow}(v) \leq \nu(v) \leq \widehat{\mu}(v)$ for all $v \in B(G_\tau)$. For the other nodes $v \notin B(G_\tau)$, we can simply set $\nu(v) \leftarrow \top$.

▶ **Theorem 8.** *Let $\mu \in \mathcal{L}$ be a node labeling such that G_τ does not have loose arcs. Given $\nu \in \mathcal{L}$ where $\nu \geq \mu^{\mathcal{G}_\tau^\uparrow}$ and $\nu(v) \leq \widehat{\mu}(v)$ for all $v \in B(G_\tau)$, Algorithm 2 returns $\mu^{\mathcal{G}_\tau^\uparrow}$.*

Our strategy for computing such a ν is to find the cycles in Definition 6. In particular, for every base node $v \in B(G_\tau)$, we aim to find a cycle C dominated by v such that $\widehat{\mu}(v)$ can be extended to a node labeling that is feasible in C. To accomplish this goal, we first introduce the notion of *width* in Section 4.1.1, which allows us to evaluate how "good" a cycle is. It is defined using chains in the poset of subtrees of T, where the partial order is given by $\sqsubseteq$. Then, in Section 4.1.2, we show how to obtain the desired cycles by computing minimum bottleneck cycles on a suitably defined auxiliary digraph.

4.1.1 Width from a Chain of Subtrees in T

Two ordered trees T' and T'' are said to be *distinct* if $T' \not\equiv T''$ (not isomorphic in the sense of Definition 1). Let h be the height of our universal tree T. For $0 \leq j \leq h$, denote $\mathcal{T}_j$ as the set of distinct (whole) subtrees rooted at the vertices of depth $h - j$ in T. For example, $\mathcal{T}_h = \{T\}$, while $\mathcal{T}_0$ contains the trivial tree with a single vertex. Since we assumed that all the leaves in T are at the same depth, every tree in $\mathcal{T}_j$ has height j. We denote $\mathcal{T} = \cup_{j=0}^{h} \mathcal{T}_j$ as the union of all these subtrees. The sets $\mathcal{T}$ and $\mathcal{T}_j$ form posets with respect to the partial order $\sqsubseteq$. The next definition is the usual chain cover of a poset, where we additionally require that the chains form an indexed tuple instead of a set.

▶ **Definition 9.** *For $0 \leq j \leq h$, let $\mathcal{C}_j = (\mathcal{C}_j^0, \mathcal{C}_j^1, \dots, \mathcal{C}_j^\ell)$ be a tuple of chains in the poset $(\mathcal{T}_j, \sqsubseteq)$. We call $\mathcal{C}_j$ a* cover of $\mathcal{T}_j$ *if $\cup_{k=0}^{\ell} \mathcal{C}_j^k = \mathcal{T}_j$. A* cover of $\mathcal{T}$ *is a tuple $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_h)$ where $\mathcal{C}_j$ is a cover of $\mathcal{T}_j$ for all $0 \leq j \leq h$. We refer to $\mathcal{C}_j$ as the* jth-subcover *of $\mathcal{C}$. Given a cover $\mathcal{C}$ of $\mathcal{T}$, we denote the trees in the chain $\mathcal{C}_j^k$ as $T_{0,j}^k \sqsubset T_{1,j}^k \sqsubset \cdots \sqsubset T_{|\mathcal{C}_j^k|-1,j}^k$.*

An example of an ordered tree with its cover is given in Figure 5. We are ready to introduce the key concept of this subsection.

▶ **Definition 10.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. Let H be a subgraph of G_τ and $j = \lceil \pi(H)/2 \rceil$. For a fixed chain $\mathcal{C}_j^k$ in $\mathcal{C}_j$, the* kth-width *of H, denoted $\alpha_{\mathcal{C}}^k(H)$, is the smallest integer $i \geq 0$ such that there exists a node labeling $\nu : V(H) \to L(T_{i,j}^k)$ which is feasible in H. If i does not exist, then $\alpha_{\mathcal{C}}^k(H) = \infty$.*

Note that $T_{i,j}^k$ is the $(i+1)$-th smallest tree in the chain $\mathcal{C}_j^k$. We are mainly interested in the case when H is a cycle, and write $\alpha^k(H)$ whenever the cover $\mathcal{C}$ is clear from context. Observe that the definition above requires $\nu(v) \neq \top$ for all $v \in V(H)$. Hence, an odd cycle has infinite kth-width by the Cycle Lemma. As $(\mathcal{C}_j^k, \sqsubseteq)$ is a chain, for all finite $i \geq \alpha^k(H)$, there exists a node labeling $\nu : V(H) \to L(T_{i,j}^k)$ which is feasible in H. The next lemma illustrates the connection between the kth-width of an even cycle and its path decomposition.

▶ **Lemma 11.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. For an even cycle C, let $\Pi(C) = \{v_1, v_2, \dots, v_\ell\}$ and $j = \pi(C)/2$. Decompose C into arc-disjoint paths $P_1, P_2, \dots, P_\ell$ such that each P_i ends at v_i. Then, $\alpha^k(C) = \max_{i \in [\ell]} \alpha^k(P_i)$ for all $0 \leq k < |\mathcal{C}_j|$.*

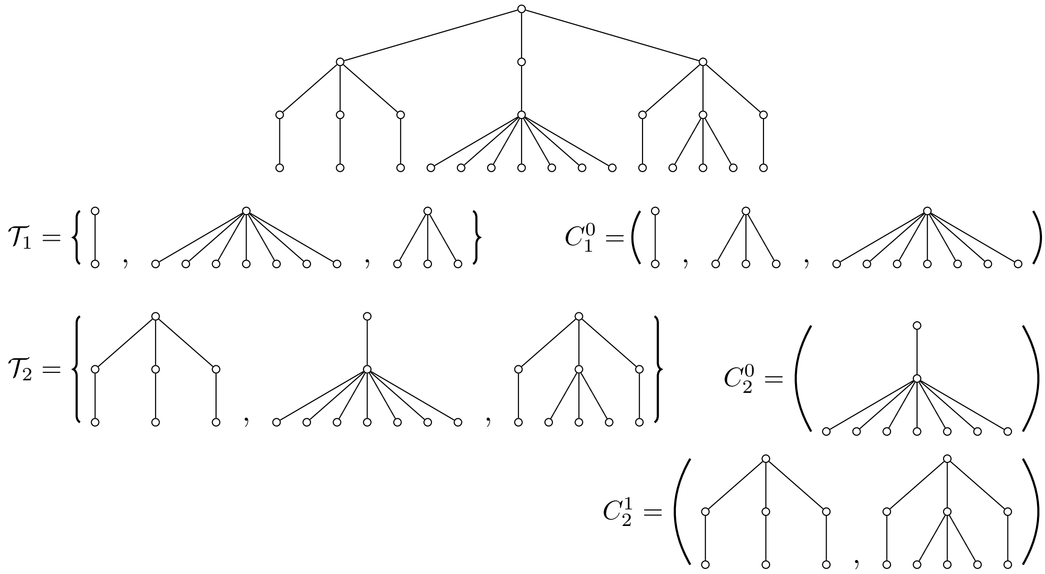


Figure 5 An ordered tree T of height 3, and a cover $\mathcal{C}_j$ of $\mathcal{T}_j$ for all $0 < j < 3$. Recall that $\mathcal{T}_j$ is the set of distinct subtrees of T rooted at depth $3 - j$, while $\mathcal{C}_j^k$ is the kth chain in $\mathcal{C}_j$.

For a base node $v \in B(G_\tau)$, let us consider the cycles in G_τ which are dominated by v. Among them, we are interested in finding one with the smallest kth-width. So, we extend the notion of kth-width to base nodes in the following way.

▶ **Definition 12.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. Let $v \in B(G_\tau)$ be a base node and $j = \pi(v)/2$. For $0 \leq k < |\mathcal{C}_j|$, define the* kth-width *of v as $\alpha^k_{\mathcal{C}}(v) := \min\left\{\alpha^k_{\mathcal{C}}(C) : C \text{ is a cycle dominated by } v\right\}$.*

Again, we write $\alpha^k(v)$ whenever $\mathcal{C}$ is clear from context. Observe that $T^k_{\alpha^k(v),\pi(v)/2}$ is the smallest tree in the chain $\mathcal{C}^k_{\pi(v)/2}$ which can encode a node labeling that is feasible on some cycle dominated by v.

Given a leaf $\xi \in L(T)$ and integers $i, j, k \in \mathbb{Z}_{\geq 0}$, the following subroutine locates a subtree of T that is a member of the chain $\mathcal{C}^k_j$ and into which $T^k_{i,j}$ is embeddable.

Raise(ξ, i, j, k)

Given a leaf $\xi \in L(T)$ and integers $i, j, k \in \mathbb{Z}_{\geq 0}$, return the smallest leaf $\xi' \in L(T)$ such that (1) $\xi' \geq \xi$; and (2) ξ' is the smallest leaf in the subtree $T^k_{i',j}$ for some $i' \geq i$. If ξ' does not exist, then return $\top$.

Now, fix a base node $v \in B(G_\tau)$. For any $0 \leq k < |\mathcal{C}_{\pi(v)/2}|$, notice that the label returned by Raise$(\mu(v), \alpha^k(v), \frac{\pi(v)}{2}, k)$ is an upper bound on the threshold label $\widehat{\mu}(v)$. The smallest such label over all k is precisely the threshold label $\widehat{\mu}(v)$, as the next lemma shows.

▶ **Lemma 13.** *Fix $v \in B(G_\tau)$. Let ξ^k be the label returned by* Raise$(\mu(v), \alpha^k(v), \frac{\pi(v)}{2}, k)$ *for all $0 \leq k < |\mathcal{C}_{\pi(v)/2}|$. If there are no loose arcs in G_τ with respect to μ, then $\widehat{\mu}(v) = \min_k \xi^k$.*

The necessary number of chains in the subcover $\mathcal{C}_{\pi(v)/2}$ can be large if T is an arbitrary ordered tree. Fortunately, the universal trees constructed in the literature admit covers with small subcovers. In the full version, we prove that a succinct (n, h)-universal tree has a cover with only 1 chain per subcover, whereas a succinct Strahler (n, h)-universal tree (introduced by Daviaud et al. [7]) has a cover with at most $\log n$ chains per subcover.

Let $\rho(T, \mathcal{C})$ denote the running time of Raise. We provide efficient implementations of Raise for succinct universal trees and succinct Strahler universal trees in the full version. They have the same running time as Tighten, i.e., $\rho(T, \mathcal{C}) = O(\log n \log h)$.

4.1.2 Estimating the Width of Base Nodes

In light of the previous discussion, we can now focus on computing the kth-width of a base node $w \in B(G_\tau)$. Fix a $0 \leq k < |\mathcal{C}_{\pi(w)/2}|$. Since we ultimately need a label that lies between $\mu^{\mathcal{G}_\tau^\uparrow}(w)$ and $\widehat{\mu}(w)$ in order to initialize Algorithm 2, it suffices to compute a "good" under-estimation of $\alpha^k(w)$. In this subsection, we reduce this problem to computing a minimum bottleneck cycle in an auxiliary digraph D with nonnegative arc costs $c^k \geq 0$.

For a base node $w \in B(G_\tau)$, let K_w denote the strongly connected component containing w in $(G_\tau)_{\pi(w)}$, the subgraph of G_τ induced by nodes with priority at most $\pi(w)$. Let $K'_w \subseteq K_w$ be the subgraph obtained by deleting the incoming arcs $\delta^-(v)$ for all $v \in \Pi(K_w) \setminus \{w\}$. Then, we define J_w as the subgraph of K'_w induced by those nodes which can reach w in K'_w. These are the nodes which can reach w in K_w without encountering an intermediate node of priority $\pi(w)$.

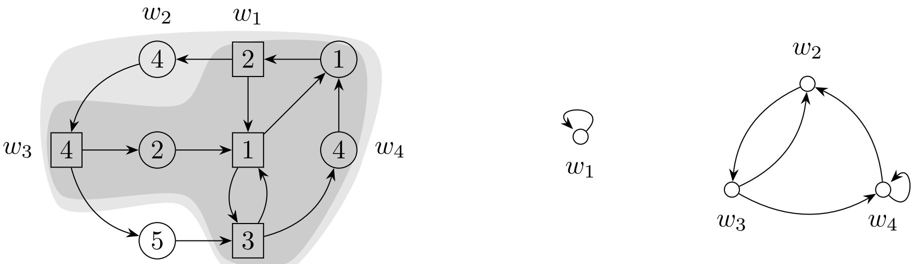


Figure 6 An example of a 1-player game (G_τ, π) for Even is given on the left, with its auxiliary digraph D on the right. Nodes in V_0 and V_1 are drawn as squares and circles respectively. Base nodes are labeled as w_1, w_2, w_3, w_4. The light gray region is K_{w_4}, while the dark gray region is J_{w_4}.

The auxiliary digraph D is constructed as follows. Its node set is $B(G_\tau)$. For every ordered pair (v, w) of base nodes where $\pi(v) = \pi(w)$, add the arc vw if v has an outgoing arc in J_w. Note that if $(v, w) \in D$, then v can reach w by only seeing smaller priorities on the intermediate nodes. As ordered pairs of the form (v, v) are also considered, D may contain self-loops. Observe that D is a disjoint union of strongly connected components, each of which consists of base nodes with the same priority (see Figure 6 for an example). For $w \in B(G_\tau)$, we denote D_w as the component in D which contains w.

To finish the description of D, it is left to assign the arc costs c^k. Note that the graph structure of D is independent of k. We give a range in which the cost of each arc should lie. Fix a base node $w \in B(G_\tau)$ and let $j = \pi(w)/2$. Recall that $\mathcal{J}_w^\downarrow = \{\text{Drop}_e : e \in E(J_w)\}$ is the set of Drop operators in the subgraph $J_w \subseteq G_\tau$. For each $0 \leq i < |\mathcal{C}_j^k|$, let $\lambda_{i,w}^k : V(J_w) \to \bar{L}(T_{i,j}^k)$ be the greatest simultaneous fixed point of $\mathcal{J}_w$ subject to $\lambda_{i,w}^k(w) = \min L(T_{i,j}^k)$. Then, for each arc $vw \in E(D)$, the lower and upper bounds of $c^k(vw)$ are given by

$$\begin{aligned} \underline{c}^k(vw) &:= \min\left\{i : \lambda_{i,w}^k(u) \neq \top \text{ for some } u \in N_{J_w}^+(v)\right\} \\ \bar{c}^k(vw) &:= \min\left\{\alpha^k(P) : P \text{ is a } u\text{-}w \text{ path in } J_w \text{ where } u \in N_{J_w}^+(v)\right\} \end{aligned} \tag{1}$$

respectively. The lower bound $\underline{c}^k(vw)$ is the smallest integer $i \geq 0$ such that the greatest simultaneous fixed point $\lambda_{i,w}^k$ assigns a non-top label to an out-neighbor of v in J_w. On the other hand, the upper bound $\bar{c}^k(vw)$ is the minimum kth-width of a path from an out-neighbor of v to w in J_w. Note that these quantities could be equal to $+\infty$.

▶ **Lemma 14.** *For every arc $vw \in E(D)$, we have $\underline{c}^k(vw) \le \overline{c}^k(vw)$.*

For a cycle C in D, its (bottleneck) c^k-*cost* is defined as $c^k(C) := \max_{e \in E(C)} c^k(e)$. Note that self-loops in D are considered cycles. The next theorem allows us to obtain the desired initial node labeling ν for Algorithm 2 by computing minimum bottleneck cycles in D.

▶ **Theorem 15.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. Let $\mu \in \mathcal{L}$ be a node labeling such that G_τ does not have loose arcs. For a base node w, let c^k be arc costs in D_w such that $\underline{c}^k \le c^k \le \overline{c}^k$ for all $0 \le k < |\mathcal{C}_{\pi(w)/2}|$. For each k, let i^k be the minimum c^k-cost of a cycle containing w in D_w, and ξ^k be the label returned by* $\textsc{Raise}(\mu(w), i^k, \frac{\pi(w)}{2}, k)$. *Then, $\mu^{\mathcal{G}_\tau^\uparrow}(w) \le \min_k \xi^k \le \widehat{\mu}(w)$.*

4.1.3 The Label-Correcting Algorithm

The overall algorithm for computing $\mu^{\mathcal{G}_\tau^\uparrow}$ is given in Algorithm 3. The main idea is to initialize the labels on base nodes via the recipe given in Theorem 15, before running Algorithm 2. The labels on $V \setminus B(G_\tau)$ are initialized to $\top$. The auxiliary graph D serves as a condensed representation of the "best" paths between base nodes. The arc costs are chosen such that minimum bottleneck cycles in D give a good estimate on the width of base nodes.

Algorithm 3 Label-Correcting: (G_τ, π) 1-player game for Even, $\mathcal{C}$ cover of $\mathcal{T}$ for some universal tree T, $\mu : V \to \bar{L}(T)$ node labeling such that G_τ does not contain loose arcs.

```
procedure LabelCorrecting($(G_\tau, \pi), \mathcal{C}, \mu$)
    $\nu(v) \leftarrow \top$ for all $v \in V$
    Construct auxiliary digraph $D$
    for all components $H$ in $D$ do
        for $k = 0$ to $|\mathcal{C}_{\pi(H)/2}| - 1$ do
            Assign arc costs $c^k$ to $H$ where $\underline{c}^k \le c^k \le \overline{c}^k$
            for all $v \in V(H)$ do
                $i^k \leftarrow$ minimum $c^k$-cost of a cycle containing $v$ in $H$
                $\nu(v) \leftarrow \min(\nu(v), \textsc{Raise}(\mu(v), i^k, \frac{\pi(v)}{2}, k))$
    $\nu \leftarrow \textsc{BellmanFord}((G_\tau, \pi), \nu)$
    return $\nu$
```

In Algorithm 3, the arc costs c^k can be obtained using Algorithm 2. For each base node $w \in B(G_\tau)$, in order to compute $c^k(e)$ for all incoming arcs $e \in \delta_D^-(w)$, we run Algorithm 2 on the subgraph J_w for $|\mathcal{C}^k_{\pi(w)/2}|$ times. If the chain $\mathcal{C}^k_{\pi(w)/2}$ is too long, then this can be sped up using binary search. For specific families of trees such as succinct universal trees, one can compute c^k even faster. More details are given in the full version. Overall, this yields the following generic running time bound.

▶ **Theorem 16.** *In $O(mn^2\gamma(T) \cdot \max_{j,k} |\mathcal{C}_j| \min\{|\mathcal{C}_j^k|, n \log |\mathcal{C}_j^k|\} + n\rho(T, \mathcal{C}) \cdot \max_j |\mathcal{C}_j|)$ time, Algorithm 3 returns $\mu^{\mathcal{G}_\tau^\uparrow}$.*

As mentioned in Section 4.1.1, a succinct $(n, d/2)$-universal tree has a cover with 1 chain per subcover, while a succinct Strahler $(n, d/2)$-universal tree has a cover with at most $\log n$ chains per subcover. So, applying Algorithm 3 to these trees yields the following runtimes.

▶ **Corollary 17.** *For a succinct universal tree, $\mu^{\mathcal{G}_\tau^\uparrow}$ is returned in $O(mn^2 \log n \log d)$.*

▶ **Corollary 18.** *For a succinct Strahler universal tree, $\mu^{\mathcal{G}_\tau^\uparrow}$ is returned in $O(mn^2 \log^3 n \log d)$.*

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows. Theory, algorithms, and applications.* Englewood Cliffs, NJ: Prentice Hall, 1993.

2 Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Solving parity games via priority promotion. *Formal Methods Syst. Des.*, 52(2):193–226, 2018.

3 Massimo Benerecetti, Daniele Dell'Erba, Fabio Mogavero, Sven Schewe, and Dominik Wojtczak. Priority promotion with parysian flair. *arXiv*, abs/2105.01738, 2021. arXiv:2105.01738.

4 Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. A discrete subexponential algorithm for parity games. In *20th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 2607 of *Lecture Notes in Computer Science*, pages 663–674, 2003.

5 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 252–263, 2017.

6 Wojciech Czerwinski, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdzinski, Ranko Lazic, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2333–2349, 2019.

7 Laure Daviaud, Marcin Jurdzinski, and K. S. Thejaswini. The strahler number of a parity game. In *47th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 168 of *LIPIcs*, pages 123:1–123:19, 2020.

8 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 368–377, 1991.

9 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In *5th International Conference on Computer-Aided Verification, CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396, 1993.

10 John Fearnley, Sanjay Jain, Bart de Keijzer, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space. *Int. J. Softw. Tools Technol. Transf.*, 21(3):325–349, 2019.

11 Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 145–156, 2009.

12 Oliver Friedmann. *Exponential Lower Bounds for Solving Infinitary Payoff Games and Linear Programs.* PhD thesis, University of Munich, 2011. URL: http://files.oliverfriedmann.de/theses/phd.pdf.

13 A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Manage. Sci.*, 12(5):359–370, 1966.

14 Marcin Jurdzinski. Deciding the winner in parity games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

15 Marcin Jurdzinski. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, 2000.

16 Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–9, 2017.

17 Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

18 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 224–233, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276748.

19 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 639–648, 2018.

20 Matthias Mnich, Heiko Röglin, and Clemens Rösner. New deterministic algorithms for solving parity games. *Discret. Optim.*, 30:73–95, 2018.

21 Pierre Ohlmann. *Monotonic Graphs for Parity and Mean-Payoff Games*. PhD thesis, University of Paris, 2021. URL: `https://www.irif.fr/~ohlmann/contents/these.pdf`.

22 Pawel Parys. Parity games: Zielonka's algorithm in quasi-polynomial time. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPIcs*, pages 10:1–10:13, 2019.

23 Anuj Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, EECS Department, University of California, Berkeley, 1995.

24 Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *22nd International Workshop on Computer Science Logic, CSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 369–384, 2008.

25 Sven Schewe. Solving parity games in big steps. *J. Comput. Syst. Sci.*, 84:243–262, 2017.

26 Jens Vöge and Marcin Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *12th International Conference on Computer-Aided Verification, CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, 2000.

27 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Countdown μ-Calculus

Jędrzej Kołodziejski
University of Warsaw, Poland

Bartek Klin
University of Oxford, UK

Abstract

We introduce the countdown μ-calculus, an extension of the modal μ-calculus with ordinal approximations of fixpoint operators. In addition to properties definable in the classical calculus, it can express (un)boundedness properties such as the existence of arbitrarily long sequences of specific actions. The standard correspondence with parity games and automata extends to suitably defined countdown games and automata. However, unlike in the classical setting, the scalar fragment is provably weaker than the full vectorial calculus and corresponds to automata satisfying a simple syntactic condition. We establish some facts, in particular decidability of the model checking problem and strictness of the hierarchy induced by the maximal allowed nesting of our new operators.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Automata over infinite objects

Keywords and phrases countdown μ-calculus, games, automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.64

Related Version *Full Version*: https://arxiv.org/abs/2208.00536

Funding *Jędrzej Kołodziejski*: National Science Center (NCN) grant 2021/41/B/ST6/00535

Acknowledgements We are grateful to Mikołaj Bojańczyk for numerous helpful suggestions.

1 Introduction

The modal μ-calculus [14] is a well-known logic for defining and verifying behavioural properties of state-and-transition systems. It extends propositional logic with basic next-step modalities and fixpoint operators to describe long-term behaviour. It is expressive enough to include other temporal logics such as CTL* as fragments, but it has good computational properties, and its simple syntax and semantics makes it a convenient formalism to study.

The μ-calculus has a straightforward inductively-defined semantics, but it is often useful to consider an alternative (but equivalent) semantics based on parity games. A formula φ together with a model $\mathcal{M}$ define a game between two players called ∀dam and ∃ve. Positions in the game are of the form (m, ψ) where m is a point in $\mathcal{M}$ and ψ is a subformula of φ, and moves are defined so that ∃ve has a winning strategy from (m, φ) if and only if φ holds in m. Among other advantages, the game-based semantics provides more efficient algorithms for model checking of μ-calculus formulas than an inductive computation of fixpoints [9].

The model component can be abstracted away from parity games. Indeed, a formula φ itself gives rise to an alternating parity automaton $\mathcal{A}_\varphi$ that recognizes models. The behaviour of an automaton on a model is defined in terms of a parity game, states of $\mathcal{A}_\varphi$ are subformulas of φ, and the transition relation is defined so that it accepts a model $\mathcal{M}$ rooted in a point m if and only if φ holds in m. The advantage of this is that $\mathcal{A}_\varphi$, while conceptually closer to a parity game, is a finite structure even if it is then applied to infinite models.

The modal μ-calculus is a rather expressive formalism: it can define all bisimulation-invariant properties definable in monadic second-order logic (MSO) [13], such as "there is an infinite path of τ-labeled edges". However, there are some properties of interest which are not definable even in MSO. Notable examples include (un)boundedness properties such as "for every number n, there is a path with at least n consecutive τ-labeled edges". An extension of

MSO called MSO+U, aimed at defining such properties, has been considered [6]. However, the satisfiability problem of MSO+U turned out to be undecidable even for word models [4]. Since the modal μ-calculus is a fragment of MSO, it is worthwhile to extend it with a mechanism for defining (un)boundedness properties, in the hope of retaining decidability.

In this paper we propose such an extension: the *countdown μ-calculus* μ^α-ML. In addition to μ-calculus operators, it features countdown operators μ^α and ν^α parametrized by ordinal numbers α. Instead of least and greatest fixpoints, they define ordinal approximations of those fixpoints. Intuitively, while the meaning of classical μ-calculus formulas $\mu x.\varphi(x)$ and $\nu x.\varphi(x)$ is defined by infinite unfolding of the formula φ until a fixpoint is reached, for $\mu^\alpha x.\varphi(x)$ and $\nu^\alpha x.\varphi(x)$ the unfolding stops after α steps (which makes a difference if α is smaller than the *closure ordinal* of φ). The classical fixpoint operators are kept but renamed to μ^∞ and ν^∞, to make clear the lack of any restrictions on the unfolding process.

An inductive definition of the semantics of countdown formulas is just as straightforward as in the classical case. With some more effort, we are able to formulate game-based semantics as well. We introduce *countdown games* and *countdown automata*, which are similar to parity games and alternating automata known from the classical setting, but are additionally equipped with counters that are decremented and reset by the two players according to specific rules. Intuitively, the counters say how many more times various ranks can be visited, in similar manner to the signatures introduced by Walukiewicz [17, Section 3]. A player responsible for decrementing a counter may lose the game if the value of that counter is zero, just as a player responsible for finding the next position in a game may lose if there is no position to go to. The key mechanism of countdown games is implicit in [11], where the authors investigate a nonstandard semantics for the scalar fragment of the μ-calculus equivalent to replacing every μ and ν by our countdown operators μ^α and ν^α, respectively. However, the authors do not abstract from formulas in their definition of games, nor consider the full vectorial calculus that corresponds to automata.

A correspondence between countdown formulas, automata and games is as tight as for the classical μ-calculus. However, complications arise: the distinction between *vectorial* and *scalar* formulas, which in the classical case disappears to a large extent due to the so-called Bekić principle, now becomes pronounced. We prove that vectorial countdown calculus is more expressive than its scalar fragment. We also prove that the countdown operator nesting hierarchy of formulas is proper.

We conjecture that the satisfiability problem is decidable for μ^α-ML. Unfortunately, the lack of positional determinacy in countdown games prevents us from using proof techniques known from parity automata (where one can transform an alternating automaton into a nondeterministic one that guesses the positional strategy). Nevertheless, the existence of an automata model equivalent to logic is encouraging. Apart from allowing us to solve some fragments of the logic, it implies that μ^α-ML does not share some of the troublesome properties of MSO + U that result in undecidability. In particular, it can be used to show that all languages definable in μ-ML have *bounded topological complexity* (i.e. at most Σ^1_2, see [15] for an introduction to topological methods in computer science). Since MSO + U defines a Σ^1_n-complete language for every $n < \omega$ [12, Theorem 2.1], [15, Theorem 7], it follows that some MSO + U-definable languages are not expressible in μ^α-ML (whether μ^α-ML-definability implies MSO + U-definability remains an open question). Since by [8, Theorem 1.3] every logic closed under boolean combinations, projections and defining the language U from Example 4 contains MSO + U, this means that our calculus is *not closed under projections*. This is an arguably good news, as in the light of [3, Theorem 1.4], giving up closure under projections is the only way to go if one wants to design a decidable extension of MSO closed under boolean operations. Decidability of the weak variant $\mathsf{WMSO}+\mathsf{U}$ of $\mathsf{MSO}+\mathsf{U}$ over infinite words [2] and infinite (ranked) trees [5] shows that such extensions are possible. In fact, both results are obtained by establishing a correspondence with equivalent automata models, namely deterministic max-automata [2, Theorem 1] and nested limsup automata [5, Theorem 2]. Since the existence of accepting runs for such automata can be expressed in μ^α-ML, we get that μ^α-ML contains $\mathsf{WMSO}+\mathsf{U}$ on infinite words and trees. The opposite inclusion is false (due to topological reasons), at least for the trees. The relation between μ^α-ML and the ωB-, ωS- and ωBS-automata of [7] remains unclear, as these models do not admit determinization. Also, the relation between our logic and regular cost functions (see e.g. [10]) is less immediate than it could seem at first glance and requires further research.

2 Preliminaries

Fixpoints. Let Ord be the class of all ordinals, and Ord_∞ the class Ord extended with an additional element ∞ greater than all ordinals.

Knaster-Tarski theorem says that every monotonic function $F : A \to A$ on a complete lattice A has the least and the greatest fixpoint, which we denote F^∞_μ and F^∞_ν. Moreover:

- F^∞_μ is the limit of the increasing sequence $F^\alpha_\mu = \bigvee_{\beta<\alpha} F(F^\beta_\mu)$
- F^∞_ν is the limit of the decreasing sequence $F^\alpha_\nu = \bigwedge_{\beta<\alpha} F(F^\beta_\nu)$

where $\alpha \in \mathsf{Ord}$ and $\bigvee, \bigwedge$ are the join and meet operations in A.

Parity games. A *parity game* is played between two players $\exists$ve and $\forall$dam (or simply $\exists$ and $\forall$). It consists of a set of *positions* $V = V_\exists \sqcup V_\forall$ divided between both players, an edge relation $E \subseteq V \times V$, and a labeling $\mathsf{rank} : V \to \mathcal{R}$ for some finite linear order $\mathcal{R} = \mathcal{R}_\exists \sqcup \mathcal{R}_\forall$ divided between the two players.

A *play* is a sequence of positions. After a play $\pi = v_1 \dots v_n \in V^*$, the owner of v_n chooses $(v_n, v_{n+1}) \in E$ and the game moves to v_{n+1}. A player who has no legal moves loses immediately. To determine the winner of an infinite play, we look at the highest $r \in \mathcal{R}$ such that positions with rank r appear infinitely often in the play, and the owner of r loses.

A *strategy* for a player $P \in \{\exists, \forall\}$ is a partial map $\sigma : V^*V_P \to E$ that tells the player how to move. A play $v_1v_2\dots$ is *consistent* with σ if for every n such that $v_n \in V_P$ we have $\sigma(v_1 \dots v_n) = v_{n+1}$. A strategy σ is *winning* from a position v if every play that begins in v and is consistent with σ is a win for P. A strategy is *positional* if $\sigma(\pi)$ depends only on the last position in π. Parity games are *positionally determined*: if a player has a winning strategy from v then (s)he has a winning positional strategy.

Modal μ-calculus. A model $\mathcal{M}$ for a fixed set Act of atomic *actions* consists of a set of *points* $M \ni \mathsf{m}, \mathsf{n}, \cdots$ together with a binary relation $\xrightarrow{\tau} \subseteq M \times M$ for every $\tau \in \mathsf{Act}$.

Formulas of the modal μ-calculus μ-ML are given by the grammar:

$$\varphi ::= x \mid \top \mid \bot \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mu x.\varphi \mid \nu x.\varphi \mid \langle\tau\rangle\varphi \mid [\tau]\varphi \tag{1}$$

where x ranges over a fixed infinite set Var of variables and $\tau \in \mathsf{Act}$. Given a valuation $\mathsf{val} : \mathsf{Var} \to \mathcal{P}(M)$, the semantics $[\![\varphi]\!]^{\mathsf{val}} \subseteq M$ for all formulas φ is defined inductively, with $\mu x.\varphi$ and $\nu x.\varphi$ denoting the least and greatest fixpoints, respectively, of the monotonic function $H \mapsto [\![\varphi]\!]^{\mathsf{val}[x \mapsto H]}$ on the complete lattice $\mathcal{P}(M)$. More details can be found e.g. in [1, 16], but they can also be discerned from Section 3 below, where the semantics of countdown μ-calculus is presented in detail.

The above syntax does not include negation, but μ-calculus formulas are semantically closed under negation. For every formula $\widetilde{\varphi}$ there is a formula $\widetilde{\varphi}$ that acts as the negation of φ on every model, defined by induction in a straightforward way:

$$\widetilde{\varphi_1 \vee \varphi_2} = \widetilde{\varphi_1} \wedge \widetilde{\varphi_2}, \qquad \widetilde{\langle\tau\rangle\varphi} = [\tau]\widetilde{\varphi}, \qquad \widetilde{\mu x.\varphi} = \nu x.\widetilde{\varphi}, \qquad \text{etc.} \tag{2}$$

Vectorial μ-calculus. A syntactically richer version of the modal μ-calculus admits mutual fixpoint definitions of multiple properties, in formulas such as $\mu_1(x_1, x_2).(\varphi_1, \varphi_2)$, where variables x_1 and x_2 may occur both in φ_1 and φ_2. Given a valuation val as before, this formula is interpreted as the least fixpoint of the monotonic function $(H_1, H_2) \mapsto ([\![\varphi_1]\!]^{\mathsf{val}[x_i \mapsto H_i]}, [\![\varphi_1]\!]^{\mathsf{val}[x_i \mapsto H_i]})$ on the complete lattice $\mathcal{P}(M)^2$; the resulting pair of sets is then projected to the first component as dictated by the subscript in μ_1. Tuples of any size are allowed. This *vectorial* calculus is expressively equivalent to the scalar version described before, thanks to the so-called *Bekić principle* which says that the equality:

$$\mu \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} . \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} \mu x_1.f_1(x_1,\ \mu x_2.f_2(x_1, x_2)) \\ \mu x_2.f_2(\mu x_1.f_1(x_1, x_2),\ x_2) \end{pmatrix} \tag{3}$$

holds for every pair of monotone operations $f_i : A_1 \times A_2 \to A_i$ on complete lattices A_1, A_2, and similarly for the greatest fixpoint operator ν in place of μ.

3 Countdown μ-calculus

We now introduce the *countdown μ-calculus* μ^α-ML. We begin with the scalar version.

3.1 The scalar fragment

As before, fix an infinite set Var of variables and a set Act of actions. The syntax of *(scalar) countdown μ-calculus* is defined as follows:

$$\varphi ::= x \mid \top \mid \bot \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mu^\alpha x.\varphi \mid \nu^\alpha x.\varphi \mid \langle\tau\rangle\varphi \mid [\tau]\varphi \tag{4}$$

for $x \in \mathsf{Var}$, $\tau \in \mathsf{Act}$ and $\alpha \in \mathsf{Ord}_\infty$; the presence of ordinal numbers α is the only syntactic difference with (1). A formula with no free variables is called a *sentence*. In case $|\mathsf{Act}| = 1$, we may skip the labels and write $\Diamond$ and $\Box$ instead of $\langle\tau\rangle$ and $[\tau]$. In statements that apply both to least and greatest fixpoints, we will sometimes use η^α to denote either μ^α or ν^α.

Given a model $\mathcal{M}$, for every valuation $\mathsf{val} : \mathsf{Var} \to \mathcal{P}(M)$, the *semantics* $[\![\varphi]\!]^{\mathsf{val}} \subseteq M$ is defined inductively as follows:

$$\begin{aligned}
[\![x]\!]^{\mathsf{val}} &= \mathsf{val}(x); \\
[\![\top]\!]^{\mathsf{val}} = M \quad &\text{and} \quad [\![\bot]\!]^{\mathsf{val}} = \emptyset \\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathsf{val}} = [\![\varphi_1]\!]^{\mathsf{val}} \cup [\![\varphi_2]\!]^{\mathsf{val}} \quad &\text{and} \quad [\![\varphi_1 \wedge \varphi_2]\!]^{\mathsf{val}} = [\![\varphi_1]\!]^{\mathsf{val}} \cap [\![\varphi_2]\!]^{\mathsf{val}}; \\
[\![\langle\tau\rangle\varphi]\!]^{\mathsf{val}} = \{\mathsf{m} \in M \mid \exists_{\mathsf{n} \in [\![\varphi]\!]^{\mathsf{val}}}\ \mathsf{m} \xrightarrow{\tau} \mathsf{n}\} \quad &\text{and} \quad [\![[\tau]\varphi]\!]^{\mathsf{val}} = \{\mathsf{m} \in M \mid \forall_{\mathsf{n} \in [\![\varphi]\!]^{\mathsf{val}}}\ \mathsf{m} \xrightarrow{\tau} \mathsf{n}\}; \\
[\![\mu^\alpha x.\varphi]\!]^{\mathsf{val}} = F^\alpha_\mu \quad &\text{and} \quad [\![\nu^\alpha x.\varphi]\!]^{\mathsf{val}} = F^\alpha_\nu
\end{aligned}$$

where in the last clause $F(H) = [\![\varphi]\!]^{\mathsf{val}[x \mapsto H]}$. We will skip the index val if it is immaterial or clear from the context.

This obviously contains the classical μ-calculus, but is capable of capturing *boundedness* and *unboundedness* properties which are not expressible in the classical setting:

▶ **Example 1.** For $|\mathsf{Act}| = 1$, consider the formula $\nu^\alpha x.\Diamond x$. In a model $\mathcal{M}$, for $\alpha < \omega$ the set $[\![\nu^\alpha x.\Diamond x]\!]$ consists of the points from which there is a path of length at least α. Hence, $\nu^\omega x.\Diamond x$ holds in a point if there are arbitrarily long finite paths starting from there.

3.2 The vectorial calculus

The (full) *countdown μ-calculus* is defined as for its scalar fragment, except that fixpoint operators act on tuples (vectors) of formulas rather than on single formulas.

▶ **Definition 2.** *The syntax of* countdown μ-calculus *is given as follows:*

$$\varphi ::= x \mid \top \mid \bot \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mu_i^\alpha \overline{x}.\overline{\varphi} \mid \nu_i^\alpha \overline{x}.\overline{\varphi} \mid \langle\tau\rangle\varphi \mid [\tau]\varphi$$

for $1 \leq i \leq n < \omega$, $\overline{x} = \langle x_1, ..., x_n\rangle \in \mathsf{Var}^n$, $\overline{\varphi} = \langle\varphi_1, ..., \varphi_n\rangle$ a tuple of formulas, $\tau \in \mathsf{Act}$ and $\alpha \in \mathsf{Ord}_\infty$.

▶ **Definition 3.** *The meaning $[\![\varphi]\!]^{\mathsf{val}} \subseteq M$ of a formula φ in a model $\mathcal{M}$ under valuation* val *is defined by induction the same way as for the scalar formulas except for the operators μ_i^α and ν_i^α, in which case:*

$$[\![\mu_i^\alpha \overline{x}.\overline{\varphi}]\!]^{\mathsf{val}} = \pi_i(F_\mu^\alpha) \quad \text{and} \quad [\![\nu_i^\alpha \overline{x}.\overline{\varphi}]\!]^{\mathsf{val}} = \pi_i(F_\nu^\alpha)$$

where the monotone map $F : (\mathcal{P}(M))^n \to (\mathcal{P}(M))^n$ is given as:

$$F(H_1, ..., H_n) = ([\![\varphi_1]\!]^{\mathsf{val}'}, ..., [\![\varphi_n]\!]^{\mathsf{val}'})$$

for $\mathsf{val}' = \mathsf{val}[x_1 \mapsto H_1, ..., x_n \mapsto H_n]$ and $\pi_i : (\mathcal{P}(M))^n \to \mathcal{P}(M)$ is the i-th projection.

Note that operators μ^∞ and ν^∞ are equivalent to μ and ν from the classical μ-calculus. Furthermore, for every ordinal α, the formula $\mu_i^{\alpha+1}\overline{x}.\overline{\psi}$ is equivalent to

$$\psi_i[x_1 \mapsto \mu_1^\alpha \overline{x}.\overline{\psi}, \dots, x_n \mapsto \mu_n^\alpha \overline{x}.\overline{\psi}]$$

and similarly for $\nu^{\alpha+1}$. As a result, without loss of generality we may assume that in countdown operators μ^α and ν^α only limit ordinals α are used.

The countdown μ-calculus is semantically closed under negation in the same way as the classical calculus, extending (2) with the straightforward $\widetilde{\mu_i^\alpha \overline{x}.\overline{\varphi}} = \nu_i^\alpha \overline{x}.\overline{\widetilde{\varphi}}$ and $\widetilde{\nu_i^\alpha \overline{x}.\overline{\varphi}} = \mu_i^\alpha \overline{x}.\overline{\widetilde{\varphi}}$.

In Section 6 we will compare the expressive power of the vectorial and scalar countdown μ-calculus in detail. For now, let us show that Bekić principle (3) fails for countdown operators:

▶ **Example 4.** An infinite word $W \in \Gamma^\omega$ over the alphabet $\Gamma = \{\mathsf{a}, \mathsf{b}\}$ can be seen as a model for $\mathsf{Act} = \Gamma$ with ω as the set of points and with transition relations defined by:

$$n \xrightarrow{\tau} m \iff m = n + 1 \text{ and } W_n = \tau.$$

For every regular language $K \subseteq \Gamma^*$ and $x \in \mathsf{Var}$, it is straightforward to define a fixpoint formula (in the classical μ-calculus, so without countdown operators) $\langle K\rangle x$ that holds in a point n, for a valuation val, if and only if there exists a word $w \in K$ and a path in W labelled with w that starts in n and ends in a point that belongs to $\mathsf{val}(x)$. Then, the formula:

$$\varphi = \nu_1^\omega(x_1, x_2).(\langle\Gamma^*\rangle x_2, \langle\mathsf{a}\rangle x_2)$$

is true in a word W iff it contains arbitrarily long blocks of consecutive a's. To see this, observe that at the i-th step of approximation: (i) the second component (x_2) contains a point n iff the next i transitions are all labelled with a, and (ii) the first component (x_1) contains a point n iff the second component contains at least one point after n.

However, the following scalar formula constructed by analogy to the Bekić principle:

$$\psi = \nu^\omega x_1.\langle\Gamma^*\rangle(\nu^\omega x_2.\langle\mathsf{a}\rangle x_2)$$

is equivalent to $\langle\Gamma^*\rangle(\nu^\omega x_2.\langle\mathsf{a}\rangle x_2)$, and the formula under $\langle\Gamma^*\rangle$ holds in a point iff all the future transitions from that point are labelled with a. Thus, ψ holds (in any point) iff the word W is of the form $\Gamma^*\mathsf{a}^\omega$, and so ψ is not equivalent to φ.

4 Countdown Games

The notion of a countdown game extends that of a parity game. As for parity games, it assumes a fixed finite linear order of ranks $\mathcal{R} = \mathcal{R}_\exists \sqcup \mathcal{R}_\forall$. In addition, we fix a subset $\mathcal{D} \subseteq \mathcal{R}$ of *nonstandard* ranks; at positions with these ranks countdowns will occur. Denote $\mathcal{D}_\exists = \mathcal{D} \cap \mathcal{R}_\exists$ and $\mathcal{D}_\forall = \mathcal{D} \cap \mathcal{R}_\forall$.

A *countdown game* consists of a set of *positions* $V = V_\exists \sqcup V_\forall$ divided between players ∃ve and ∀dam, an edge relation $E \subseteq V \times V$, a labelling $\mathsf{rank} : V \to \mathcal{R}$, and an initial counter valuation $\mathsf{ctr}_I : \mathcal{D} \to \mathsf{Ord}$. Each nonstandard rank has an associated counter.

Each game *configuration* consists of a position $v \in V$ together with a counter valuation $\mathsf{ctr} : \mathcal{D} \to \mathsf{Ord}$. We consider *positional* and *countdown* configurations, denoted respectively $\langle v, \mathsf{ctr}\rangle$ and $[v, \mathsf{ctr}]$, with the following moves allowed:

- From a positional configuration $\langle v, \mathsf{ctr}\rangle$, the owner of v chooses an edge $(v, w) \in E$ and the game proceeds from the countdown configuration $[w, \mathsf{ctr}]$;
- From a countdown configuration $[v, \mathsf{ctr}]$, the owner of $r = \mathsf{rank}(v)$ chooses a counter valuation ctr' such that:
  - $\mathsf{ctr}'(r') = \mathsf{ctr}_I(r')$ for $r' < r$,
  - $\mathsf{ctr}'(r) < \mathsf{ctr}(r)$ (if r is nonstandard),
  - $\mathsf{ctr}'(r') = \mathsf{ctr}(r')$ for $r' > r$,

  and the game proceeds from the positional configuration $\langle v, \mathsf{ctr}'\rangle$. In words: counters for ranks lower than r are reset, the counter for r (if any) is decremented, and counters for higher ranks are left unchanged. Note that if r is standard then there is no real choice here: ctr' is determined by ctr. And if r is nonstandard then the move amounts to choosing an ordinal $\alpha < \mathsf{ctr}(r)$.

Every play of the game alternates between positional and countdown configurations, and in each move only one component of the configuration is modified. Therefore, although a play is formally a sequence of configurations, it can be more succinctly represented as an alternating sequence of positions and counter valuations:

$$\pi = v_1 \mathsf{ctr}_2 v_2 \mathsf{ctr}_2 v_3 \mathsf{ctr}_3 \cdots \tag{5}$$

This has the same length as the sequence of configurations, and we will call it the length of the play. A *phase* of a game is a set of its finite plays that is convex with respect to the prefix ordering.

In any configuration, if the player responsible for making the next move is stuck, (s)he looses immediately. Otherwise, in an infinite play, the owner of the greatest rank appearing infinitely often looses, as in parity games. Strategies and winning strategies are defined as for classical parity games, as partial functions from finite plays to moves.

Given configuration γ, we denote the game *initialized in the configuration* γ by $\mathcal{G}, \gamma$. The default initial counter assignment is ctr_I and the default initial mode is the positional one, meaning that $\mathcal{G}, v$ stands for $\mathcal{G}, \langle v, \mathsf{ctr}_I\rangle$.

Note that the only way the counters may interfere with a play is when a counter has value 0 and so its owner cannot decrement it. It is therefore beneficial for a player to have greater ordinals at his/her counters.

Countdown games are not positionally determined, in the sense that the players may need to look at the counter values in order to choose a winning move (although they are *configurationally determined*, since a countdown game $\mathcal{G}$ can be seen as a parity game with configurations of $\mathcal{G}$ as its positions).

5 Countdown Automata

Countdown automata are a stepping stone between formulas and games. A countdown formula will define an automaton, which will then recognize a model in terms of a countdown game. Since formulas can have free variables, for technical reasons we will also consider automata with free variables. These variables resemble terminal states in that they can be targets of transitions, but no transitions originate in them, and whether they accept or not depends on an external valuation.

▶ **Definition 5.** *A* countdown automaton *consists of:*

- *a finite set of states $Q = Q_\exists \sqcup Q_\forall$ divided between two players;*
- *an initial state $q_I \in Q$;*
- *a transition function $\delta : Q \to \mathcal{P}(Q \sqcup \mathsf{Var}) \sqcup (\mathsf{Act} \times (Q \sqcup \mathsf{Var}))$ (we call the left part* ϵ-transitions *and the right one* modal transitions*);*
- *an assignment of ranks $\mathsf{rank} : Q \to \mathcal{R}$ and an assignment of initial counter values $\mathsf{ctr}_I : \mathcal{D} \to \mathsf{Ord}$, as in a countdown game.*

The language of an automaton is defined in terms of a countdown game, analogously to parity games and parity automata.

▶ **Definition 6.** *Fix an automaton $\mathcal{A} = (Q, q_I, \delta, \mathsf{rank}, \mathsf{ctr}_I)$. Given a model $\mathcal{M}$, a valuation $\mathsf{val} : \mathsf{Var} \to \mathcal{P}(M)$ and a point $\mathsf{m}_I \in M$, we define the semantic game $\mathcal{G}^{\mathsf{val}}(\mathcal{A})$ to be the countdown game $(V, E, \mathsf{rank}', \mathsf{ctr}_I)$ where positions are of the form $V = M \times (Q \sqcup \mathsf{Var})$ and the edge relation E is defined as follows. In a position (m, q) for $q \in Q$:*

- *if $\delta(q) \subseteq Q \sqcup \mathsf{Var}$, outgoing edges (called ϵ-edges, or ϵ-moves) are $\{((\mathsf{m}, q), (\mathsf{m}, z)) \mid z \in \delta(q)\}$,*
- *if $\delta(q) = (\tau, p)$, outgoing edges (modal edges, modal moves) are $\{((\mathsf{m}, q), (\mathsf{n}, p)) \mid \mathsf{m} \xrightarrow{\tau} \mathsf{n}\}$.*

There are no outgoing edges from positions (m, x) for $x \in \mathsf{Var}$.

For $q \in Q$, the owner of the position (m, q) is the owner of the state q, and $\mathsf{rank}'(\mathsf{m}, q) = \mathsf{rank}(q)$. For $x \in \mathsf{Var}$, the position (m, x) belongs to $\forall$dam if $\mathsf{m} \in \mathsf{val}(x)$ and to $\exists$ve otherwise. The rank $\mathsf{rank}'(\mathsf{m}, x)$ can be set arbitrarily, as it does not affect the outcome of the game. The initial counter assignment ctr_I is kept the same.

The language $[\![\mathcal{A}]\!]^{\mathsf{val}} \subseteq M$ of an automaton $\mathcal{A}$ is the set of all points $\mathsf{m} \in M$ for which the configuration $\langle(\mathsf{m}, q_I), \mathsf{ctr}_I\rangle$ in the game $\mathcal{G}^{\mathsf{val}}(\mathcal{A})$ is winning for $\exists$ve.

It is worth to mention that although in general countdown games are not positional, one can show a much weaker but still useful fact: in the particular case of semantic games, the winning player always has a strategy that does not look at the counters in the initial *pre-modal* phase of the game (that is, *before the first modal move*).

The countdown calculus and countdown automata have the same expressive power, i.e. there are language-preserving translations $\varphi \mapsto \mathcal{A}_\varphi$ and $\mathcal{A} \mapsto \varphi_\mathcal{A}$ between formulas and automata. As in the classical setting, the link between formulas and automata is very useful in establishing facts about the logic. For example, one can use game semantics to show that every formula of the standard μ-ML can be transformed into an equivalent guarded one. Thanks to the equivalence between *countdown* formulas and *countdown* automata, the same is true for μ^α-ML.

We will now explain the translations between logic and automata in turn.

5.1 From formulas to automata – Game Semantics

Every countdown formula $\varphi \in \mu^{\alpha}$-ML gives rise to a countdown automaton $\mathcal{A}_\varphi$ such that $[\![\varphi]\!]^{\mathsf{val}} = [\![\mathcal{A}_\varphi]\!]^{\mathsf{val}}$ for every model $\mathcal{M}$ and valuation val. Specifically, given a formula φ (with some free variables), we define an automaton $\mathcal{A}_\varphi = (Q, q_I, \delta, \mathsf{rank}, \mathsf{ctr}_I)$ (over the same free variables) as follows:

- $Q = \mathsf{SubFor}(\varphi) - \mathsf{FreeVar}(\varphi)$ is the set of all subformulas other than the free variables of φ (*without* identifying different occurrences of identical subformulas, i.e., here a subformula means a path in the syntactic tree of φ from the root of φ to the root node of the subformula). Ownership of a state in Q depends on the topmost connective, with $\exists$ve owning $\vee$ and $\langle\tau\rangle$ and $\forall$dam owning $\wedge$ and $[\tau]$; ownership of fixpoint subformulas, countdown subformulas and variables can be set arbitrarily as it will not matter;
- $q_I = \varphi$;
- the transition function is defined by cases:
  - $\delta(\theta_1 \vee \theta_2) = \delta(\theta_1 \wedge \theta_2) = \{\theta_1, \theta_2\}$,
  - $\delta(\langle\tau\rangle\theta) = \delta([\tau]\theta) = (\tau, \theta)$,
  - $\delta(\eta_i^{\alpha}\overline{x}.\overline{\theta}) = \{\theta_i\}$ (for $\eta = \mu$ or $\eta = \nu$),
  - $\delta(x) = \{\theta_i\}$, where $\eta_j^{\alpha}(x_1, ..., x_n).(\theta_1, ..., \theta_n)$ is the (unique) subformula of φ binding x with $x = x_i$.
- For the ranking function, assume that the lowest rank in $\mathcal{R}$ is standard and call it 0 (ownership of this rank does not matter). Then let rank assign 0 to all subformulas of φ except for immediate subformulas of fixpoint operators. To those, assign ranks in such a way that subformulas have strictly smaller ranks than their superformulas, and for every subformula $\eta_i^{\alpha}\overline{x}.\overline{\varphi}$:
  - all formulas in the tuple $\overline{\varphi}$ have the same rank r,
  - r belongs to $\exists$ve if $\eta = \mu$ and to $\forall$dam if $\eta = \nu$, and
  - if $\alpha = \infty$ then r is standard, otherwise it is nonstandard and $\mathsf{ctr}_I(r) = \alpha$.

We denote $\mathcal{G}^{\mathsf{val}}(\varphi) = \mathcal{G}^{\mathsf{val}}(\mathcal{A}_\varphi)$.

▶ **Theorem 7** (Adequacy). *For every model $\mathcal{M}$ and valuation* val, $[\![\varphi]\!]^{\mathsf{val}} = [\![\mathcal{A}_\varphi]\!]^{\mathsf{val}}$.

Proof (sketch). As with the classical *mu*-calculus, the proof proceeds by induction on the complexity of the formula. The only new cases of $\mu^{\alpha}\overline{x}.\overline{\varphi}$ and $\nu^{\alpha}\overline{x}.\overline{\varphi}$ are proven by transfinite induction on α. ◀

▶ **Example 8.** For $\mathsf{Act} = \{\tau\}$, consider the formula $\varphi = \nu^{\omega}x.\Diamond x$ from Example 1. The automaton $\mathcal{A}_\varphi$ has three states: $Q = \{\varphi, \Diamond x, x\}$, with φ the initial state, and the transition function comprises two deterministic ϵ-transitions and one modal transition:

$$\delta(\varphi) = \{\Diamond x\}, \qquad \delta(\Diamond x) = (\tau, x), \qquad \delta(x) = \{\Diamond x\}.$$

The state $\Diamond x$ is owned by $\exists$ve; ownership of the other two states does not matter. The automaton uses two ranks, $0 < 1$, where 0 is standard and 1 is nonstandard, assigned to states by: $\mathsf{rank}(\varphi) = \mathsf{rank}(x) = 0$ and $\mathsf{rank}(\Diamond x) = 1$. Rank 1 is owned by $\forall$dam; ownership of rank 0 does not matter. (Note how the state $\Diamond x$ is owned by $\exists$ve, but its rank is owned by $\forall$dam). The initial counter value is $\mathsf{ctr}_I(1) = \omega$.

Now consider any model $\mathcal{M}$. Since Act has only one element, $\mathcal{M}$ is simply a directed graph. The semantic game $\mathcal{G}(\varphi)$ on $\mathcal{M}$ (φ has no free variables, so neither has $\mathcal{A}_\varphi$ and we need not consider valuations val) has positions of the form (m, q) where $\mathsf{m} \in M$ and $q \in Q$, with ownership and rank inherited from q. Edges are of the form:

- $((\mathsf{m}, \varphi), (\mathsf{m}, \Diamond x))$ and $((\mathsf{m}, x), (\mathsf{m}, \varphi))$ – the ϵ-edges,
- $((\mathsf{m}, \Diamond x), (\mathsf{n}, x))$ such that $\mathsf{m} \to \mathsf{n}$ is an edge in $\mathcal{M}$ – the modal edges.

Configurations of the game arise from positions together with counter valuations; there is only one nonstandard rank, so a counter valuation is simply an ordinal.

For a point $\mathsf{m} \in \mathcal{M}$, the default initial configuration of the game is the positional configuration $\langle(\mathsf{m}, \varphi), \omega\rangle$. A play that begins in this configuration proceeds as follows:

1. The first move is deterministic, to the countdown configuration $[(\mathsf{m}, \diamond x), \omega]$.
2. $\forall$dam, as the owner of the rank of $\diamond x$, makes the next move: he chooses a number $k < \omega$, and the games moves to the positional configuration $\langle(\mathsf{m}, \diamond x), k\rangle$.
3. $\exists$ve owns the position, so she makes the next move: she chooses a point $\mathsf{n} \in M$ such that $\mathsf{m} \xrightarrow{\tau} \mathsf{n}$, and the game moves to the countdown configuration $[(\mathsf{n}, x), k]$.
4. The rank of x is standard, so in the next move the counter does not change and the game moves to $\langle(\mathsf{n}, x), k\rangle$. The next move is also deterministic, to the countdown configuration $[(\mathsf{n}, \diamond x), k]$. The game then goes back to step 2. above, with k in place of ω.

From this it is clear that $\exists$ve wins from $\langle(\mathsf{m}, \varphi), \omega\rangle$ if and only if $\mathcal{M}$ has arbitrarily long paths that begin in m, as stated in Example 1.

5.2 From automata to formulas

▶ **Theorem 9.** *For every countdown automaton $\mathcal{A}$ there exists a countdown formula $\varphi_{\mathcal{A}}$ s.t. $[\![\mathcal{A}]\!]^{\mathsf{val}} = [\![\varphi_{\mathcal{A}}]\!]^{\mathsf{val}}$ for every model $\mathcal{M}$ and valuation* val.

Proof (sketch). We sketch the construction of $\varphi_{\mathcal{A}}$. For an automaton $\mathcal{A} = (Q, q_I, \delta, \mathsf{rank}, \mathsf{ctr}_I)$, by induction on $r \in \mathcal{R}$ we build a formula $\psi_{r,q}$ for each $q \in Q$. Then we put $\varphi_{\mathcal{A}} = \psi_{r_{\mathsf{max}}, q_I}$. Thus for the base case of the lowest rank $r = 0$:

- if $\delta(s) = (\tau, p)$ then for $\psi_{0,s}$ we put $\langle\tau\rangle x_p$ if q belongs to $\exists$ve and $[\tau]x_p$ if q belongs to $\forall$dam,
- if $\delta(s) \subseteq Q$ then for $\psi_{0,s}$ we put $\bigvee_{p \in \delta(s)} x_p$ if q belongs to $\exists$ve and $\bigwedge_{p \in \delta(s)} x_p$ if q belongs to $\forall$dam.

For the inductive step, let $q_1, \ldots, q_d$ be all states in Q with rank r. For every q_i define the vectorial formula:

$$\theta_i = \eta^{\alpha}_{q_i}(x_{q_1}, \ldots, x_{q_d}).(\psi_{r,q_1}, \ldots, \psi_{r,q_d})$$

with $\alpha = \mathsf{ctr}_I(r)$ and $\eta = \mu$ if r belongs to $\exists$ve and $\eta = \nu$ if r belongs to $\forall$dam. Then put $\psi_{r+1,q} = \psi_{r,q}[x_{q_1} \mapsto \theta_1, \ldots, x_{q_d} \mapsto \theta_d]$. ◀

6 Vectorial vs. scalar calculus

In this section we investigate the relation between scalar and vectorial formulas. We have already seen with Example 4 that unlike with standard fixpoints, the Bekić principle is not valid in the countdown setting. Interestingly, scalar formulas correspond to automata with a simple syntactic restriction.

▶ **Proposition 10.** *Scalar countdown formulas and automata where every two states have different ranks have equal expressive power.*

Proof (sketch). Inspecting the translations between formulas and automata from Sections 5.1 and 5.2, it is evident that injectively ranked automata are translated to scalar formulas, and that, although in our translation the choice of the assignment of ranks is not deterministic, every scalar formula can be translated to an injectively ranked automaton. ◀

Since the Bekić principle fails, a natural question is whether there is another way of transforming vectorial formulas to scalar form (or, equivalently, arbitrary countdown automata to injectively ranked ones). We shall give a negative answer in Theorem 11. However, before we proceed, let us analyse the following example, which shows that scalar formulas are more expressive than they may seem, covering in particular the property from Example 4.

6.1 Languages of unbounded infixes

Fix a regular language of finite words $L \subseteq \Gamma^*$. Let $\mathcal{U}(L) \subseteq \Gamma^\omega$ be the language of all infinite words that contain arbitrarily long infixes from L. For instance, the language from Example 4 is $\mathcal{U}(\mathsf{a}^*)$. We shall now show that $\mathcal{U}(L)$ can be defined in the countdown μ-calculus, first by a vectorial formula, then by a scalar one.

Consider a finite deterministic automaton $\mathcal{A} = (Q, \delta, q_I, F)$ that recognizes L. Let $\delta^+ : \Gamma^+ \times Q \to Q$ be the unique inductive extension of the transition function $\delta : \Gamma \times Q \to Q$ to nonempty words. Define $K_{p,q} = \{w \in \Gamma^+ \mid \delta^+(w, p) = q\}$ the (regular) language of nonempty words leading from p to q in $\mathcal{A}$, and let $K_{p,F}$ denote the union $\bigcup_{q \in F} K_{p,q}$. By the pigeonhole principle we have $\mathcal{U}(L) = \bigcup_{q \in Q} \mathcal{U}_q(L)$, where $\mathcal{U}_q(L) \subseteq \Gamma^\omega$ consists of words such that for every $n < \omega$, w has an infix $w_n = v_I u_1 ... u_n v_F \in L$ s.t. (i) $v_I \in K_{q_I,q}$, (ii) $u_1, ..., u_n \in K_{q,q}$, and (iii) $v_F \in K_{q,F}$. Then $\mathcal{U}_q(L)$ can be defined by a vectorial formula:

$$\mathcal{U}_q(L) = [\![\nu_1^\omega(x_1, x_2).(\langle \Gamma^* K_{q_I,q} \rangle x_2, \langle K_{q,q} \rangle x_2 \wedge \langle K_{q,F} \rangle \top)]\!]$$

where $\langle K \rangle \psi$ is the formula as explained in Example 4. Indeed, the corresponding semantic game on a word w proceeds as follows:

1. $\forall$dam chooses a number $n < \omega$ as the value of his only counter,
2. $\exists$ve skips a prefix $v_0 v_I \in \Gamma^* K_{q_I,q}$ of w,
3. $\forall$dam decrements his counter;
4. $\exists$ve keeps moving through $u_1, u_2, ... \in K_{q,q}$ so that after each step, some state in F is reachable from q by some prefix of the remaining word. After each such choice of u_i $\forall$dam has to decrement his counter, and so $\exists$ve wins iff she can make at least $n - 1$ such steps.

The two different stages in which $\forall$dam's counter is decremented reflect the two-phase dynamics of the game: first $\forall$dam challenges $\exists$ve with a number, and then $\exists$ve shows that she can provide an infix long enough.

It is more tricky to define the language $\mathcal{U}_q(L)$ with a scalar formula, but it turns out to be possible. To this end, observe that without loss of generality we may restrict attention to words w such that:

1. the infixes $w_n \in L$ start arbitrarily far in w;
2. each w_n can be decomposed as $v_I u_1 ... u_n v_F \in L$ s.t. (i) $v_I \in K_{q_I,q}$, (ii) $u_1, ..., u_n \in K_{q,q}$, (iii) $v_F \in K_{q,F}$, and additionally *(iv) all u_i begin with the same letter* $\mathsf{a} \in \Gamma$;
3. there are at least two distinct letters $\mathsf{a}, \mathsf{b} \in \Gamma$ that appear infinitely often in w;
4. the first letter of w is b.

Indeed, for (1) note that otherwise w_n start in the same position k for all n large enough. But then even the stronger property "There exists a position k such that the run of $\mathcal{A}$ from k visits q and F infinitely often" holds, and this is easily definable by a fixpoint formula.

Item (2) follows from the pigeonhole principle and the observation that in $w_{n\times|\Gamma|} = v_I u_1 ... u_{n\times|\Gamma|} v_F$ at least n u_i's begin with the same letter.

For (3) observe that otherwise w has a suffix a^ω for some $\mathsf{a} \in \Gamma$, in which case membership in $\mathcal{U}_q(L)$ is definable by a fixpoint formula. This is because an ultimately periodic word is bisimilar to a finite model, and so every monotone map reaches its fixpoints in finitely many steps, meaning that the countdown operator ν^ω is equivalent to ν^∞.

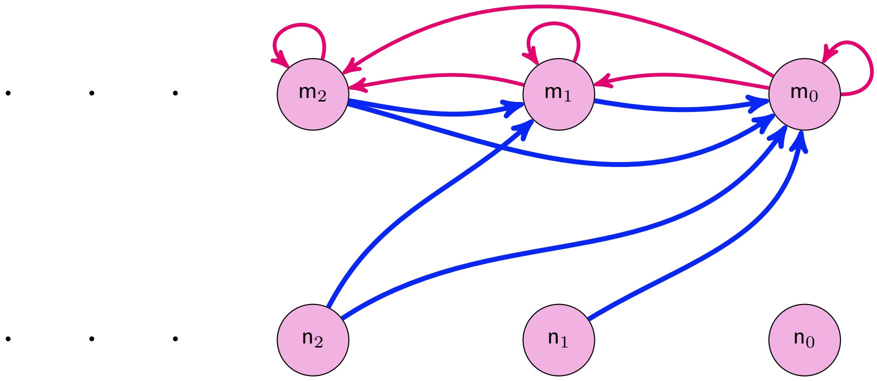


Figure 1 The model $\mathcal{M}$. Blue arrows represent edges labeled both with a and b, and pink arrows are edges labeled only with b.

Finally, for (4) note that the language $\mathcal{U}_q(L)$ is closed under adding and removing finite prefixes, and so if a formula φ defines $\mathcal{U}_q(L) \cap \mathsf{b}\Gamma^\omega$, then the formula $\langle\Gamma^*\rangle(\langle\mathsf{b}\rangle\top \wedge \varphi)$ defines $\mathcal{U}_q(L)$.

With this in mind, define:

$$\varphi = \nu^\omega x.(\langle\mathsf{b}\rangle\top \wedge \langle\Gamma^* K_{q_I,q}\rangle(\langle\mathsf{a}\rangle\top \wedge x)) \vee (\langle K_{q,q}\rangle(\langle\mathsf{a}\rangle\top \wedge x) \wedge \langle\mathsf{a}\rangle\top \wedge \langle K_{q,F}\rangle\top).$$

Note how $\langle\mathsf{b}\rangle\top \wedge x$ and $\langle\mathsf{a}\rangle\top \wedge x$ replace x_1 and x_2 from the vectorial formula. Consider the corresponding semantic game on a word w. Consider configurations of the game with the main disjunction as the formula component. Every infinite play of the game must visit such configurations infinitely often. In such a configuration, if the next letter in the model is either a or b then $\exists$ve must choose the right or left disjunct respectively. In particular, once the game reaches a configuration where $\langle\mathsf{a}\rangle\top$ holds, it must also hold every time the variable x in unraveled in the future. As a result, $\exists$ve wins from a configuration where $\langle\mathsf{a}\rangle\top$ holds against $\forall$dam's counter $n < \omega$ iff there is $u_1 ... u_{n+1} v_F$ starting in the current position such that $u_1, ..., u_{n+1} \in K_{q,q}$, each u_i starts with a, and $v_F \in K_{q,F}$. Moreover, $\exists$ve wins from a position where $\langle\mathsf{b}\rangle\top$ holds, against $\forall$dam's $n+1 < \omega$, iff there is $v_I \in \Gamma^* K_{q_I,q}$ starting in the current position such that the next position after v_I satisfies $\langle\mathsf{a}\rangle\top$ and $\exists$ve wins from there against n. Putting this together, we get that $\exists$ve wins from a position satisfying $\langle\mathsf{b}\rangle\top$ against n iff there is $v_I u_1 ... u_n v_F = w_n$ as in condition (2) above. Since the game starts with $\forall$dam choosing an arbitrary $n < \omega$, it follows that indeed φ defines $\mathcal{U}_q(L)$.

6.2 Greater expressive power of the vectorial calculus

We now show an example of a property that is definable in the vectorial countdown calculus but not in the scalar one.

Fixing $\mathsf{Act} = \{\mathsf{a}, \mathsf{b}\}$, consider a model $\mathcal{M} = (M, \xrightarrow{\mathsf{a}}, \xrightarrow{\mathsf{b}})$ with points $M = \{\mathsf{m}_i, \mathsf{n}_i \mid i < \omega\}$, and with exactly the edges: $\mathsf{m}_i \xrightarrow{\mathsf{a}} \mathsf{m}_j$, $\mathsf{n}_i \xrightarrow{\mathsf{a}} \mathsf{m}_j$ and $\mathsf{n}_i \xrightarrow{\mathsf{b}} \mathsf{m}_j$ for all $i > j$; and $\mathsf{m}_i \xrightarrow{\mathsf{b}} \mathsf{m}_j$ for all i and j. Note that the relation $\xrightarrow{\mathsf{a}}$ is a subset of $\xrightarrow{\mathsf{b}}$. The model is shown in Fig. 1.

Consider the vectorial sentence $\nu_1^\omega(x_1, x_2).(\langle\mathsf{b}\rangle x_2, \langle\mathsf{a}\rangle x_2)$. This describes the property *there are arbitrarily long paths with labels in* ba^*, and so it is true in all points m_i and false in all points n_i. The following result immediately implies that this property cannot be defined in the scalar countdown calculus:

▶ **Theorem 11.** *For every scalar sentence φ, there exists $i < \omega$ s.t.* $\mathsf{n}_i \in [\![\varphi]\!] \iff \mathsf{m}_i \in [\![\varphi]\!]$.

Proof (sketch). The heart of the proof is Proposition 10 which says that scalar formulas correspond to injectively ranked automata. In such an automaton, whenever the counter corresponding to rank r is modified, the automaton must be in *the same state*, which allows the players to copy their strategies between different positions of the semantic game. ◀

7 Strictness of the countdown nesting hierarchy

A natural question is whether greater *coutdown nesting*, i.e. the maximal nesting of μ^α and ν^α operators with $\alpha \neq \infty$, results in more expressive power. We give a positive answer: under mild assumptions, the hierarchy is strict. From now on, focus on the monomodal case (i.e. $|\mathsf{Act}| = 1$) and we assume that the only ordinal used by formulas is ω.[1]

▶ **Theorem 12.** *For every $k < \omega$, formulas with countdown nesting $k + 1$ have strictly more expressive power than those with nesting at most k.*

In order to prove strictness, it suffices to prove it on a restricted class of models. We will show that the hierarchy is strict already on the class of transitive, linear, well-founded models – i.e. (up to isomorphism) ordinals.

More specifically, an ordinal $\kappa \in \mathsf{Ord}$ can be seen as a model with $\alpha \to \beta$ iff $\alpha > \beta$. Since κ is an induced submodel of κ' whenever $\kappa \leq \kappa'$, we can consider a single ordinal model with κ big enough. For our purposes, the first uncountable ordinal ω_1 is be sufficient.

We call a subset $S \subseteq \omega_1$ *stable above* α if either $[\alpha, \omega_1) \subseteq S$ or $[\alpha, \omega_1) \cap S = \emptyset$. A *stabilization point* of a valuation $\mathsf{val} : \mathsf{Var} \to \mathcal{P}(\omega_1)$ is the least $\alpha \leq \omega_1$ such that interpretations of all the variables are stable above α.

Observe that the set $[\omega^k, \omega_1) \subseteq [0, \omega_1)$ can be defined by the following sentence with countdown nesting k:

$$[\omega^k, \omega_1) = [\![\nu^\omega x_1 ... \nu^\omega x_k . \Diamond(\textstyle\bigwedge_{i \leq k} x_i)]\!]. \tag{6}$$

Indeed, the semantic game can be decomposed into two alternating phases: (i) ∀dam chooses a tuple of finite ordinals $(\alpha_1, ..., \alpha_k) \in \omega^k$ and (ii) ∃ve responds with a successor in the model. Since at each step ∀dam has to pick a lexicographically smaller tuple (and he starts by picking any tuple) it is easy to see that he wins iff the initial point is at least ω^k. We will show that for all $k > 0$, countdown nesting k is *necessary* to define this language. The proof relies on the following lemma.

▶ **Lemma 13.** *For every $k < \omega$ and a formula φ with countdown nesting k, there exists an ordinal $\alpha_\varphi < \omega^{k+1}$ such that φ stabilizes α_φ above the valuation, i.e. for every valuation* val *stabilizing at β, $[\![\varphi]\!]^{\mathsf{val}}$ is stable above $\beta + \alpha_\varphi$.*

Proof (sketch). By induction on the complexity of the formula φ. The base case is immediate, as for every $x \in \mathsf{Var}$ it suffices to take $\alpha_x = 0$. For propositional connectives and modal operators we take $\alpha_{\psi_1 \vee \psi_2} = \alpha_{\psi_1 \wedge \psi_2} = \mathsf{max}(\alpha_{\psi_1}, \alpha_{\psi_2})$ and $\alpha_{\Diamond\psi} = \alpha_{\Box\psi} = \alpha_\psi + 1$. The most interesting case are countdown and fixpoint operators. There the lemma follows from the fact that for every formula φ there is a finite constant $t_\varphi < \omega$ such that for every valuation val stable above κ, in the part $[\kappa, \omega_1)$ of the model above κ, φ changes its truth value at most t_φ times. ◀

[1] This assumption could be replaced with a weaker requirement: there exists a maximal ordinal α that we are allowed to use, and α is additively indecomposable.

From this the theorem follows immediately, as the sentence φ has no free variables and thus it stabilizes at $\alpha_\varphi < \omega^{k+1}$ regardless of the valuation.

8 Decidability issues

We briefly discuss decidability issues in the countdown μ-calculus. Note that in a finite model every monotone map reaches its fixpoints in finitely many steps. Hence, if we replace every η^α in φ with η^∞ and denote the resulting formula by $\widehat{\varphi}$, then *in every finite model* $[\![\varphi]\!] = [\![\widehat{\varphi}]\!]$. It immediately follows that:

▶ **Proposition 14.** *The model checking problem for the μ^α-ML, i.e. the problem: "Given $\varphi \in \mu^\alpha$-ML and a point m in a (finite) model $\mathcal{M}$, does $\mathsf{m} \models \varphi$?" is decidable.*

Note that as a corollary we get that deciding the winner of a given (finite) countdown game $\mathcal{G}$ is also decidable, as set of positions where ∃ve wins can be easily defined in μ^α-ML.

A more interesting problem is *satisfiability*: "Given $\varphi \in \mu^\alpha$-ML, is there a model $\mathcal{M}$ and a point m s.t. $\mathsf{m} \models \varphi$?".

▶ **Proposition 15.** *A formula $\varphi \in \mu^\alpha$-ML has* positive countdown *if it does not use ν^α with $\alpha \neq \infty$. The satisfiability problem is decidable for such formulas.*

Proof. Observe that for φ with positive countdown, in every model we have $[\![\varphi]\!] \subseteq [\![\widehat{\varphi}]\!]$. Hence, if φ is satisfiable, then so is $\widehat{\varphi}$ – but since μ-ML has a finite model property, this means that $\widehat{\varphi}$ has a *finite* model, where $\widehat{\varphi}$ and φ are equivalent. Thus, φ is satisfiable iff $\widehat{\varphi}$ is, and the problem reduces to μ-ML satisfiability. ◀

Dualizing the above we get that the *validity* problem is decidable for formulas with *negative countdown*, i.e. with $\alpha = \infty$ for every μ^α.

References

1 André Arnold and Damian Niwinski. *Rudiments of calculus.* Elsevier, 2001.

2 Mikołaj Bojańczyk. Weak mso with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011. doi:10.1007/s00224-010-9279-2.

3 Mikołaj Bojańczyk, Edon Kelmendi, Rafał Stefański, and Georg Zetzsche. Extensions of ω-regular languages. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 266–272, 2020. doi:10.1145/3373718.3394779.

4 Mikolaj Bojanczyk, Pawel Parys, and Szymon Torunczyk. The MSO+U Theory of (N,<) Is Undecidable. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:8, 2016. doi:10.4230/LIPIcs.STACS.2016.21.

5 Mikołaj Bojanczyk and Szymon Torunczyk. Weak MSO+U over infinite trees. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 648–660, 2012. doi:10.4230/LIPIcs.STACS.2012.648.

6 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic*, pages 41–55, September 2004. doi:10.1007/978-3-540-30124-0_7.

7 Mikołaj Bojańczyk and Thomas Colcombet. Bounds in w-regularity. In *Proceedings - Symposium on Logic in Computer Science*, pages 285–296, January 2006. doi:10.1109/LICS.2006.17.

8 Mikołaj Bojańczyk, Laure Daviaud, Bruno Guillon, Vincent Penelle, and A. V. Sreejith. Undecidability of mso+"ultimately periodic". *ArXiv*, abs/1807.08506, 2018.

9 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 0(0), 2017. doi:10.1137/17M1145288.

10 Thomas Colcombet. Regular Cost Functions, Part I: Logic and Algebra over Words. *Logical Methods in Computer Science*, Volume 9, Issue 3, 2013. doi:10.2168/LMCS-9(3:3)2013.

11 Lauri Hella, Antti Kuusisto, and Raine Rönnholm. Bounded game-theoretic semantics for modal mu-calculus. In *GandALF*, 2020.

12 Szczepan Hummel and Michał Skrzypczak. The topological complexity of mso+u and related automata models. *Fundamenta Informaticae*, 119:87–111, 2012.

13 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory*, pages 263–277, 1996.

14 Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

15 Michał Skrzypczak. *Descriptive Set Theoretic Methods in Automata Theory – Decidability and Topological Complexity*, volume 9802 of *Lecture Notes in Computer Science*. Springer, 2016. doi:10.1007/978-3-662-52947-8.

16 Yde Venema. Lectures on the modal μ-calculus, 2020.

17 Igor Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001. doi:10.1006/inco.2000.2894.

Rabbits Approximate, Cows Compute Exactly!

Balagopal Komarath ✉
IIT Gandhinagar, India

Anurag Pandey ✉
Department of Computer Science, Saarland University, Saarland Informatics Campus, Germany

Nitin Saurabh ✉
Department of Computer Science and Engineering, IIT Hyderabad, India

Abstract

Valiant, in his seminal paper in 1979, showed an efficient simulation of algebraic formulas by determinants, showing that VF, the class of polynomial families computable by polynomial-sized algebraic formulas, is contained in VDet, the class of polynomial families computable by polynomial-sized determinants. Whether this containment is strict has been a long-standing open problem. We show that algebraic formulas can in fact be efficiently simulated by the determinant of tetradiagonal matrices, transforming the open problem into a problem about determinant of general matrices versus determinant of tetradiagonal matrices with just three non-zero diagonals. This is also optimal in a sense that we cannot hope to get the same result for matrices with only two non-zero diagonals or even tridiagonal matrices, thanks to Allender and Wang (Computational Complexity'16) which showed that the determinant of tridiagonal matrices cannot even compute simple polynomials like $x_1x_2 + x_3x_4 + \cdots + x_{15}x_{16}$.

Our proof involves a structural refinement of the simulation of algebraic formulas by width-3 algebraic branching programs by Ben-Or and Cleve (SIAM Journal of Computing'92). The tetradiagonal matrices we obtain in our proof are also structurally very similar to the tridiagonal matrices of Bringmann, Ikenmeyer and Zuiddam (JACM'18) which showed that, if we allow approximations in the sense of geometric complexity theory, algebraic formulas can be efficiently simulated by the determinant of tridiagonal matrices of a very special form, namely the continuant polynomial. The continuant polynomial family is closely related to the Fibonacci sequence, which was used to model the breeding of rabbits. The determinants of our tetradiagonal matrices, in comparison, is closely related to Narayana's cows sequences, which was originally used to model the breeding of cows. Our result shows that the need for approximation can be eliminated by using Narayana's cows polynomials instead of continuant polynomials, or equivalently, shifting one of the outer diagonals of a tridiagonal matrix one place away from the center.

Conversely, we observe that the determinant (or, permanent) of band matrices can be computed by polynomial-sized algebraic formulas when the bandwidth is bounded by a constant, showing that the determinant (or, permanent) of bandwidth k matrices for all constants $k \geq 2$ yield VF-complete polynomial families. In particular, this implies that the determinant of tetradiagonal matrices in general and Narayana's cows polynomials in particular yield complete polynomial families for the class VF.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Computational complexity and cryptography; Theory of computation → Circuit complexity

Keywords and phrases Algebraic complexity theory, Algebraic complexity classes, Determinant versus permanent, Algebraic formulas, Algebraic branching programs, Band matrices, Tridiagonal matrices, Tetradiagonal matrices, Continuant, Narayana's cow sequence, Padovan sequence

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.65

Related Version *Full Version*: `https://nitinsau.github.io/pubs/cow-sequences.pdf` [13]

Acknowledgements We thank Arkadev Chattopadhyay for herding us towards this beautiful problem and sharing his insights! We also thank Meena Mahajan for bringing the reference [3] to our notice. We also thank the organizers of GCT2022 workshop for hosting a talk by Avi Wigderson which prompted us towards this investigation. We thank anonymous reviewers for helpful suggestions.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 65; pp. 65:1–65:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Valiant in his seminal work [20] laid the foundation for investigation of algebraic analog of the P versus NP problem, the flagship problem of theoretical computer science. He introduced algebraic formulas and determinants as models for computing polynomial families and identified them as notions of efficient computation, while the permanent family, $\mathrm{per}_n(x_{11},\ldots,x_{nn}) := \sum_{\sigma\in\mathbb{S}_n}\prod_{i=1}^{n} x_{i,\sigma(i)}$ was identified as a family that is highly likely to be hard to compute. He defined the complexity class VF as the set of polynomial families that can be computed by formulas of polynomially-bounded size, and VDet as the set of families that can be expressed as the determinant of a symbolic matrix of polynomially-bounded dimension. He also showed, among other things, that a polynomial computable by an algebraic formula of size s can be expressed as the determinant of a symbolic matrix of size $(s+2)\times(s+2)$, thus showing the containment $\mathsf{VF}\subseteq\mathsf{VDet}$. Conversely, the smallest known formulas for the determinant family, $\det_n(x_{11},\ldots,x_{nn}) := \sum_{\sigma\in\mathbb{S}_n}\mathrm{sgn}(\sigma)\prod_{i=1}^{n} x_{i,\sigma(i)}$, have size $n^{O(\log n)}$ [11, 6]. Thus the two notions of efficient computation are not known to be equivalent. It is a long standing open problem whether algebraic formulas of polynomial size exist for the determinant family.

▶ **Problem 1.** *Is the determinant family strictly more expressive than algebraic formulas? In other words, is* $\mathsf{VF}\subsetneq\mathsf{VDet}$*?*

An improved construction of a formula for the determinant family has resisted all attempts for long, which can be interpreted as an evidence to an affirmative answer to Problem 1. Though the relationship between the classes VF and VDet is poorly understood as of now, they themselves are very natural otherwise. Not only they contain many natural examples of polynomial families, there are many differing, but equivalent, ways to define them too.

For example, the class VDet is equivalently captured by the model of algebraic branching programs of polynomial size, denoted VBP. Recall, an algebraic branching program (ABP) is a directed acyclic graph G with two special nodes, say s (source node) and t (sink node), and edges labeled with variables or constants. For every s-to-t path p in G we associate a monomial m_p obtained by multiplying the edge labels on this path. The polynomial computed by the algebraic branching program G is defined to be the sum over all monomials given by s-to-t paths, i.e., $\sum_{p:\ s\text{-to-}t\text{ path }p} m_p$. Rephrasing the characterization, we know $\mathsf{VDet}=\mathsf{VBP}$. We can assume, wlog, branching programs to be layered, i.e., the vertices are topologically ordered in layers, from left to right, such that the edges only go between consecutive layers. Then the *width* of a branching program is defined to be the maximum number of vertices in any one layer.

In an influential work, Ben-Or and Cleve [5] showed that branching programs of constant width characterize formulas. In other words, they showed $\mathsf{VF}=\mathsf{VBP}_3$, where VBP_3 denotes the class of algebraic branching programs of width 3 and polynomial size. In light of this, Problem 1 can be rephrased as asking whether $\mathsf{VBP}_3\subsetneq\mathsf{VBP}$, that is, whether algebraic branching programs of width 3 are computationally strictly weaker than algebraic branching programs of arbitrary width. This seems even more likely when phrased this way!

In a recent work, Bringmann, Ikenmeyer and Zuiddam [8] took this one step further by showing that the topological closure of VF is equivalent to the topological closure of VBP_2, i.e. $\overline{\mathsf{VF}}=\overline{\mathsf{VBP}_2}$, where VBP_2 is the class corresponding to algebraic branching programs of width 2! Stated differently, they showed that algebraic branching programs of width 2 can efficiently *approximate* all polynomials that are efficiently computed (or, approximated) by algebraic formulas. In fact, the equivalent width 2 algebraic branching programs given by the reduction have very special structure, which make them equivalent to the determinant of tridiagonal symbolic matrices of a very special form. These tridiagonal matrices have non-trivial entries, variables and constants, on the main diagonal while the other two diagonals are fixed to all ±1s. Determinant of such tridiagonal symbolic matrices is well-studied in the literature and is known as the *continuant*, deriving its name from continued fractions since continuants are used to represent the convergents of continued fractions. They are also related to the Fibonacci sequence via the following recursive definition: $F_0 := 1$, $F_1 := x_1$, and $F_n := x_n F_{n-1} + F_{n-2}$ for all $n \geq 2$. Thus, for a positive resolution of Problem 1, it is sufficient to show that the determinant of certain family of tridiagonal matrices, namely the continuant family $\{F_n\}$, *cannot* efficiently approximate the determinant of general matrices.

The continuant is known to have rich algebraic structures [16, 10, 9, 17], which may be helpful in separating VF from VDet. Although quite promising, an additional challenge this formulation poses is that we now need to deal with approximations. In other words, we need to show a stronger separation $\overline{\mathsf{VF}} \subsetneq \mathsf{VDet}$. It would be very pleasing if we could have the result of Bringmann, Ikenmeyer and Zuiddam [8] without using approximations. That is, if the following would be true – the continuant family $\{F_n\}$ can *efficiently exactly* simulate formulas. However, such a result is an impossibility! Allender and Wang [4] showed that the simple polynomial, $x_1x_2 + x_3x_4 + \cdots + x_{15}x_{16}$, cannot even be expressed by the continuant family, irrespective of efficiency. Thus, one may wonder what is the *simplest* class of matrices whose determinants can *efficiently exactly* simulate algebraic formulas?

Motivated by this question, we study the determinant of matrices with few diagonals, also known as band matrices, and identify two polynomial families that are as simple as the continuant family $\{F_n\}$, but unlike it they simulate formulas exactly and efficiently.

The Narayana's cows polynomial. The m-th polynomial in this family, denoted $N_m(x_1, \ldots, x_m)$, is defined by the recurrence $N_0 := 1$, $N_1 := x_1$, $N_2 := x_1x_2$, and $N_m = x_m N_{m-1} + N_{m-3}$ for all $m \geq 3$. Just as the continuant polynomial is based on the Fibonacci sequence, the Narayana's cows polynomial is based on the Narayana's cows sequence [1, 21]. This sequence originated in the following problem studied by the 14-th century mathematician Narayana Pandita in his book Ganita Kaumudi [18]: *A cow produces a calf every year. Cows start producing calves from the beginning of the fourth year. Then, starting from* 1 *cow in the first year, how many cows are there after m years?* This sequence is given by the recurrence: $N_m = N_{m-1} + N_{m-3}$ with $N_0 = N_1 = N_2 = 1$, where N_{m-1} gives the population after m years. Thus, the sequence captures the growth in the population of cows in the same way as the Fibonacci sequence captures the growth in the population of rabbits. The Narayana's cows sequence has wide applications in combinatorics. (See, e.g., [1, 14] and references therein.)

The Padovan polynomial. The recurrences for Fibonacci and Narayana's cows sequences are similar. Exploring this similarity and considering the only remaining two-term recurrence: $P_n = P_{n-2} + P_{n-3}$, we obtain another lesser known cousin of Fibonacci, called as the Padovan sequence [2, 23, 19]. Analogously, we can define the Padovan polynomial via the recurrence $P_0 := 1$, $P_1 := 0$, $P_2 := x_1$, and $P_n = x_{n-1}P_{n-2} + P_{n-3}$ for all $n \geq 3$. This generalizes the univariate Padovan polynomial that is known in the literature [22].

Our results complement the results of Bringmann, Ikenmeyer and Zuiddam [8] by showing that the aforementioned polynomial families, namely Narayana's cows and Padovan, based on the lesser known cousins of Fibonacci, are complete for the class VF. In other words, both families can efficiently exactly simulate formulas.

$$\begin{pmatrix} x_1 & -1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & 1 & x_n \end{pmatrix}$$

(a) Continuant polynomial $F_n := x_nF_{n-1} + F_{n-2}$.

$$\begin{pmatrix} x_1 & 0 & 1 & & \\ 1 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & \ddots & \ddots & 0 \\ & & & 1 & x_m \end{pmatrix}$$

(b) Narayana's cows polynomial $N_m := x_mN_{m-1} + N_{m-3}$.

$$\begin{pmatrix} 0 & -x_1 & 1 & & \\ 1 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & \ddots & \ddots & -x_{n-1} \\ & & & 1 & 0 \end{pmatrix}$$

(c) Padovan polynomial $P_n := x_{n-1}P_{n-2} + P_{n-3}$.

Figure 1 Polynomial families defined by determinants of simple matrices and their recurrences.

1.1 Our findings

We discover the simplest class of matrices whose determinants characterize algebraic formulas. We find that tetradiagonal matrices of a very special form suffice for this purpose.

▶ **Theorem 2** (Informal, See Theorem 16 and Corollary 23). *The determinant family of tetradiagonal symbolic matrices is polynomially equivalent to algebraic formulas.*

In fact, the tetradiagonal matrices (Figures 1b and 1c) that is sufficient for efficiently simulating algebraic formulas are remarkably similar to the tridiagonal matrices (Figure 1a) used by Bringmann, Ikenmeyer and Zuiddam [8] to efficiently approximate algebraic formulas. It follows from the above theorem and Allender and Wang's separation [4], that tetradiagonal matrices are more expressive than tridiagonal matrices, but at the same time it can also be seen (Figure 1) to be nearly as simple as tridiagonal matrices – having just one extra diagonal whose entries are all 0s! We thus have the following equivalent reformulation of Problem 1.

> *Is the minimum size of a tetradiagonal matrix whose determinant equals* $\det_n$ *superpolynomially large, where* $\det_n$ *is the determinant of a general* $n \times n$ *symbolic matrix?*

This further motivated us to investigate matrices with few non-zero diagonals. Such matrices are called *band matrices* in the literature. We say that a matrix M is a band matrix of type (k_1, k_2) if all the non-zero entries of the matrix is concentrated between k_1 diagonals below the main diagonal and k_2 diagonals above the main diagonal. That is, $M_{ij} = 0$ if $j < i - k_1$ or $j > i + k_2$. A band matrix of type (k_1, k_2) will also be referred as (k_1, k_2)-diagonal matrix. The bandwidth of such matrices are defined to be $k := \max(k_1, k_2)$.

For example, diagonal matrices are $(0, 0)$-diagonal and has bandwidth 0, tridiagonal matrices are $(1, 1)$-diagonal with bandwidth 1, tetradiagonal matrices are either $(1, 2)$-diagonal or $(2, 1)$-diagonal with bandwidth 2, and pentadiagonal matrices are $(2, 2)$-diagonal with bandwidth 2. Figures 1b and 1c are examples of $(1, 2)$-diagonal matrices.

If follows from Theorem 2 that $(1, 2)$-diagonal matrices can simulate formulas, and hence any (k_1, k_2)-diagonal matrix can simulate formulas as long as $\min(k_1, k_2) \geq 1$ and $\max(k_1, k_2) \geq 2$. It is then interesting to investigate the converse, i.e., for which (k_1, k_2)-diagonal matrices their determinants have small formulas?

We observe that determinants of bandwidth k matrices can be computed by polynomial-sized algebraic formulas when the bandwidth k is bounded by a constant. In fact, our constructions give efficient *(syntactic) multilinear* ABPs and circuits for low bandwidth matrices. These are circuits for which *every* intermediate polynomial that is computed is also multilinear. A polynomial is said to be *multilinear* if every monomial of the polynomial is multilinear, and a monomial is called *multilinear* if every variable has degree at most 1 in it. In comparison, polynomial size circuits for the determinant of general matrices given by Berkowitz [6] and polynomial size ABPs given by Mahajan and Vinay [15] are non-multilinear.

▶ **Theorem 3** (Informal, See Corollary 23). *Determinants of symbolic band matrices are computable by polynomial-sized algebraic formulas when bandwidth is bounded by a constant.*

In fact, the above theorem holds for the permanent of a band matrix too. Combining Theorems 2 and 3, we get a nice characterization of algebraic formulas in terms of determinants (or, permanents) of band matrices of small bandwidth. In other words, determinants of band matrices with bounded bandwidth yield polynomial families which are complete for the complexity class VF.

▶ **Theorem 4** (Informal, See Theorem 16, Theorem 19, and Corollary 23). *For all constant $k \geq 2$, the determinant (or, permanent) family of symbolic matrices of bandwidth k is* VF*-complete.*

1.2 Proof methods

Ideas for Theorem 2 (Simulating formulas via determinant of tetradiagonal matrices). We prove Theorem 2 in Section 3, where we begin with tetradiagonal matrices of type $(1, 2)$. That is, the non-zero entries are limited to one diagonal below the main diagonal, the main diagonal, and two diagonals above the main diagonal. We first show that the symbolic determinant of such tetradiagonal matrices can be written as a product of 3×3 matrices whose entries are variables (or their negations), 0, and 1, where the number of matrices in the product is linear in the size of the original matrix. This is obtained by exploiting a simple recurrence revealed while computing the determinant of these $(1, 2)$ tetradiagonal matrices using Laplace expansion, see Lemma 12. Thus, to prove Theorem 2, it is sufficient to show that algebraic formulas can be efficiently simulated by the matrix product of the 3×3 matrices obtained above. In fact, Ben-Or and Cleve, in their simulation of algebraic formulas using width 3 algebraic branching programs, showed that algebraic formulas can be efficiently simulated by the matrix product of 3×3 matrices. Thus, it might be tempting to conclude that we are already done. However, it turns out that the 3×3 matrices whose products equals the determinant of tetradiagonal matrices desire more structure than the matrices used in the proof of Ben-Or and Cleve. This is where the core technical novelty of our work lies – we show that algebraic formulas can indeed be efficiently simulated by product of 3×3 matrices of the form whose products are equivalent to the determinant of $(1, 2)$-tetradiagonal matrices. In fact, we are able to efficiently simulate formulas with even more structure on the matrices, allowing us to conclude that formulas can be efficiently simulated by tetradigonal matrices where the variable entries are only on the main diagonal, the diagonal below the main diagonal is all 1s, whereas the two diagonals above the main diagonal are all 0s and all 1s respectively, see Section 3.1 for details.

Ideas for Theorem 3 (Formulas for determinant of symbolic band matrices). Theorem 3 is relatively simpler to derive from the literature. We prove it in Section 4 taking two different constructions for computing determinants of general matrices and carefully specializing those constructions in the case of bandwidth k matrices, ensuring that the undesirable blowups are limited to parameter k, allowing us to get polynomial-sized formulas when k is bounded by a constant. In our first construction, we modify the construction of Grenet for computing permanent of an $n \times n$ matrix using algebraic branching programs. For bandwidth k matrices, we are able to get syntactic multilinear ABPs of length linear in the size of matrix and exponential in the bandwidth, see Theorem 22 for details. Applying standard conversion from ABPs to formulas yield Theorem 3. This gives us a formula of depth $O(k \log(n))$ and size $n^{O(k)}$. In our second construction, we adapt the generalized Laplace expansion to low bandwidth matrices, see Theorem 26 for details. The construction yields a syntactic multilinear arithmetic circuit of size $O(\exp(k)n)$ and depth $O(\mathrm{poly}(k)\log(n))$, which can be converted to algebraic formulas using standard conversion from circuits to formulas, giving an alternative proof of Theorem 3.

The rest of this paper is organized as follows: Section 3 gives efficient simulations of algebraic formulas via determinant of tetradiagonal symbolic matrices. Subsections 3.1 and 3.2 show that Narayana's cows polynomials and Padovan polynomials are complete for VF. Section 4 shows that determinants of all matrices with constant bandwidth have polynomial size formulas. See the full version [13] for omitted proofs.

2 Preliminaries

We define computational models that are of interest in this paper.

▶ **Definition 5.** *An* algebraic circuit *C is a rooted directed acyclic graph where the source nodes are labeled by elements of $\mathbb{F}$ or variables $x_1, \ldots, x_n$, and the internal nodes have in-degree 2 and are labeled by $+$ or $\times$. It naturally computes a polynomial $p \in \mathbb{F}[x_1, \ldots, x_n]$ in a bottom-up fashion. An* algebraic formula *is a circuit whose underlying graph is a tree. The* size *of a circuit is the number of nodes in the graph and the* depth *of a circuit is the number of edges on the longest path from the root to some source node.*

We recall the definition of ABPs.

▶ **Definition 6.** *An ABP is a layered directed acyclic graph with source node s and sink node t such that each edge is labeled by a variable or a constant. The polynomial computed by the ABP is given by $\sum_p m_p$, where p is an s to t path and m_p is the product of edge labels on the path p. The* width *of an ABP is the maximum number of nodes in any layer and the* length *is the number of layers.*

It is easy to see that width-w, length-ℓ ABPs are equivalent to product of a sequence of ℓ matrices of order $w \times w$.

We now define a notion of reduction that allows us to relate the complexity of polynomials under the above model.

▶ **Definition 7.** *A polynomial $f(x) \in \mathbb{F}[x_1, \ldots, x_n]$ is a* projection *of a polynomial $g(y) \in \mathbb{F}[y_1, \ldots, y_m]$, denoted $f \leq g$, if and only if $f(x_1, \ldots, x_n) = g(a_1, \ldots, a_m)$, where $a_i \in \mathbb{F} \cup \{x_1, x_2, \ldots, x_n\}$.*

It is easy to see that if g is computed by a formula of size s and depth d and $f \leq g$, then f is also computed by a formula of size at most s and depth at most d.

As is usually the case in algorithms and complexity, formula size or depth for fixed polynomials is rarely of interest. Instead, we look at families of polynomials and the asymptotic growth of size and depth of formulas computing them.

▶ **Definition 8.** *A* polynomial family *over a field $\mathbb{F}$ is a sequence $f = (f_n)_{n\in\mathbb{N}}$ of polynomials such that both the number of variables and the degree of f_n are polynomially bounded functions in n.*

A sequence of formulas $F = (F_n)_{n\in\mathbb{N}}$ is said to compute a polynomial family $f = (f_n)$ if and only if F_n computes f_n for all n. The size *and* depth *of the sequence F is defined as functions such that $s(n)$ and $d(n)$ are the size and depth of F_n.*

We now extend the definition of reductions to families.

▶ **Definition 9.** *A polynomial family (f_n) is a p-*projection *of another family (g_m), denoted $(f_n) \leq_p (g_m)$ if and only if there exists a polynomially bounded function $t : \mathbb{N} \mapsto \mathbb{N}$ such that $f_n \leq g_{t(n)}$ for all n.*

Note that if (g_n) is computed by polynomial size formulas and $(f_n) \leq_p (g_n)$, then (f_n) is also computed by polynomial size formulas.

▶ **Definition 10.** *The algebraic class* VF *is defined as the set of all polynomial families that have polynomial size formulas.*

The class VDet *(equivalently* VBP*) is defined as the set of all polynomial families f such that $f \leq_p \det$, where $\det$ is the family of determinants over $n \times n$ symbolic matrices.*

Using the above notion of reduction, we can define completeness for classes.

▶ **Definition 11.** *A polynomial family $f = (f_n)$ is said to be* complete *for a class $\mathcal{C}$ w.r.t p-projections if and only if $f \in \mathcal{C}$ and for all $g \in \mathcal{C}$, we have $g \leq_p f$.*

Note that det is trivially complete for VDet. Ben-Or and Cleve [5] showed that the polynomial family $\mathrm{per}_{3,n}$, the $(1,1)$ entry of the product of n 3×3 symbolic matrices is complete for VF.

3 Determinant of $(1,2)$-diagonal matrix versus algebraic formulas

In this section, we show that the determinants of $(1,2)$-diagonal symbolic matrices are polynomially equivalent to algebraic formulas, thereby, proving Theorem 2. We begin with the easier direction, that is, by showing that the determinant of $(1,2)$-diagonal symbolic matrix has polynomial-sized algebraic formulas. In fact, we give a polynomial-sized algebraic branching programs for them of width-3, which can then be converted into a polynomial-sized formula using a divide and conquer algorithm.

▶ **Lemma 12.** *The determinant (or, permanent) of $(1,2)$-diagonal symbolic matrix of dimensions $n \times n$ can be computed by a width-3 syntactic multilinear ABP of length at most $3n - 2$.*

In particular, they can be computed by (syntactic) multilinear formulas of size poly(n)*.*

Proof. Let M denote the following $(1,2)$-diagonal symbolic matrix,

$$M = \begin{pmatrix} x_{11} & x_{12} & x_{13} & & \\ x_{21} & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & x_{n-2,n} \\ & & \ddots & \ddots & x_{n-1,n} \\ & & & x_{n,n-1} & x_{n,n} \end{pmatrix}. \tag{1}$$

For $0 \leq i \leq n-1$, define $K(n-i)$ to be the determinant of the principal submatrix of M obtained by deleting both the first i rows and columns. Furthermore, set $K(0) := 1$ and $K(-1) := 0$. Note that, by definition, $K(n) = \det(M)$ and $K(1) = x_{nn}$. Then we have the following recursive formula for $K(n)$:

$$K(n) \quad = x_{11}K(n-1) - x_{12}x_{21}K(n-2) + x_{13}x_{32}x_{21}K(n-3). \tag{2}$$

The correctness of the above formula easily follows from a backward induction on i. Rewriting the recurrence in a matrix form we obtain

$$\begin{bmatrix} K(n) \\ K(n-1) \\ K(n-2) \end{bmatrix} = \begin{bmatrix} x_{11} & -x_{12}x_{21} & x_{13}x_{32}x_{21} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} K(n-1) \\ K(n-2) \\ K(n-3) \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} x_{21} & x_{11} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -x_{12} & x_{13} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & x_{32} \end{bmatrix} \begin{bmatrix} K(n-1) \\ K(n-2) \\ K(n-3) \end{bmatrix} \tag{4}$$

Unrolling Eq. (4) and using $K(1) = x_{nn}$, $K(0) = 1$, and $K(-1) = 0$ we obtain the claimed width-3 ABP for $K(n)$. ◀

We now consider a special kind of $(1, 2)$-diagonal symbolic matrices where entries in both the lowermost and the uppermost diagonals are only 1. We show that the determinant (or, permanent) of such a matrix is equivalent to a special kind of width-3 ABP. These matrices would serve as the key building block in our main proofs. However, we first need a name for the special kind of $(1, 2)$-diagonal matrices that we are going to be dealing with.

▶ **Definition 13.** *Let* $(\alpha, \beta, \gamma, \delta) \in (\mathbb{F} \cup \{*\})^4$. *A* $(1, 2)$*-diagonal matrix is said to be of type* $(\alpha, \beta, \gamma, \delta)$ *if all entries on the lowermost diagonal, main diagonal, first upper diagonal and second upper diagonal equals* α*,* β*,* γ *and* δ *respectively. Furthermore, if* α*,* β*,* γ*, or* δ *equals* $*$ *then the entries on the respective diagonals are* ***not*** *restricted.*

For example, a general $(1, 2)$-diagonal symbolic matrix, shown in Equation 1, is of type $(*, *, *, *)$ and a $(1, 2)$-diagonal matrix of type $(\alpha, \beta, \gamma, \delta) \in \mathbb{F}^4$ is also a *Toeplitz* matrix. The special kind of $(1, 2)$-diagonal matrices that we consider are of type $(1, *, *, 1)$. We now characterize the determinant of such matrices by a restricted width-3 ABP where the interconnections between layers are given by a special 3×3 matrix.

▶ **Lemma 14.** *Let* M *denote the following* $(1, 2)$*-diagonal symbolic matrix of type* $(1, *, *, 1)$ *of dimension* $n \times n$*:*

$$M = \begin{pmatrix} x_{11} & x_{12} & 1 & & \\ 1 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & \ddots & \ddots & x_{n-1,n} \\ & & & 1 & x_{n,n} \end{pmatrix}.$$

Then, $\det(M)$ *is given by the* $(1, 1)$ *entry of the following iterated matrix multiplication over* 3×3 *matrices,*

$$\begin{bmatrix} x_{11} & -x_{12} & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{22} & -x_{23} & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdots\cdots \begin{bmatrix} x_{(n-1)(n-1)} & -x_{(n-1)n} & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{nn} & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Conversely, the $(1,1)$ *entry of the following iterated matrix multiplication:*

$$\begin{bmatrix}\alpha_1 & \beta_1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\alpha_2 & \beta_2 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\cdots\cdots\begin{bmatrix}\alpha_{(n-1)} & \beta_{(n-1)} & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\alpha_n & \beta_n & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix},$$

is given by the determinant of the following $(1,2)$*-diagonal matrix of type* $(1,*,*,1)$*,*

$$M = \begin{pmatrix} \alpha_1 & -\beta_1 & 1 & & \\ 1 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & \ddots & \ddots & -\beta_{n-1} \\ & & & 1 & \alpha_n \end{pmatrix}.$$

Proof. The equivalence follows from observing that in this special case the recurrence of (3) becomes

$$\begin{bmatrix}K(n)\\ K(n-1)\\ K(n-2)\end{bmatrix} = \begin{bmatrix}x_{11} & -x_{12} & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}K(n-1)\\ K(n-2)\\ K(n-3)\end{bmatrix},$$

$$= \begin{bmatrix}x_{11} & -x_{12} & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\cdots\begin{bmatrix}x_{(n-1)(n-1)} & -x_{(n-1)n} & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}K(1)\\ K(0)\\ K(-1)\end{bmatrix},$$

where $K(i), -1 \leq i \leq n$, as defined in the proof of Lemma 12, is the determinant of the principal submatrix of M obtained by deleting both the first $n-i$ rows and columns with $K(0) = 1$ and $K(-1) = 0$. ◀

3.1 Narayana's cows polynomial is VF-complete

In this section, we simulate algebraic formulas with tetradiagonal matrices of type (1,$*$,0,1). The determinant of such matrices follow the same recurrence as that of Narayana's cows polynomial described in Section 1. This simulation along with Lemma 12 finishes the proof of completeness of Narayana's cows polynomial families for the class VF.

We know from Lemma 14 that the determinant (or, permanent) of $(1,2)$-diagonal matrices of type $(1,*,0,1)$ is equivalent to the $(1,1)$ entry of an iterated matrix multiplication where the base matrices are of the form: $\begin{bmatrix}* & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$. For notational convenience, let us denote the base matrix $\begin{bmatrix}z & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$ by $A(z)$. In the following we will only work with iterated matrix multiplication over the base matrix $A(*)$ and use the equivalence given by Lemma 14 to represent the matrix product as the determinant (or, permanent) of $(1,2)$-diagonal matrix of type $(1,*,0,1)$.

For a better understanding of the algorithm we will present the algorithm in a recursive way. In particular we will have intermediate computations where the matrices will be of the form $\begin{bmatrix}0 & f & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$. We will denote such matrices by $B(f)$. Note that $A(0) = B(0)$. We now state and prove our simulation of formulas as a product of base matrices $A(z)$.

▶ **Lemma 15.** *Let p be a polynomial computed by a formula of depth d. Then, both $A(p)$ and $A(-p)$ can be expressed as an iterated matrix multiplication of length at most $30 \cdot 4^d - 29$ over the base matrices $A(z)$, where z is either a field constant, a variable, or a negated variable.*

Proof. The proof is by induction on depth.

Base case: $d = 0$. Then it computes either a field constant, a variable or a negated variable which can be represented by a single base matrix $A(z)$, where z is the label of the node.

Induction step: $d = m$. There are two cases to be considered depending on whether the node at depth m is an addition or a multiplication node.

Case 1: (Addition). Suppose $p = f + g$, where f and g are computable by depth $m - 1$ formulas. By induction hypothesis, we can express both $A(f)$ and $A(g)$. Then, $A(p) = A(f) \cdot A(0) \cdot A(0) \cdot A(g)$. In other words,

$$\begin{bmatrix} f+g & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} g & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Similarly one can express $A(-p)$.

Case 2: (Multiplication). Suppose $p = f \cdot g$, where f and g are computable by depth $m - 1$ formulas. We will use the following equation to compute $f \cdot g$.

$$A(f \cdot g) = A(0) \cdot A(0) \cdot B(-g) \cdot B(f) \cdot A(0) \cdot B(g) \cdot B(-f). \tag{5}$$

We will now show how to compute $B(f)$ using matrices of type $A(\cdot)$ which will complete the recursive algorithm. Similar to Eq. (5), the following equation computes $B(f \cdot g)$ using matrices of type $A(\cdot)$.

$$B(f \cdot g) = A(0) \cdot A(-f) \cdot A(0) \cdot A(g) \cdot A(f) \cdot A(0) \cdot A(-g). \tag{6}$$

We can thus use appropriate substitutions in Eq. (6) to get $B(f)$, $B(g)$, $B(-f)$, and $B(-g)$ and complete the algorithm. However, note that to compute $B(f)$ we need to make two calls to f as $A(-f)$ and $A(f)$. This would result in a total length of $O(8^d)$. To bring down the length to $O(4^d)$, we now show how to compute $B(f)$ using a single call to $A(f)$. Consider the following equation:

$$B(f) = A(0) \cdot B(-1) \cdot A(0) \cdot A(1) \cdot A(f) \cdot A(0) \cdot A(-1) \cdot B(1) \cdot A(0) \cdot A(0). \tag{7}$$

We can use Eq. (6) to obtain $B(-1)$ and $B(1)$, thus completing the algorithm to compute $B(f)$ with a single call to $A(f)$. We can now use equations (5) and (7) to compute $A(f \cdot g)$. Similarly one can express $A(-p)$.

The upper bound on the length of the iterated matrix multiplication follows from the following recurrence: $T(d) \leq 4 \cdot T(d-1) + 87$ and $T(0) = 1$. ◀

As a corollary to Lemmas 12 and 15, we obtain the following characterization of formulas.

▶ **Theorem 16.** *Let M_n denote the following $(1, 2)$-diagonal symbolic matrix of type $(1, *, 0, 1)$ of dimension $n \times n$:*

$$M_n = \begin{pmatrix} x_1 & 0 & 1 & & \\ 1 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & \ddots & \ddots & 0 \\ & & & 1 & x_n \end{pmatrix}.$$

Then the sequences of polynomials $\{\det(M_n)\}_{n \geq 1}$ and $\{\mathrm{per}(M_n)\}_{n \geq 1}$ are VF*-complete with respect to p-projections.*

Proof. Follows from Lemmas 12 and 15, and depth reduction of formulas [7]. ◀

We are now all set to deduce the completeness of the Narayana's Cows polynomial family for class VF.

▶ **Theorem 17.** *Narayana's cows polynomial family is* VF*-complete.*

Proof. We observe that the determinants of the sequence of $(1,2)$-diagonal symbolic matrices of type $(1,*,0,1)$ follow the recurrence $N_m = x_m N_{m-1} + N_{m-3}$ for all $m \geq 3$, which is precisely the recurrence defining the Narayana's cows polynomials as described in Section 1. ◀

3.2 Padovan polynomial is VF-complete

In this section, we simulate algebraic formulas with tetradiagonal matrices of type $(1,0,*,1)$ instead. This time, the determinant of such matrices follow the same recurrence as that of Padovan polynomial described in Section 1. This simulation along with Lemma 12 finishes the proof of completeness of Padovan polynomial families for the class VF.

Again from Lemma 14 we know that the determinant (or, permanent) of $(1,2)$-diagonal matrices of type $(1,0,*,1)$ is equivalent to the $(1,1)$ entry of an iterated matrix multiplication where the base matrices are of the form: $\begin{bmatrix} 0 & * & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$. Recall we denote base matrices of the form $\begin{bmatrix} 0 & z & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ by $B(z)$. In the following we will only work with iterated matrix multiplication over the base matrix $B(*)$ and use the equivalence given by Lemma 14 to represent the matrix product as the determinant (or, permanent) of $(1,2)$-diagonal matrix of type $(1,0,*,1)$.

▶ **Lemma 18.** *Let p be a polynomial computed by a formula of depth d. Then, both $B(p)$ and $B(-p)$ can be expressed as an iterated matrix multiplication of length at most $30 \cdot 4^d - 29$ over the base matrices $B(z)$, where z is either a field constant, a variable, or a negated variable.*

The proof is analogous to the proof of Lemma 15. (See the full version [13].)

As a corollary to Lemmas 12 and 18, we obtain another characterization of formulas.

▶ **Theorem 19.** *Let M_n denote the following $(1,2)$-diagonal symbolic matrix of type $(1,0,*,1)$ of dimension $n \times n$:*

$$M_n = \begin{pmatrix} 0 & x_1 & 1 & & \\ 1 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & \ddots & \ddots & x_{n-1} \\ & & & 1 & 0 \end{pmatrix}.$$

Then the sequences of polynomials $\{\det(M_n)\}_{n\geq 2}$ and $\{\mathrm{per}(M_n)\}_{n\geq 2}$ are VF*-complete with respect to p-projections.*

Proof. The containment in VF follows from Lemma 12. While the hardness follows by translating the iterated product in Lemma 18 to a $(1,2)$-diagonal symbolic matrix of type $(1,0,*,1)$ using Lemma 14. Note that to apply Lemma 14 one has to multiply the iterated product on the right by $B(0)$ (to move the polynomial to $(1,1)$ entry). However, this only increases the length by 1. Finally using the depth reduction of formulas [7] completes the proof. ◀

We are now all set to deduce the completeness of Padovan polynomial family for class VF.

▶ **Theorem 20.** *Padovan polynomial family is* VF*-complete.*

Proof. We observe that the determinants of the sequence of $(1,2)$-diagonal symbolic matrices of type $(1,0,*,1)$ in Theorem 19 follow the recurrence $P_n = x_{n-1}P_{n-2} + P_{n-3}$, for all $n \geq 3$, if we negate all variables in the matrix, which is precisely the recurrence for the Padovan polynomials as described in Section 1. ◀

4 Matrices of small bandwidth

Our main goal in this section is to prove that for all fixed k, the determinant of matrices of bandwidth k can be computed by polynomial sized formulas. Along with the results in Section 3, this gives a complete characterization of the algebraic complexity of the determinant of constant bandwidth matrices (Theorem 24). Following the spirit of parameterized algorithms, we consider the bandwidth k as a parameter, and show that we can construct efficient syntactic multilinear ABPs (Theorem 22) and circuits (Theorem 26) for computing the determinant where the undesirable blowup (exponential for size, polynomial for depth) is limited to the parameter k.

Our parameterized constructions are derived from Grenet's syntactic multilinear ABP construction for the $n \times n$ permanent [12] and the generalized Laplace expansion that constructs syntactic multilinear circuits for the $n \times n$ determinant and permanent. We state the bounds given by those constructions below:

▶ **Lemma 21.** *The determinant (or, permanent) of an $n \times n$ symbolic matrix can be computed by a syntactic multilinear circuit of size $O(n2^n)$ and depth $O(n)$. Moreover, it can be computed by a syntactic multilinear ABP of length at most $n+2$ and width at most $\binom{n}{n/2}$.*

Notice that the ABP in Lemma 21 has width that is exponential in n. Our construction for matrices of bandwidth k shows that this exponential blowup can be limited to k.

▶ **Theorem 22.** *The determinant (permanent) of a (k,k)-diagonal symbolic matrix of dimension $n \times n$ can be computed using a syntactic multilinear ABP of length $n+2$ and width $\binom{2k}{k}$.*

Proof. We begin with a high-level recall of Grenet's construction [12]. In his construction, the start node is in layer 0. All monomials computed at layer i correspond to some permutation that maps rows $[i]$ to some set of i columns. Further, a node in a particular layer keeps track of the subset of columns in the monomials computed at that node. This means that in layer $n/2$, it has to keep track of $\binom{n}{n/2}$ distinct sets resulting in exponential (in n) width. The edges between layers are specified such that these invariants are preserved.

We now build a layered ABP for small bandwidth matrices that is a modification of Grenet's construction.

For matrices of bandwidth k, we can make use of the fact that rows that are separated by at least $2k$ rows have no common non-zero columns. Therefore, instead of keeping track of a subset of all columns, we can keep track of a subset of only a few columns. More specifically, any monomial computed at layer i (assume $k \leq i \leq n-k$ for simplicity, the rest of the rows are handled similarly) must pick i columns from $[i+k]$ since all columns further to the right are zero for these rows. Moreover, the columns $[i-k]$ have to be picked by the first i rows since these columns are zero from row $i+1$. Therefore, rows up to i must pick exactly k columns from the $2k$ sized set of columns $[i-k+1, i+k]$. In layer i, we have exactly one node for each k sized subset of this $2k$ sized set. This ABP has $n + 2$ layers and each layer has at most $\binom{2k}{k}$ nodes. This is precisely where we improve over Grenet's construction when specialized to matrices of bandwidth k. We refer to the full version [13] for details. ◀

By using standard conversion from ABP to formula, we obtain the following corollary.

▶ **Corollary 23.** *For all fixed k, the determinant (or, permanent) of symbolic matrices of bandwidth k can be computed using polynomial sized formulas.*

Along with the results in Section 3, the above corollary gives a complete characterization of the algebraic complexity of determinant (or, permanent) of constant bandwidth matrices.

▶ **Theorem 24.** *For all constant $k \geq 2$, the determinant (or, permanent) family of symbolic matrices of bandwidth k is* VF*-complete.*

▶ Remark 25. For completeness, we add that for $k = 0$ (symbolic diagonal matrices), the determinant (or, permanent) family is complete for width-1 ABPs, and for $k = 1$, the determinant (or, permanent) family is complete for width-2 ABPs.

The ABP given by Theorem 22 has depth n. On the other hand, converting it to a formula makes the depth $O(k \log(n))$ but the size $n^{O(k)}$. If we are interested in arithmetic circuits, we can eliminate the dependence of k in the exponent of n while keeping the depth logarithmic in n. Compared to Lemma 21, our construction, which is an adaption of the generalized Laplace expansion to low bandwidth matrices, limits the exponential blowup in size *and* the polynomial blowup in depth to the parameter k.

▶ **Theorem 26.** *The determinant (or, permanent) of an $n \times n$ (k, k)-diagonal symbolic matrix can be computed using a syntactic multilinear circuit of size $O(exp(k)n)$ and depth $O(k \log(n))$.*

We refer to the full version [13] for details.

References

1 OEIS Foundation Inc. (2022). Narayana's Cows Sequence, Entry A000930 in the On-Line Encyclopedia of Integer Sequences. `https://oeis.org/A000930`, 1964. Accessed: 2022-04-26.

2 OEIS Foundation Inc. (2022). Padovan Sequence, Entry A000931 in the On-Line Encyclopedia of Integer Sequences. `https://oeis.org/A000931`, 1964. Accessed: 2022-04-26.

3 E. Allender, V. Arvind, and M. Mahajan. Arithmetic complexity, kleene closure, and formal power series. *Theory Comput. Syst.*, 36(4):303–328, 2003.

4 E. Allender and F. Wang. On the power of algebraic branching programs of width two. *Comput. Complex.*, 25(1):217–253, 2016.

5 M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

6 S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984.

7 R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

8 K. Bringmann, C. Ikenmeyer, and J. Zuiddam. On algebraic branching programs of small width. *J. ACM*, 65(5):32:1–32:29, 2018.

9 C. Conley and V. Ovsienko. Lagrangian Configurations and Symplectic Cross-Ratios. *Mathematische Annalen*, pages 1105–1145, 2018.

10 C. Conley and V. Ovsienko. Rotundus: Triangulations, Chebyshev Polynomials, and Pfaffians. *Math Intelligencer*, 40:45–50, 2018.

11 L. Csanky. Fast parallel matrix inversion algorithms. *SIAM Journal on Computing*, 5(4):618–623, 1976.

12 B. Grenet. An Upper Bound for the Permanent versus Determinant Problem. Manuscript, 2011.

13 B. Komarath, A. Pandey, and N. Saurabh. Rabbits approximate, cows compute exactly! Manuscript, 2022. URL: `https://nitinsau.github.io/pubs/cow-sequences.pdf`.

14 X. Lin. On the Recurrence Properties of Narayana's Cows Sequence. *Symmetry*, 13(1), 2021. URL: `https://www.mdpi.com/2073-8994/13/1/149`.

15 M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Electron. Colloquium Comput. Complex.*, 4, 1997.

16 S. Morier-Genoud. Coxeter's Frieze Patterns at the Crossroads of Algebra, Geometry and Combinatorics. *Bulletin of the London Mathematical Society*, 47(6):895–938, 2015.

17 S. Morier-Genoud and V. Ovsienko. Farey Boat: Continued Fractions and Triangulations, Modular Group and Polygon Dissections. *Jahresber. Dtsch. Math. Ver.*, 121:91–136, 2019.

18 Narayana Pandita. Ganita Kaumudi, 1356. India.

19 I. Stewart. Tales of a neglected number. *Scientific American*, 274(6):102–103, 1996. URL: `http://www.jstor.org/stable/24989576`.

20 L. G. Valiant. Completeness Classes in Algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 249–261, 1979.

21 Wikipedia contributors. Narayana Pandita (mathematician). `https://en.wikipedia.org/w/index.php?title=Narayana_Pandita_(mathematician)&oldid=1071293682`, 2022. [Online; accessed 26-April-2022].

22 Wikipedia contributors. Padovan Polynomials. `https://en.wikipedia.org/w/index.php?title=Padovan_polynomials&oldid=1080802324`, 2022. [Online; accessed 27-April-2022].

23 Wikipedia contributors. Padovan Sequence. `https://en.wikipedia.org/w/index.php?title=Padovan_sequence&oldid=1078995920`, 2022. [Online; accessed 27-April-2022].

Finding 3-Swap-Optimal Independent Sets and Dominating Sets Is Hard

Christian Komusiewicz
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Marburg, Germany

Nils Morawietz
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Marburg, Germany

Abstract

For PLS-complete local search problems, there is presumably no polynomial-time algorithm which finds a locally optimal solution, even though determining whether a solution is locally optimal and replacing it by a better one if this is not the case can be done in polynomial time.

We study local search for Weighted Independent Set and Weighted Dominating Set with the 3-swap neighborhood. The 3-swap neighborhood of a vertex set S in G is the set of vertex sets which can be obtained from S by exchanging at most three vertices. We prove the following dichotomy: On the negative side, the problem of finding a 3-swap-optimal independent set or dominating set is PLS-complete. On the positive side, locally optimal independent sets or dominating sets can be found in polynomial time when allowing all 3-swaps except a) the swaps that remove two vertices from the current solution and add one vertex to the solution or b) the swaps that remove one vertex from the current solution and add two vertices to the solution.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Discrete optimization; Theory of computation → Graph algorithms analysis

Keywords and phrases Local Search, Graph problems, 3-swap neighborhood, PLS-completeness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.66

Funding *Nils Morawietz*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project OPERAH, KO 3669/5-1.

1 Introduction

Local search is one of the most successful paradigms for developing algorithms for NP-hard optimization problems. In the most fundamental version of local search, the hill-climbing algorithm, one starts by computing some feasible solution to the problem at hand. This solution is then replaced by a better solution in its local neighborhood, as long as such a better solution exists. The final output is a locally optimal solution. Even though such a locally optimal solution might be arbitrarily bad in comparison to a globally optimal solution, local search approaches turned out to find good solutions in practice [1, 3, 6]. A crucial aspect in this approach is the definition of the local neighborhood of solutions. Intuitively, there is the following trade-off: for more restricted neighborhoods, computing a local optimum should be easier but the quality of the local optimum may be worse than for less restricted neighborhoods. Note that the choice of the neighborhood may affect the computational difficulty of each improvement step as well as the number of necessary improvement steps. Naturally, in applications of hill-climbing, the neighborhoods are chosen in such a way that each improvement step can be performed efficiently.

To study the difficulty of computing locally optimal solutions in such a setting, Johnson et al. [5] introduced the complexity class PLS. This class contains all local search problems for which one can compute some starting solution and search the local neighborhood of a feasible solution S in polynomial time. Thus, the local search problems in PLS are exactly those that have a hill-climbing algorithm where each step only takes polynomial time.

For many unweighted problems, membership in PLS directly implies a polynomial-time algorithm for computing locally optimal solutions, since the maximum objective value is bounded by a polynomial of the input size. The situation is different for weighted problems, where we may need a superpolynomial number of improvement steps to reach a local optimum. To give evidence that for some local search problems in PLS it may be hard to compute locally optimal solutions, Johnson et al. [5] introduced PLS-reductions and showed that there are PLS-complete problems. These problems are as hard as any problem in PLS and none of them is known to be solvable in polynomial time.

We study two famous graph problems, Weighted Independent Set and Weighted Dominating Set through the lens of PLS-completeness. The solutions in both problems are vertex sets and we consider the most fundamental neighborhoods for such solutions: k-swap-neighborhoods, where we may add or remove up to k vertices from a solution.

Previous Work. PLS-completeness has been shown for a number of problems [4, 5, 11, 13, 14]. The initial PLS-complete problem is called Flip, where the input is a Boolean circuit [5]. Another prominent PLS-complete problem is Max Cut with the flip neighborhood, where the neighbors of a partition (A, B) are the partitions that can be obtained by moving one vertex from A to B or vice versa [13]. Max Cut with the flip neighborhood is PLS-complete on graphs with maximum degree 5 [4] and polynomial-time solvable on 3-regular graphs [12].

Johnson et al. [5] already considered the Weighted Independent Set problem and argued that one can show PLS-completeness for a neighborhood that is inspired by the Kernighan-Lin algorithm for Max Cut [7]. Moreover, Johnson et al. [5] specifically called for the study of simpler neighborhoods for the Weighted Independent Set problem. Schäffer and Yannakakis [13] argued that Weighted Independent Set is PLS-complete for a 2-step neighborhood which consists of an addition of a vertex v to the independent set S and a removal of all its neighbors from S in the first step and a (maximal) series of improving vertex additions in the second step. Note that the first step is not necessarily improving. Further studies have shown PLS-completeness for Weighted Independent Dominating Set with a k-swap neighborhood with constant but unspecified k [8] and for Weighted Max-Π-Subgraph with hereditary properties Π and the above-described 2-step-neighborhood [14]. We are not aware of any results for the k-swap neighborhoods with small constant k considered in this work.

From a more applied point of view, local search has been shown to give very good results for Independent Set [1, 2, 3, 6, 9] and weighted Weighted Independent Set [10]. In particular, for Independent Set, Andrade et al. [1] presented fast algorithms for finding improving 5-swaps and showed that the subroutine of finding 5-swap optimal solutions gives very good results when used in an iterative local search framework. Later, it was shown that k-swap-optimal solutions for Independent Set can be efficiently computed for k up to 25 and that, on a set of large sparse real-world networks globally optimal solutions can be found via a simple hill-climbing algorithm for $k \geq 9$ [6].

Our Results. We provide a complexity analysis for Weighted Independent Set with the k-swap neighborhood, denoted Weighted Independent Set/k-swap. Our main result is the PLS-completeness of Weighted Independent Set/3-swap on graphs of constant[1] maximum degree. We first show in Section 3 that Weighted Independent Set/7-swap is PLS-complete on graphs of maximum degree at most 6. Then, in Section 4, we extend the constructed instance of Weighted Independent Set/7-swap to obtain PLS-completeness for Weighted Independent Set/3-swap. Next, we show that by a small modification of the construction, the PLS-completeness for 3-swaps can be transferred to Weighted Dominating Set. Finally, in Section 5, we show that if we allow all 3-swaps except either a) the swaps that remove two vertices and add one or b) the swaps that remove one vertex and add two, we can find locally optimal solutions for Weighted Independent Set and Weighted Dominating Set in polynomial time. We extend this result to a slightly more general neighborhood and a certain type of subset optimization problems. Proofs of statements marked with a ”(*)” are deferred to the full version.

2 Preliminaries

For integers i and j with $i \leq j$, we define $[i, j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$. For two sets A and B, we denote with $A \oplus B := (A \setminus B) \cup (B \setminus A)$ the *symmetric difference* of A and B.

An (undirected) graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq \{\{u, v\} \mid u \in V, v \in V, u \neq v\}$. For vertex sets $S \subseteq V$ and $T \subseteq V$ we denote with $E_G(S, T) := \{\{s, t\} \in E \mid s \in S, t \in T\}$ the edges between S and T. Moreover, we define $G[S] := (S, E_G(S, S))$ as the *subgraph of G induced by S*. For a vertex $v \in V$, we denote with $N_G(v) := \{w \in V \mid \{v, w\} \in E\}$ the *open neighborhood* of v in G and with $N_G[v] := \{v\} \cup N_G(v)$ the *closed neighborhood* of v in G. Analogously, for a vertex set $S \subseteq V$, we define $N_G[S] := \bigcup_{v \in S} N_G[v]$ and $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. If G is clear from the context, we may omit the subscript.

A vertex set $S \subseteq V$ is an *independent set* in G if there is no edge between any pair of vertices of S in G and a *clique* in G if there is an edge between each pair of vertices of S in G. A vertex set $S \subseteq V$ is a *dominating set* in G if for each vertex v of G, at least one vertex of $N[v]$ is contained in S.

The two main problems considered here are now defined as follows.

Weighted Independent Set
Input: A graph $G = (V, E)$ and a vertex-weight function $\omega : V \to \mathbb{N}$.
Output: An independent set in G of maximum total weight.

Weighted Dominating Set
Input: A graph $G = (V, E)$ and a vertex-weight function $\omega : V \to \mathbb{N}$.
Output: A dominating set in G of minimum total weight.

An *optimization problem* L consists of a set $\mathcal{D}_L$ of *instances*, for each instance $I \in \mathcal{D}_L$, a set of *feasible solutions* $\mathcal{S}_L(I)$ for I, an *objective function* val_L which assigns a non-negative rational number to each pair (I, s), and is specified to be either a *minimization* or a *maximization* problem. An optimization problem L is an NP-*optimization problem* if the encoding length of each solution $s \in \mathcal{S}_L(I)$ of I is polynomially bounded by $|I|$, one can determine in polynomial time for each pair (I, s) whether $s \in \mathcal{S}_L(I)$, and the objective function can be evaluated in polynomial time.

[1] Our proof gives a degree bound of 3140.

Let I be an instance and of an optimization problem L and let s and s' be feasible solutions for I. We say that *s is better than s'* if a) L is a maximization problem and $\text{val}_L(I, s) > \text{val}_L(I, s')$ or b) L is a minimization problem and $\text{val}_L(I, s) < \text{val}_L(I, s')$.

▶ **Definition 2.1.** *An* NP*-optimization problem L is a* subset-weight optimization problem *if it consists of*

- *a polynomial-time computable function U that maps each instance I of L to a universe $U(I)$ and each feasible solution of I is a subset of $U(I)$,*
- *a polynomial-time computable function f which checks for an instance I of L and a set $S \subseteq U(I)$ if S is a feasible solution for I,*
- *a polynomial-time computable function g which computes for an instance I of L some feasible solution for I, and*
- *a polynomial-time computable weight function ω which assigns a non-negative rational weight to each pair (I, u), where I is an instance of L and u is an element of $U(I)$*

and one wants to find a feasible solution S for I of either minimal or maximal weight, where the weight of S is defined as $\omega(I, S) := \sum_{u \in S} \omega(I, u)$.

If one wants to find a feasible solution of maximal weight, L is a *subset-weight maximization problem*; otherwise, L is a *subset-weight minimization problem.*

Weighted Independent Set is a subset-weight maximization problem: the feasible solutions are the independent sets of G, these are all subsets of the vertex set V (the universe), one can check in polynomial time if a vertex set S is an independent set, and the total weight of S is defined as the sum of the weights of the vertices of S. Similarly, Weighted Dominating Set is a subset-weight minimization problem.

Let L be a subset-weight optimization problem, let I be an instance of L, and let $S \subseteq U(I)$ be a feasible solution for I. A *k-swap*, for $k \in \mathbb{N}$, is a subset $W \subseteq U(I)$ of size at most k. We say that W is *valid for S in I* if $S \oplus W$ is also a feasible solution for I. We say that two feasible solutions S and S' for I are *k-neighbors in I* if $W := S \oplus S'$ has size at most k. Additionally, we say that S' is an *improving k-neighbor of S in I* and that W is an *improving k-swap* if the total weight of S' is better than the total weight of S. If there is no improving k-neighbor of S in I, S is *k-optimal in I*. Let S be a subset of $U(I)$ and let k, k_{in}, and k_{out} be natural numbers such that $k_{\text{in}} + k_{\text{out}} \leq k$. A k-swap W is a *$(k_{\text{in}}, k_{\text{out}})$-swap for S in G*, if $|W \setminus S| \leq k_{\text{in}}$ and if $|W \cap S| \leq k_{\text{out}}$. Similar to k-swaps, we also define the notions of *valid $(k_{\text{in}}, k_{\text{out}})$-swaps*, *improving $(k_{\text{in}}, k_{\text{out}})$-swaps*, *$(k_{\text{in}}, k_{\text{out}})$-neighbors*, *improving $(k_{\text{in}}, k_{\text{out}})$-neighbors*, and *$(k_{\text{in}}, k_{\text{out}})$-optimal.*

A *partition of a graph $G = (V, E)$* is a pair (A, B), where $A \cup B = V$ and $A \cap B = \emptyset$. The *cut* of a partition (A, B) is the edge set $E_G(A, B)$, that is, the set of edges having one endpoint in A and one endpoint in B. Let $\omega : E \to \mathbb{N}$ be an edge-weight function. A *flip of a vertex $v \in V$ in a partition* (A, B) is the partition (A', B'), where $A' := A \oplus \{v\}$ and $B' := B \oplus \{v\}$. Moreover, we say that *(A', B') is improving over* (A, B) if the total weight of cut $E_G(A', B')$ is larger than the total weight of the cut $E_G(A, B)$, that is, if $\omega(E_G(A', B')) > \omega(E_G(A, B))$. Furthermore, we say that a partition (A, B) is *flip-optimal* if there is no vertex $v \in V$ such that the flip (A', B') of v in (A, B) is improving over (A, B).

In the corresponding minimization problem one wants to find a cut of maximal weight.

Max Cut
Input: A graph $G = (V, E)$ and an edge-weight function $\omega : E \to \mathbb{N}$.
Output: A partition (A, B) of G such that $E_G(A, B)$ has maximum total weight.

A *local search problem* $(L, \mathcal{N})$ consists of

- an optimization problem L and
- a *neighborhood structure* $\mathcal{N}$ *for* L that maps for each instance I of L, each valid solution S of I to a set $\mathcal{N}(I, S) \subseteq \mathcal{S}_L(I)$ of valid solutions for I, the *neighbors of* S *with respect to* $\mathcal{N}$.

The goal of $(L, \mathcal{N})$ is to find for a given instance I of L a *locally optimal solution* S *with respect to* $\mathcal{N}$, that is, a feasible solution for I such that no solution in $\mathcal{N}(I, S)$ is better than S. We may write a local search problem $(L, \mathcal{N})$ as $L/\mathcal{N}$. An example for a local search problem is Weighted Independent Set/k-swap, where the neighbors of an independent set S are the valid k-swap neighbors of S.

A local search problem $(L, \mathcal{N})$ *is in the complexity class* PLS, if

- there is an algorithm which computes in polynomial time some feasible solution S for a given instance I of L and
- there is an algorithm which in polynomial time determines whether a given solution S is locally optimal with respect to $\mathcal{N}$ for an instance I of L and, if this is not the case, outputs a better neighbor for S.

Note that for each constant k, Weighted Independent Set/k-swap is contained in PLS.

Let $(L_1, \mathcal{N}_1)$ and $(L_2, \mathcal{N}_2)$ be local search problems. We say that $(L_1, \mathcal{N}_1)$ *is* PLS-*reducible to* $(L_2, \mathcal{N}_2)$ if for each instance I_1 of L_1, one can compute in polynomial time an instance I_2 of L_2 and a *solution mapper* f, that is, a polynomial-time computable function f that maps each solution S_2 of I_2 to a solution $f(S_2)$ of I_1 such that if S_2 is locally optimal for I_2 with respect to $\mathcal{N}_2$, then $f(S_2)$ is locally optimal for I_1 with respect to $\mathcal{N}_1$. A local search problem $(L, \mathcal{N})$ is PLS-*hard* if for each local search problem $(L', \mathcal{N}')$ of PLS, there is a PLS-reduction from $(L', \mathcal{N}')$ to $(L, \mathcal{N})$. Due to the transitivity of PLS-reductions, this can be done by proving a PLS-reduction from any PLS-hard local search problem $(L', \mathcal{N}')$. Moreover, $(L, \mathcal{N})$ is PLS-*complete* if $(L, \mathcal{N})$ is contained in PLS and PLS-hard. An example for a PLS-complete local search problem is Max Cut/flip, where the neighbors of a partition (A, B) are the improving partitions of (A, B) one can obtain by flipping some vertex of G [13]. This is the case even on graphs of degree at most 5 [4].

3 Hardness of Finding 7-optimal Independent Sets

To obtain the PLS-completeness of Weighted Independent Set/3-swap on graphs of constant maximum degree, we first show PLS-completeness for Weighted Independent Set/7-swap and use the obtained graph as starting point in a subsequent reduction to Weighted Independent Set/3-swap.

▶ **Theorem 3.1.** *Weighted Independent Set/7-swap is* PLS*-complete on graphs of maximum degree 6.*

As mentioned above, Weighted Independent Set/k-swap is contained in PLS for each constant value of k. Hence, we only have to show that Weighted Independent Set/7-swap is PLS-hard.

Construction. We present a PLS-reduction from Max Cut/flip to Weighted Independent Set/7-swap. Let $I = (G = (V, E), \omega)$ be an instance of Max Cut/flip where G has a maximum degree of five. For these instances Max Cut/flip is known to be PLS-complete [4]. We describe how to obtain in polynomial time an instance $I' = (G' = (V', E'), \omega')$ of Weighted Independent Set/7-swap and a polynomial-time computable solution-mapper f for I and I' such that if an independent set S is 7-optimal in I', then $f(S)$ is flip-optimal in I.

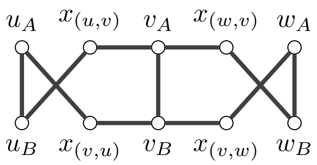


Figure 1 An example for the vertices and edges added to G' for a vertex $v \in V$ with two neighbors u and w in G in the reduction from Max Cut/flip to Weighted Independent Set/7-swap.

We start with an empty graph G' and add, for each vertex $v \in V$, two new adjacent vertices v_A and v_B to G'. Next, for each edge $\{u, v\} \in E$, we add two new vertices $x_{(u,v)}$ and $x_{(v,u)}$ to G' and make $x_{(u,v)}$ adjacent to u_B and v_A and $x_{(v,u)}$ adjacent to u_A and v_B. This completes the construction of G'. Figure 1 shows an example for the vertices and edges added to G' for a vertex $v \in V$ with two neighbors u and w in G. Note that G' has a maximum degree of six. In the following, let $V_A := \{v_A \mid v \in V\}$ and $V_B := \{v_B \mid v \in V\}$.

Next, we define the weight function $\omega' : V' \to \mathbb{N}$. Let $Z := \sum_{e \in E} \omega(e)$ denote the total weight of all edges. We set for each vertex $v \in V$, $\omega'(v_A) := \omega'(v_B) := 16 \cdot Z$ and for each edge $\{u, v\} \in E$, we set $\omega'(x_{(u,v)}) := \omega'(x_{(v,u)}) := 8 \cdot \omega(\{u, v\})$. In principle, the factors of 8 can be omitted but they will come in handy later when we present the PLS-reduction to Weighted Independent Set/3-swap.

This completes the construction of I'. It remains to define the solution-mapper f. For an independent set S, we define $f(S)$ to be the partition (A, B) of G where $A := \{v \in V \mid v_A \in S\}$ and $B := V \setminus A$. Recall that for vertex $v \in V$, the vertices v_A and v_B are adjacent in G'.

Correctness. To show the correctness of the reduction, we first analyze the structure of 7-optimal independent sets in G'. To this end, we define a notion of *nice* independent sets in G' and show that all 7-optimal independent sets in G' are nice. Recall that if some vertex v of V is contained in A for $f(S) = (A, B)$, then v_B is not contained in S.

▶ **Definition 3.2.** *Let S be an independent set in G' and let $A := \{v \in V \mid v_A \in S\}$ and let $B := \{v \in V \mid v_B \in S\}$. We call S* nice *if (A, B) is a partition of G and for each edge $\{u, v\} \in E$ with $(u, v) \in A \times B$, $x_{(u,v)} \in S$.*

▶ **Lemma 3.3 (*).** *If an independent set S in G is not nice, then S is not 7-optimal in G'.*

Hence, we only have to consider nice independent sets in G' when considering 7-optimal independent sets in G'. Now, to prove Theorem 3.1 it remains to show the following.

▶ **Lemma 3.4.** *Let S be a nice independent set in G'. If $f(S)$ is not flip-optimal in G, then S is not 7-optimal in G'.*

Proof. Let $(A, B) := f(S)$. By definition of f and the fact that S is nice, $A = \{v \in V \mid v_A \in S\}$ and $B = \{v \in V \mid v_B \in S\}$. Suppose that (A, B) is not flip-optimal in G. Then, there is some vertex $v \in V$ where the total weight of the edges that are incident with v and that are in the cut $E_G(A, B)$ is less than the total weight of the edges that are incident with v and that are not in the cut $E_G(A, B)$. That is, either a) $v \in A$ and $\omega(E_G(\{v\}, A)) > \omega(E_G(\{v\}, B))$ or b) $v \in B$ and $\omega(E_G(\{v\}, B)) > \omega(E_G(\{v\}, A))$.

Without loss of generality we may assume that $v \in A$ and $\omega(E_G(\{v\}, A)) > \omega(E_G(\{v\}, B))$. Let $X_A := N_G(v) \cap A$ denote the neighbors of v in A and let $X_B := N_G(v) \cap B$ denote the neighbors of v in B. Since S is nice, we know that $x_{(v,u)} \in S$ for each $u \in X_B$. Moreover,

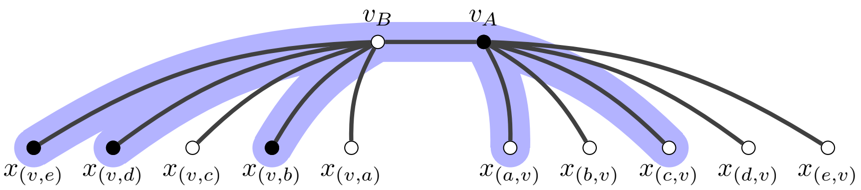


Figure 2 An example of an improving 7-swap W for a nice independent set S in G' that simulates the flip of a vertex $v \in V$ from A to B in G, where $N_G(v) = \{a, b, c, d, e\}$ and $N_G(v) \cap B = \{b, d, e\}$. The black vertices are the vertices of S and all vertices of W are highlighted by the blue shape.

for each $w \in X_A$, $x_{(v,w)} \notin S$ since $w_A \in S$ and $x_{(w,v)} \notin S$ since $v_A \in S$. We show that the swap $W := \{v_A, v_B\} \cup \{x_{(v,u)} \mid u \in X_B\} \cup \{x_{(w,v)} \mid w \in X_A\}$ is a valid improving 7-swap for S in G'. An example of the swap W is illustrated in Figure 2. First of all, note that W has size at most 7 since G has a maximum degree of 5 and thus $|X_A \cup X_B| \leq 5$. Moreover, by the fact that $\omega'(x_{(y,z)}) := \omega'(x_{(z,y)}) := 8 \cdot \omega(\{y, z\})$ for each edge $\{y, z\} \in E$,

$$\begin{aligned}\omega'(\{x_{(w,v)} \mid w \in X_A\}) &= 8 \cdot \omega(\{\{v, w\} \mid w \in X_A\}) \\ &> 8 \cdot \omega(\{\{v, u\} \mid u \in X_B\}) = \omega'(\{x_{(v,u)} \mid u \in X_B\}).\end{aligned}$$

Hence, W is improving, since $\omega'(v_A) = \omega'(v_B)$. It remains to show that W is valid. Since W removes all adjacent vertices of v_B from S, $S \oplus W$ does not contain any neighbor of v_B. Moreover, since $v_A \in W$ and for each $w \in X_A$, $w_A \in S \oplus W$ and thus $w_B \notin S \oplus W$, no vertex $x_{(w,v)}$ is adjacent to any vertex in $S \oplus W$. As a consequence, W is valid and thus S is not 7-optimal. ◀

4 Hardness of Finding 3-optimal Independent Sets

The main result of this section is the following.

▶ **Theorem 4.1.** *Weighted Independent Set/3-swap is* PLS*-complete on graphs of constant maximum degree.*

Construction. To show this, we extend the graph G' by additional gadgets to simulate 7-swaps by a sequence of 3-swaps. We describe how to obtain in polynomial time an instance $I'' = (G'' = (V'', E''), \omega'')$ of Weighted Independent Set/3-swap and a polynomial-time computable solution-mapper f for I and I'' such that if an independent set S is 3-optimal in I'', then $f(S)$ is flip-optimal in I. As described above, we extend the graph G' and the weight function ω' of the instance $I' = (G' = (V', E'), \omega')$ of Weighted Independent Set/7-swap described above. As above, let $V_A := \{v_A \mid v \in V\}$ and let $V_B := \{v_B \mid v \in V\}$. Moreover, we set for each $v \in V$, $X_v := \{x_{(v,w)}, x_{(w,v)} \mid w \in N(v)\}$, that is, X_v is the set of vertices in G' that were introduced for the incident edges of v in G. We write $N_G(v)$ and $N_G[v]$ when considering the neighborhood of a vertex v of V in G and $N(v)$ or $N[v]$ when considering the neighborhood of a vertex v of V'' in G''.

Initially, we add edges such that for each vertex $v \in V$, $u \in N_G(v)$, and $w \in N_G(V)$, the vertices $x_{(u,v)}$ and $x_{(v,w)}$ are adjacent in G''. Note that this includes edges between $x_{(u,v)}$ and $x_{(v,u)}$ in G'' for each edge $\{u, v\} \in E$. The idea is that in an independent set S in G'', for each vertex $v \in V$, there is no vertex $x_{(u,v)} \in S$ if S already contains a vertex $x_{(v,w)}$. Hence, for each vertex v of G, at most one of v_A and v_B has neighbors in $X_v \cap S$.

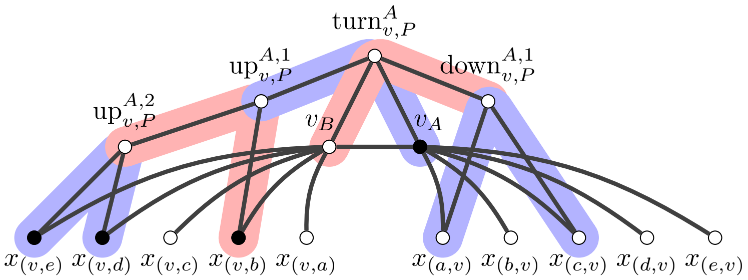


Figure 3 A sequence $(W_1, W_2, W_3, W_4, W_5)$ of improving 3-swaps (from left to right highlighted alternating by either a blue or a red shape) for a nice independent set S in G'' that simulates the flip of a vertex $v \in V$ from A to B in G, where $N_G(v) = \{a, b, c, d, e\}$ and $P = N_G(v) \cap B = \{b, d, e\}$ with $P(1) = b$, $P(2) = d$, and $P(3) = e$. The black vertices are the vertices of S. Note that not all edges of this subgraph are shown but only the important ones for the sequence of improving 3-swaps.

Next, we add gadgets to allow us to simulate 7-swaps in G' by a sequence of 3-swaps in G''. To this end, we first compute for each vertex $v \in V$ the collection $\mathcal{P}_v$ of subsets $P \subseteq N_G(v)$ fulfilling $\omega(E_G(\{v\}, P)) < \omega(E_G(\{v\}, N_G(v) \setminus P))$. Intuitively, if in a partition (A, B) of G, flipping a vertex v from A to B is improving, then the set $N_G(v) \cap B$ is contained in $\mathcal{P}_v$ and vice versa. Note that $\emptyset \in \mathcal{P}_v$ and $N_G(v) \notin \mathcal{P}_v$ for each $v \in V$. Next, we add, for each $P \in \mathcal{P}_v$ with $P \neq \emptyset$, a set of $|N_G(v)| - 1$ new vertices to G''. Let $Q := N_G(v) \setminus P$. Now, fix an ordering on both P and Q and let $P(i)$ denote the ith element of P and let $Q(j)$ denote the jth element of Q where $i \in [1, |P|]$ and $j \in [1, |Q|]$.

These vertices are of three types: up, turn, and down. To simulate a flip of v from A to B, we have to remove the neighbors of v_B from S, add v_B to S and add the vertices representing edges of $(A \cap N_G(v)) \times \{v\}$ to S. Intuitively, the up-vertices allow us – with a sequence of improving 3-swaps – to remove all neighbors of v_B from S except v_A and the up-vertex of highest level. Afterwards, the turn-vertex allows us – with two improving 3-swaps – to remove v_A from S and add v_B and the down-vertex of highest level to S. Finally, the down-vertices allow us – with a sequence of improving 3-swaps – to add the vertices representing edges of $(A \cap N_G(v)) \times \{v\}$ to S such that at the end, none of these auxiliary vertices remains in S. In total, this allows us to simulate improving 7-swaps in I' and thus improving flips in I by a sequence of improving 3-swaps in I''. An example for a sequence of improving 3-swaps in G'' to simulate a flip in G is illustrated in Figure 3. This figure shows only the edges that are important for the sequence of improving 3-swaps and not all edges of this subgraph.

First, we add for each $i \in [1, |P|-1]$, a new vertex $\mathrm{up}^{A,i}_{v,P}$ to G such that the neighborhood of $\mathrm{up}^{A,i}_{v,P}$ is exactly X_v minus the vertices $\{x_{(v,P(j))} \mid j < i\}$. Hence, $\mathrm{up}^{A,1}_{v,P}$ is adjacent to all vertices of X_v. Furthermore, we add an edge between $\mathrm{up}^{A,i}_{v,P}$ and each vertex of $\{w_A \mid w \in P\} \cup \{w_B \mid w \in Q\}$ and an edge between $\mathrm{up}^{A,i}_{v,P}$ and v_B. Moreover, we set

$$\omega''(\mathrm{up}^{A,i}_{v,P}) := 4 - i + 8 \cdot \sum_{j=i}^{|P|} \omega(\{v, P(j)\}) = 4 - i + \sum_{j=i}^{|P|} \omega''(x_{(v,P(j))}).$$

Intuitively, one can obtain an improving 3-neighbor for S in G as follows:

1. if $w_A \notin S$ for any neighbor $w \in N_G(v)$ in P and $u_B \notin S$ for any neighbor $u \in N_G(v)$ in P, then we remove $x_{(v,P(|P|-1))}$ and $x_{(v,P(|P|))}$ and add $\mathrm{up}^{A,|P|-1}_{v,P}$, and
2. if $\mathrm{up}^{A,j}_{v,P} \in S$, where $j \in [2, |P| - 1]$, then we remove $\mathrm{up}^{A,j}_{v,P}$ and $x_{(v,P(j))}$ and add $\mathrm{up}^{A,j-1}_{v,P}$.

Hence, during the simulation of an improving flip of a vertex v in a partition (A, B) from A to B in G, we can replace all vertices of X_v by the vertex $\text{up}_{v,P}^{A,1}$ with a sequence of improving 3-swaps, where $P = N_G(v) \cap B$.

Second, we add a vertex $\text{turn}_{v,P}^{A}$ to G'' which is adjacent to all vertices of X_v, $\{v_A, v_B\}$, and to the vertices of $\{w_A \mid w \in P\} \cup \{w_B \mid w \in Q\}$. Recall that $\omega''(v_A) = \omega''(v_B) = 16 \cdot Z$ where $Z = \sum_{e \in E} \omega(e)$. We set

$$\omega''(\text{turn}_{v,P}^{A}) := 4 + \omega''(v_A) + 8 \cdot \sum_{w \in P} \omega(\{v, w\}) = 4 + 16 \cdot Z + \sum_{w \in P} \omega''(x_{(v,w)}).$$

Intuitively, one can obtain an improving 3-neighbor for S in G if $\text{up}_{v,P}^{A,1}$ is contained in S, by removing both $\text{up}_{v,P}^{A,1}$ and v_A from S and adding $\text{turn}_{v,P}^{A}$ to S.

Third, we add, similar to $\text{up}_{v,P}^{A,i}$, for each $i \in [1, |Q| - 1]$ a new vertex $\text{down}_{v,P}^{A,i}$ to G such that the neighborhood of $\text{down}_{v,P}^{A,i}$ is exactly $X_v \setminus \{x_{(Q(j),v)} \mid j < i\}$. Hence, $\text{down}_{v,P}^{A,1}$ is adjacent to all vertices of X_v. Furthermore, we add an edge between $\text{down}_{v,P}^{A,i}$ and each vertex of $\{w_A \mid w \in P\} \cup \{w_B \mid w \in Q\}$ and we also an edge between $\text{down}_{v,P}^{A,i}$ and v_A. Moreover, we set

$$\omega''(\text{down}_{v,P}^{A,i}) := i - 4 + 8 \cdot \sum_{j=i}^{|Q|} \omega(\{v, Q(j)\}) = i - 4 + \sum_{j=i}^{|Q|} \omega''(x_{(v,Q(j))}).$$

Note that $\omega''(\text{down}_{v,P}^{A,i}) > 0$ since $Q \neq \emptyset$ and the images of ω are positive numbers. Intuitively, one can obtain an improving 3-neighbor for S in G as follows:

1. if $\text{turn}_{v,P}^{A} \in S$, then we remove $\text{turn}_{v,P}^{A}$ and add $\text{down}_{v,P}^{A,1}$ and v_B,
2. if $\text{down}_{v,P}^{A,j} \in S$ for some $j < |Q| - 1$, then we remove $\text{down}_{v,P}^{A,j}$ and add $\text{down}_{v,P}^{A,j+1}$ and $x_{(Q(j),v)}$, and
3. if $\text{down}_{v,P}^{A,|Q|-1} \in S$, then we remove $\text{down}_{v,P}^{A,|Q|-1}$ and add $x_{(Q(|Q|-1),v)}$ and $x_{(Q(|Q|),v)}$.

With the current graph, it is possible to simulate a flip of a vertex v from A to B. To also simulate a flip of vertex v from B to A, we add symmetric vertices to G'', that is, for each vertex $\text{up}_{v,P}^{A,j}$, we add a vertex $\text{up}_{v,P}^{B,j}$, for each vertex $\text{turn}_{v,P}^{A}$, we add a vertex $\text{turn}_{v,P}^{B}$, and for each vertex $\text{down}_{v,P}^{A,j}$, we add a vertex $\text{down}_{v,P}^{B,j}$.

The formal definition of these symmetric vertices is deferred to the full version.

Note that we did not add any vertices for $P = \emptyset \in \mathcal{P}_v$ to the graph G''. In the correctness proof, we show that a single improving 3-swap for S is sufficient to simulate the flip of a vertex v of G if no edge incident with v is currently in the cut.

For each vertex $v \in V$, let V_v denote the set of the additional vertices associated with v:

$$V_v := \bigcup_{P \in \mathcal{P}_v, P \neq \emptyset, C \in \{A,B\}} (\{\text{up}_{v,P}^{C,i} \mid i < |P|\} \cup \{\text{turn}_{v,P}^{C}\} \cup \{\text{down}_{v,P}^{C,i} \mid i < |N_G(v) \setminus P|\}).$$

To complete the construction, for each $v \in V$, we make the set $\bigcup_{w \in N_G[v]} V_w$ a clique in G''.

▶ **Lemma 4.2** (*). *The graph G'' has maximum degree at most 3140.*

It remains to define the solution-mapper f. Analogously to the solution-mapper of the presented PLS-reduction for WEIGHTED INDEPENDENT SET/7-swap, for an independent set S, we define $f(S)$ to be the partition (A, B) of G where $A := \{v \in V \mid v_A \in S\}$ and $B := V \setminus A$.

Correctness. To show the correctness of the PLS-reduction, we first analyze the structure of 3-optimal independent sets in G''. To this end, we show in the auxiliary Lemmas 4.3 – 4.5 that we can assume that each 3-optimal independent set in G'' contains for each $v \in V$ either the vertex v_A or the vertex v_B. Recall that for vertex $v \in V$, the vertices v_A and v_B are adjacent in G''. As a consequence, this then implies that for $f(S) = (A, B)$, B is exactly the set $\{v \in V \mid v_B \in S\}$. Finally, in Lemma 4.6, we then show that such an independent set S is not 3-optimal in I'' if $f(S)$ is not flip-optimal in G.

▶ **Lemma 4.3.** *Let S be an independent set in G''. If S contains a vertex $\mathrm{up}_{v,P}^{C,i}$ for $C \in \{A, B\}$, then S is not 3-optimal.*

Proof. First, we show the statement for $i = 1$. By construction, the closed neighborhood $N[\mathrm{turn}_{v,P}^{C}]$ is exactly $N[\mathrm{up}_{v,P}^{C,1}] \cup \{v_C\}$ and $\omega''(\mathrm{turn}_{v,P}^{C}) = 1 + \omega''(\mathrm{up}_{v,P}^{C,1}) + \omega''(v_C)$. Hence, S is not 3-optimal in G'' since $S' := (S \cup \{\mathrm{turn}_{v,P}^{C}\}) \setminus \{\mathrm{up}_{v,P}^{C,1}, v_C\}$ is an improving 3-neighbor of S in G''. This holds even if v_C is not in S.

Second, we show the statement for $i > 1$. Let $r := x_{(v,P(i-1))}$ if $C = A$ and $r := x_{(P(i-1),v)}$ if $C = B$. Note that by construction, the closed neighborhood $N[\mathrm{up}_{v,P}^{C,i-1}]$ is exactly $N[\mathrm{up}_{v,P}^{C,i}] \cup \{r\}$ and $\omega''(\mathrm{up}_{v,P}^{C,i-1}) = 1 + \omega''(\mathrm{up}_{v,P}^{C,i}) + \omega''(r)$. Hence, if an independent set S contains $\mathrm{up}_{v,P}^{C,i}$ with $i > 1$, then S is not 3-optimal in G'' since $(S \cup \{\mathrm{up}_{v,P}^{C,i-1}\}) \setminus \{\mathrm{up}_{v,P}^{C,i}, r\}$ is an improving 3-neighbor of S in G''. This holds even if r is not in S. ◀

▶ **Lemma 4.4 (*).** *Let S be an independent set in G''. If there is a vertex $v \in V$ such that S avoids v_A, v_B, and $T_v := \{\mathrm{turn}_{v,P}^{A}, \mathrm{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$, then S is not 3-optimal.*

Hence, we can assume that each 3-optimal independent set S in G'' contains for each vertex $v \in V$ either v_A, v_B, or exactly one vertex of $T_v := \{\mathrm{turn}_{v,P}^{A}, \mathrm{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$, and no vertex of $\{\mathrm{up}_{v,P}^{A,i}, \mathrm{up}_{v,P}^{B,i} \mid P \in \mathcal{P}_v, P \neq \emptyset, i < |P|\}$. In the following, we call an independent set S of G'' *nice* if $S \subseteq V' = V_A \cup V_B \cup \bigcup_{v \in V} X_v$ and if S is a nice independent set for the instance I' of Weighted Independent Set/7-swap. That is, if for $A := \{v \in V \mid v_A \in S\}$ and $B := \{v \in V \mid v_B \in S\}$, (A, B) is a partition of G and for each edge $\{u, v\} \in E$ with $(u, v) \in A \times B$, the vertex $x_{(u,v)}$ is contained in S. Next, we show similar to Lemma 3.4, that an independent set S in G'' is not 3-optimal if S is not nice.

▶ **Lemma 4.5.** *Let S be an independent set in G''. If S is not nice, then S is not 3-optimal.*

Proof. Let S be an independent set in G'' which is not nice. Due to Lemma 4.3, we know that S is not 3-optimal if some vertex $\mathrm{up}_{v,P}^{C,i}$ for $C \in A, B$ is contained in S. Hence, in the following, we assume that none of these vertices is contained in S. Moreover, due to Lemma 4.4, we also know that S is not 3-optimal if there is some vertex $v \in V$ such that S does not contain v_A, v_B, and no vertex of $\{\mathrm{turn}_{v,P}^{A}, \mathrm{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$. Hence, in the following we further assume that S contains one of these vertices for some $v \in V$.

In a first step, we show that if for some $v \in V$, S contains some vertex of $\{\mathrm{turn}_{v,P}^{A}, \mathrm{turn}_{v,P}^{B} \mid P \in \mathcal{P}_v, P \neq \emptyset\}$, then S is not 3-optimal. Assume without loss of generality that $\mathrm{turn}_{v,P}^{A} \in S$ and let $Q := N_G(v) \setminus P$. Recall that $\bigcup_{w \in N_G[v]} V_w$ is a clique in G'' which implies that for each $w \in N_G(v)$, S contains no vertex of $\{\mathrm{turn}_{w,P'}^{A}, \mathrm{turn}_{w,P'}^{B} \mid P' \in \mathcal{P}_w, P' \neq \emptyset\} \subseteq V_w$. Hence, due to Lemma 4.4, to for each $w \in N_G(v)$, S contains either w_A or w_B. Moreover, $\mathrm{turn}_{v,P}^{A}$ is adjacent to all vertices of X_v, all vertices of $\{w_A \mid w \in P\}$, and all vertices of $\{w_B \mid w \in Q\}$. As a consequence, for each $w \in P$, $w_B \in S$, and $w_A \notin S$, and for each $u \in Q$, $u_A \in S$ and $u_B \notin S$. Furthermore, S contains no other neighbor of v_A or v_B. Recall that $P \neq N_G(v)$ which implies $Q = \emptyset$. In the following, it suffices to distinguish between $|Q| = 1$ and $|Q| > 1$.

First, suppose that $|Q| = 1$ and let w denote the unique vertex of Q. We show that $S' := (S \cup \{v_B, x_{(w,v)}\}) \setminus \{\text{turn}^A_{v,P}\}$ is an improving 3-neighbor of S in G''. By construction, v_B and $x_{(w,v)}$ are non-adjacent in G'' and

$$\begin{aligned}\omega''(v_B) + \omega''(x_{(w,v)}) &= 16 \cdot Z + 8 \cdot \omega(\{v,w\}) = 16 \cdot Z + 8 \cdot \sum_{u \in Q} \omega(\{v,u\}) \\ &> 16 \cdot Z + 4 + 8 \cdot \sum_{u \in P} \omega(\{v,u\}) = \omega''(\text{turn}^A_{v,P})\end{aligned}$$

since the images of ω are positive numbers and $P \in \mathcal{P}_v$ which implies $\sum_{u \in Q} \omega(\{v,u\}) > \sum_{u \in P} \omega(\{v,u\})$. Hence, it remains to show that S is an independent set. Since by definition, $N[v_B] \subseteq N[\text{turn}^A_{v,P}]$, we only have to show that there is no other neighbor of $x_{(w,v)}$ in S besides $\text{turn}^A_{v,P}$. By construction, the neighborhood of $x_{(w,v)}$ in G'' is a subset of $\{v_A, w_B\} \cup V_v \cup V_w \cup X_v \cup X_w$. Now, $\text{turn}^A_{v,P}$ is adjacent to all vertices of this set except for some vertices of X_w. Hence, we only have to consider the neighbors of $x_{(w,v)}$ in X_w. By construction, these are the vertices $x_{(u,w)}$ where $u \in N_G(w)$. Since w_A is contained in S and each of these vertices $x_{(u,w)}$ is adjacent to w_A in G'', $x_{(w,v)}$ has no neighbor in S besides $\text{turn}^A_{v,P}$. As a consequence, S' is an improving 3-neighbor of S.

Second, suppose that $|Q| > 1$. Then, the vertex $\text{down}^{A,1}_{v,P}$ exists. We show that $S' := (S \cup \{v_B, \text{down}^{A,1}_{v,P}\}) \setminus \{\text{turn}^A_{v,P}\}$ is an improving 3-neighbor of S in G''. By construction, v_B and $\text{down}^{A,1}_{v,P}$ are non-adjacent in G'' and

$$\begin{aligned}\omega''(v_B) + \omega''(\text{down}^{A,1}_{v,P}) &= 16 \cdot Z - 3 + 8 \cdot \sum_{w \in Q} \omega(\{v,w\}) \\ &> 16 \cdot Z + 4 + 8 \cdot \sum_{u \in P} \omega(\{v,u\}) = \omega''(\text{turn}^A_{v,P})\end{aligned}$$

since the images of ω are positive numbers and $P \in \mathcal{P}_v$ which implies $\sum_{u \in Q} \omega(\{v,u\}) > \sum_{u \in P} \omega(\{v,u\})$. Hence, it remains to show that S is an independent set. By definition, $N[v_B] \subseteq N[\text{turn}^A_{v,P}]$ and $N[\text{down}^{A,1}_{v,P}] \subseteq N[\text{turn}^A_{v,P}]$, which implies that $\text{turn}^A_{v,P}$ is the unique neighbor of both v_B and $\text{turn}^A_{v,P}$ in S. As a consequence, S' is an independent set and thus an improving 3-neighbor of S. Hence, in the following, we can also assume that for each $v \in V$, S contains either v_A or v_B.

Next, we show that if for some $v \in V$, there is some vertex $r \in V_v$ contained in S, then S is not 3-optimal. Note that we already showed that this is the case if r is $\text{up}^{A,i}_{v,P}$, $\text{up}^{B,i}_{v,P}$, $\text{turn}^A_{v,P}$, or $\text{turn}^B_{v,P}$. Hence, it remains to show the claim for r being $\text{down}^{A,i}_{v,P}$ or $\text{down}^{B,i}_{v,P}$. Assume without loss of generality that there is a nonempty set $P \in \mathcal{P}_v$ and some $i < |Q|$ with $Q := N_G(v) \setminus P$ such that $\text{down}^{A,i}_{v,P}$ is contained in S. The proof of the fact that in this case S is not 3-optimal is deferred to the full version.

Summarizing, we can assume in the following that $S \subseteq V' = V_A \cup V_B \cup \bigcup_{v \in V} X_v$ and that (A, B) is a partition of G, where $A := \{v \in V \mid v_A \in S\}$ and $B := \{v \in V \mid v_B \in S\}$. Note that for each edge $\{v, w\} \in E$ with $(v, w) \notin A \times B$, the vertex $x_{(v,w)}$ is not contained in S since $v_B \in S$ or $w_A \in S$, and both these vertices are adjacent to $x_{(v,w)}$. Hence, it remains to show that if there is some edge $\{v, w\} \in E$ with $(v, w) \in A \times B$ such that $x_{(v,w)}$ is not contained in S, then S is not 3-optimal in G''. To this end, we show that $S \cup \{x_{(v,w)}\}$ is an independent set in G''. By construction, $x_{(v,w)}$ is adjacent to v_B, w_A, some vertices of $V_v \cup V_w$, and the vertices $\{x_{(u,v)} \mid u \in N_G(v)\} \cup \{x_{(w,u)} \mid u \in N_G(w)\}$. Recall that S contains no vertex of $V_v \cup V_w$. Moreover, since $(v, w) \in A \times B$, v_A and w_B are contained in S which implies that v_B and w_A are not contained in S. Furthermore, since all vertices of $\{x_{(u,v)} \mid u \in N_G(v)\}$ are adjacent to v_A, all vertices of $\{x_{(w,u)} \mid u \in N_G(w)\}$ are adjacent to w_B, and v_A and w_B are contained in S, $S \cup \{x_{(v,w)}\}$ is an independent set in G''.

We conclude that if S is not nice in G'', then S is not 3-optimal. ◀

Now the following implies the correctness of the PLS-reduction.

▶ **Lemma 4.6 (*).** *Let S be a nice independent set in G'' and let $A := \{v \in V \mid v_A \in S\}$ and $B := \{v \in V \mid v_B \in S\}$. If (A, B) is not flip-optimal in G, then S is not 3-optimal.*

Next, we obtain similar results for Weighted Dominating Set.

▶ **Theorem 4.7 (*).** *Let $k \geq 3$. There is a PLS-reduction from Weighted Independent Set/k-swap to Weighted Dominating Set/k-swap where the maximum degree of the output graph is at most two times the maximum degree of the input graph.*

Hence, we obtain the following due to Theorem 4.7, Theorem 3.1, and Theorem 4.1.

▶ **Corollary 4.8.** *Weighted Dominating Set/7-swap is PLS-complete on graphs of maximum degree at most 12 and Weighted Dominating Set/3-swap is PLS-complete on graphs of constant maximum degree.*

5 Finding Locally Optimal Solutions for Restricted 3-Swaps

We now show that we can find locally optimal solutions in polynomial time for Weighted Independent Set and Weighted Dominating Set if we restrict the allowed 3-swaps as follows: we either only allow swaps that add at most one vertex to the current solution or we allow only swaps that remove at most one vertex from the solution. These are exactly the $(1, 2)$-swaps and $(2, 1)$-swaps, respectively, since a 3-swap that removes three vertices from the current solution or that adds three vertices to the solution is either not improving, or the solution is not 1-optimal.

Recall that a $(k^{\text{in}}, k^{\text{out}})$-swap W for a set S is a k-swap with $k^{\text{in}} + k^{\text{out}} \leq k$ such that $|W \setminus S| \leq k^{\text{in}}$ and $|W \cap S| \leq k^{\text{out}}$.

First, we show that we can compute in $\mathcal{O}(n \cdot \log(n) + m)$ time a $(1, 2)$-optimal independent set S. More precisely, we show that S is even $(1, k)$-optimal for every $k \in \mathbb{N}$.

▶ **Theorem 5.1 (*).** *One can compute in $\mathcal{O}(n \cdot \log(n) + m)$ time an independent set which is $(1, k)$-optimal for every $k \in \mathbb{N}$.*

Second, we show that we can also compute in $\mathcal{O}(n \cdot \log(n) + m)$ time a $(2, 1)$-optimal dominating set S. More precisely, we show that S is even $(k, 1)$-optimal for every $k \in \mathbb{N}$.

▶ **Theorem 5.2 (*).** *One can compute in $\mathcal{O}(n \cdot \log(n) + m)$ time a dominating set which is $(k, 1)$-optimal for every $k \in \mathbb{N}$.*

▶ **Theorem 5.3.** *Let L be a subset-weight maximization problem, let I be an instance of L, and let $k \in \mathcal{O}(1)$. One can compute in polynomial time a $(k, 1)$-optimal solution S for I.*

Proof. Recall that since L is a subset-weight maximization problem, L consists of functions U, f, g, and ω, where for each instance I of L, $U(I)$ is the universe of I, $f(I, S)$ checks if S is a solution for I, $g(I)$ computes some feasible solution for I, and $\omega(I, u)$ assigns a weight to each $u \in U(I)$. Moreover, the functions U, f, g, and ω are polynomial-time computable.

Let I be an instance of L and let $k \geq 0$. Note that we can compute some feasible solution $S_0 := g(I)$ in polynomial time. Since $U(I)$ can be computed in polynomial time, $U(I)$ has polynomial size. Let $n := |U(I)|$. Since f and ω can be computed in polynomial time, we can check for a given solution S in $n^{\mathcal{O}(k)}$ time if there is an improving $(k+1)$-swap W such that W is a $(k,1)$-swap for S by considering all subsets of size at most $k+1$ of $U(I)$. Note that this is polynomial time since k is a constant. Hence, we can determine whether a solution S is $(k,1)$-optimal and, if this is not the case, replace S by a $(k,1)$-neighbor, both in polynomial time. Let S_x be a $(k,1)$-optimal feasible solution in I which can be obtained by a sequence $(S_0, S_1, \dots, S_x)$ of consecutive improving $(k,1)$-neighbors in I starting from $g(I) = S_0$. We show that $x \in \mathcal{O}(n^3)$, which implies that a $(k,1)$-optimal feasible solution for I can be computed in polynomial time. Let $\ell \in [1,x]$. To this end, we first show that S_ℓ is never smaller than $S_{\ell-1}$. Assume towards a contradiction that $|S_\ell| < |S_{\ell-1}|$. Then there is some $u \in S_{\ell-1}$ such that $S_\ell = S_{\ell-1} \setminus \{u\}$ since S_ℓ is a $(k,1)$-neighbor of $S_{\ell-1}$ in I. Since the weight of S_ℓ is defined as the sum of weights of the elements of S_ℓ and each element $u \in U(I)$ has a positive weight $\omega(I,u)$, the total weight of S_ℓ is not larger than the total weight of $S_{\ell-1}$, a contradiction. As a consequence, there are at most $y \leq |U(I)|$ distinct indices $\ell_1, \dots, \ell_y$ such that S_{ℓ_i} is larger than S_{ℓ_i-1}. To prove that $x \in \mathcal{O}(n^3)$, we now show that for each $\ell \in [1, x-n^2]$, there is some $z \in [\ell, \ell+n^2]$ where S_z is larger than $S_{\ell-1}$. In other words, after at most n^2 improving swaps that do not increase the size of the solution, the next improving swap increases the size of the solution.

Let $\ell \in [1, x - U(I)^2]$, let $\sigma = (u_1, \dots, u_n)$ be a fixed increasing order of the elements of $U(I)$ according to their weight, and let for each $j \in [\ell, \ell+n^2]$, $q_j := \sum_{u_i \in S_j} i$ denote the sum of the indices of S_j in σ. Note that if S_j and S_{j-1} have same size, $S_j = (S_{j-1} \cup \{u_2\}) \setminus \{u_1\}$ for two elements $u_1 \in U(I)$ and $u_2 \in U(I)$ where the weight of u_2 is larger than the weight of u_1. Hence, q_j is larger than q_{j-1} if S_j and S_{j-1} have same size. Since the value q_j can never exceed n^2, there is some $z \in [\ell, \ell+n^2]$ where the size of S_z is larger than the size of $S_{\ell-1}$. We conclude that $x \in \mathcal{O}(n^3)$ which implies that we can compute in polynomial time a feasible solution S of I such that S is $(k,1)$-optimal in I. ◀

▶ **Corollary 5.4 (*).** *Let L be a subset-weight minimization problem, let I be an instance of L and let $k \in \mathcal{O}(1)$. One can compute in polynomial time a $(1,k)$-optimal solution S for I.*

By the fact that for a maximization problem there is no improving swap that only removes elements from the solution and for a minimization problem there is no improving swap that only adds elements to the solution, Theorem 5.3 and Corollary 5.4 imply that one can find for each subset weight optimization problem L, a 2-optimal solution in polynomial time.

Since Weighted Independent Set is a subset-weight maximization problem and since Weighted Dominating Set is a subset-weight minimization problem, we conclude the following.

▶ **Corollary 5.5.** *Let k be a constant. One can compute in polynomial time a $(k,1)$-optimal independent set and one can compute in polynomial time a $(1,k)$-optimal dominating set.*

Consequently, if we allow only $(1,2)$-swaps or only $(2,1)$-swaps, we can find locally optimal solutions for Weighted Independent Set and Weighted Dominating Set in polynomial time. In contrast, if we allow $(1,2)$-swaps and $(2,1)$-swaps, then we allow all 3-swaps and both problems become PLS-complete even on graphs of constant maximum degree.

6 Conclusion

From a theoretical point of view, the most important open topic is to determine the precise degree bounds that separate the polynomial-time solvable and PLS-complete cases for Weighted Independent Set and Weighted Dominating Set for k-swaps with small constant k-values. From a practical point of view, our findings motivate, for both problems, the use of gap-variants of local search where we are searching for solutions that improve the objective value by at least some threshold d, the hope being that this decreases the number of necessary iterations while preserving solution quality.

References

1 Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

2 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

3 Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA '16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2016.

4 Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP '11)*, volume 6755 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2011. doi:10.1007/978-3-642-22006-7_15.

5 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

6 Maximilian Katzmann and Christian Komusiewicz. Systematic exploration of larger local search neighborhoods for the minimum vertex cover problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, (AAAI '17)*, pages 846–852. AAAI Press, 2017.

7 Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

8 Hartmut Klauck. On the hardness of global and local approximation. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT '96)*, volume 1097 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 1996.

9 Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017.

10 Ruizhi Li, Shuli Hu, Shaowei Cai, Jian Gao, Yiyuan Wang, and Minghao Yin. NuMWVC: A novel local search for minimum weighted vertex cover problem. *Journal of the Operational Research Society*, 71(9):1498–1509, 2020.

11 Wil Michiels, Emile H. L. Aarts, and Jan H. M. Korst. *Theoretical aspects of local search*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2007.

12 Svatopluk Poljak. Integer linear programs and local search for max-cut. *SIAM Journal on Computing*, 24(4):822–839, 1995.

13 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.

14 Shinichi Shimozono. Finding optimal subgraphs by local search. *Theoretical Computer Science*, 172(1-2):265–271, 1997.

SAT-Based Circuit Local Improvement

Alexander S. Kulikov
Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences, Russia
St. Petersburg State University, Russia

Danila Pechenev
St. Petersburg State University, Russia

Nikita Slezkin
St. Petersburg State University, Russia

Abstract

Finding exact circuit size is notoriously hard. Whereas modern computers and algorithmic techniques allow to find a circuit of size seven in the blink of an eye, it may take more than a week to search for a circuit of size thirteen. One of the reasons of this behavior is that the search space is enormous: the number of circuits of size s is $s^{\Theta(s)}$, the number of Boolean functions on n variables is 2^{2^n}.

In this paper, we explore the following natural heuristic idea for decreasing the size of a given circuit: go through all its subcircuits of moderate size and check whether any of them can be improved by reducing to SAT. This may be viewed as a local search approach: we search for a smaller circuit in a ball around a given circuit. Through this approach, we prove new upper bounds on the circuit size of various symmetric functions. We also demonstrate that some upper bounds that were proved by hand decades ago, can nowadays be found automatically in a few seconds.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases circuits, algorithms, complexity theory, SAT, SAT solvers, heuristics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.67

Related Version *Full Version*: `https://arxiv.org/abs/2102.12579` [15]

Supplementary Material *Software (Source Code)*:
`https://github.com/alexanderskulikov/circuit_improvement`

Funding *Alexander S. Kulikov*: Supported by Russian Science Foundation (18-71-10042).

Acknowledgements We are grateful to anonymous reviewers for many useful suggestions.

1 Boolean Circuits

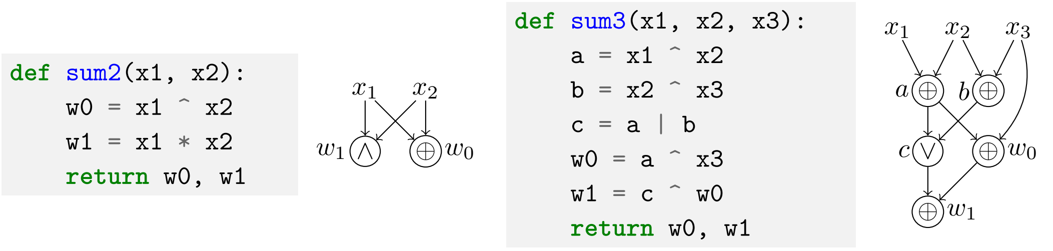
A Boolean *straight line program* of size r for input variables $(x_1, \ldots, x_n)$ is a sequence of r instructions where each instruction $g \leftarrow h \circ k$ applies a binary Boolean operation $\circ$ to two operands h, k each of which is either an input bit or the result of a previous instruction. If m instructions are designated as outputs, the straight line program computes a function $\{0,1\}^n \to \{0,1\}^m$ in a natural way. We denote the set of all such functions by $B_{n,m}$ and we let $B_n = B_{n,1}$. For a Boolean function $f\colon \{0,1\}^n \to \{0,1\}^m$, by size$(f)$ we denote the minimum size of a straight line program computing f. A Boolean *circuit* shows a graph of a program: for every instruction $g \leftarrow h \circ k$, there is a node g with two directed incoming edges from nodes h and k.

Figure 1 gives an example for the $\mathrm{SUM}_n\colon \{0,1\}^n \to \{0,1\}^l$ function that computes the binary representation of the sum of n bits:

$$\mathrm{SUM}_n(x_1, \ldots, x_n) = (w_0, w_1, \ldots, w_{l-1})\colon \sum_{i=1}^{n} x_i = \sum_{i=0}^{l-1} 2^i w_i, \quad \text{where } l = \lceil \log_2(n+1) \rceil .$$

This function transforms n bits of weight 0 into l bits of weights $(0, 1, \ldots, l-1)$.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 67; pp. 67:1–67:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



```
def sum2(x1, x2):
    w0 = x1 ^ x2
    w1 = x1 * x2
    return w0, w1
```

```
def sum3(x1, x2, x3):
    a = x1 ^ x2
    b = x2 ^ x3
    c = a | b
    w0 = a ^ x3
    w1 = c ^ w0
    return w0, w1
```

Figure 1 Optimal size straight line programs and circuits for SUM$_2$ and SUM$_3$. These two circuits are known as *half adder* and *full adder*.

The straight line programs are given in `Python` so that it is particularly easy to verify their correctness. For example, the program for SUM$_3$ can be verified with just three lines of code:

```
from itertools import product

for x1, x2, x3 in product(range(2), repeat=3):
    w0, w1 = sum3(x1, x2, x3)
    assert x1 + x2 + x3 == w0 + 2 * w1
```

Determining size(f) requires proving lower bounds: to show that size$(f) > \alpha$, one needs to prove that *every* circuit of size at most α does not compute f. Known lower bounds are far from being satisfactory: the strongest known lower bound for a function family in NP is $(3+1/86)n - o(n)$ [7]. Here, by a *function family* we mean an infinite sequence of functions $\{f_n\}_{n=1}^{\infty}$ where $f_n \in B_n$. Even proving lower bounds for specific functions (rather than function families) is difficult. Brute force approaches become impractical quickly: $|B_n| = 2^{2^n}$, hence already for $n = 6$, one cannot just enumerate all functions from B_n; also, the number of circuits of size s is $s^{\Theta(s)}$, hence checking all circuits of size s takes reasonable time for small values of s only. Knuth [11] found the exact circuit size of all functions from B_4 and B_5.

Finding the exact value of size(f) for $f \in B_6$ is already a difficult computational task for modern computers and techniques. One approach is to translate a statement "there exists a circuit of size s computing f" to a Boolean formula and to pass it to a SAT solver. Then, if the formula is satisfiable, one decodes a circuit from its satisfying assignment; otherwise, one gets a (computer generated) proof of a lower bound size$(f) > s$. This circuit synthesis approach was proposed by Kojevnikov et al. [13] and, since then, has been used in various circuit synthesis programs (`abc` [1], `mockturtle` [24], `sat-chains` [10]).

State-of-the-art SAT solvers are surprisingly efficient and allow to handle various practically important problems (with millions of variables) and even help to resolve open problems in mathematics [2]. Still, already for small values of n and s the problem of finding a circuit of size s for a function from B_n is difficult for SAT solvers. We demonstrate the limits of this approach on *counting* functions: $\mathrm{MOD}_n^{m,r}(x_1, \ldots, x_n) = [x_1 + \cdots + x_n \equiv r \bmod m]$ (here, $[\cdot]$ is the Iverson bracket: $[S]$ is equal to 1 if S is true and is equal to 0 otherwise). Using SAT solvers, Knuth [12, solution to exercise 480] found size$(\mathrm{MOD}_n^{3,r})$ for all $3 \le n \le 5$ and all $0 \le r \le 2$. Generalizing the found values, he made the following conjecture:

$$\text{size}(\mathrm{MOD}_n^{3,r}) = 3n - 5 - [(n+r) \equiv 0 \bmod 3] \text{ for all } n \ge 3 \text{ and } r. \tag{1}$$

He was also able to prove (using SAT solvers) that size$(\mathrm{MOD}_6^{3,0}) = 12$ and wrote: "The case $n = 6$ and $r \neq 0$, which lies tantalizingly close to the limits of today's solvers, is still unknown."

Knuth also describes various symmetry breaking heuristics and shows which of them give a significant speedup. Haaswijk et al. [8] show another way of speeding up the SAT-based approach for circuit synthesis: first, generate all possible circuit topologies, then, for each topology check using SAT solvers whether one can assign Boolean operation to the gates so that the resulting circuit computes a given function.

To summarize, our current abilities for checking whether there exists a Boolean circuit of size s are roughly the following: for $s \leq 6$, this can be done in a few seconds; for $7 \leq s \leq 12$, this can (sometimes) be done in a few days; for $s \geq 13$, this is out of reach.

1.1 New Results

In this paper, we explore the limits of the following natural idea: given a circuit, try to improve its size by improving (using SAT solvers, for example) the size of its subcircuit of size seven. This is a kind of a local search approach: we have no possibility to go through the whole space of all circuits, but we can at least search in a neighborhood of a given circuit. This allows us to work with circuits consisting of many gates.

As the results of experiments, we show several circuits for which the approach described above leads to improved upper bounds.

- We support Knuth's conjecture (1) for $\mathrm{MOD}_n^{3,r}$ by proving the matching upper bound:

  $$\mathrm{size}(\mathrm{MOD}_n^{3,r}) \leq 3n - 5 - [(n+r) \equiv 0 \bmod 3] \text{ for all } n \geq 3 \text{ and } r.$$

  This improves slightly the previously known upper bound $\mathrm{size}(\mathrm{MOD}_n^{3,r}) \leq 3n - 4$ by Demenkov et al. [4]. To prove Knuth's conjecture, one also needs to prove a lower bound on $\mathrm{size}(\mathrm{MOD}_n^{3,r})$. The currently strongest known lower bound for $\mathrm{size}(\mathrm{MOD}_n^{3,r})$ is $2.5n - O(1)$ due to Stockmeyer [25].
- We present improvements for $\mathrm{size}(\mathrm{SUM}_n)$ for various small n and show that some of these circuits and their parts can be used as building blocks to design efficient circuits for other functions in semiautomatic fashion. In particular, we show that a part of an optimal circuit for SUM_5 can be used to build optimal circuits of size $2.5n$ for $\mathrm{MOD}_n^{4,r}$ [25] and best known circuits of size $4.5n+o(n)$ for SUM_n [4]. In turn, an efficient circuit for SUM_5 can be found in a few seconds if one starts from a standard circuit for SUM_5 composed out of two full adders and one half adder.
- We design new circuits for the threshold function defined as follows:

  $$\mathrm{THR}_n^k(x_1, \dots, x_n) = [x_1 + \dots + x_n \geq k] \,.$$

  The best known upper bounds for THR are the following:

  $$\mathrm{size}(\mathrm{THR}_n^k) \leq kn + o(n) \text{ for } 2 \leq k \leq 4 \text{ [5] (see also [26, 6.2, Theorem 2.3])},$$
  $$\mathrm{size}(\mathrm{THR}_n^k) \leq 4.5n + o(n) \text{ for } 5 \leq k \text{ [4]}.$$

  We get the following improvement: $\mathrm{size}(\mathrm{THR}_n^k) \leq (4.5 - 2^{2-\lceil \log_2 k \rceil})n + o(n)$ for $4 \leq k = O(1)$. In particular, $\mathrm{size}(\mathrm{THR}_n^4) \leq 3.5n + o(n)$ and $\mathrm{size}(\mathrm{THR}_n^k) \leq 4n + o(n)$ for $5 \leq k \leq 8$.

The improved upper bounds are obtained in a semiautomatic fashion: first, we automatically improve a given small circuit with a fixed number of inputs using SAT solvers; then, we generalize it to every input size. For some function families, the second step is already known (for example, given a small circuit for SUM_5, it is not difficult to use it as a building block to design an efficient circuit for SUM_n for every n; see Section 3.1), though in general this still needs to be done manually.

1.2 Related work

The approach we use in this paper follows the SAT-based local improvement method (SLIM): to improve an existing discrete structure one goes through all its substructures of size accessible to a SAT solver. SLIM has been applied successfully to the following structures: branchwidth [16], treewidth [6], treedepth [20], Bayesian network structure learning [21], decision tree learning [22].

2 Program: Feature Overview and Evaluation

The program is implemented in `Python`. We give a high-level overview of its main features below. All the code shown below can be found in the file `tutorial.py` at [3]. One may run it after installing a few `Python` modules. Alternatively, one may run the Jupyter notebook `tutorial.ipynb` in the cloud (without installing anything) by pressing the badge "Colab" at the repository page [3].

2.1 Manipulating Circuits

This is done through the `Circuit` class. One can load and save circuits as well as print and draw them. A nicely looking layout of a circuit is produced by the `pygraphviz` module [19]. The program also contains some built-in circuits that can be used as building blocks. The following sample code constructs a circuit for SUM$_5$ out of two full adders and one half adder. This construction is shown in Figure 2(a). Then, the circuit is verified via the `check_sum_circuit` method. Finally, the circuit is drawn. As a result, one gets a picture similar to the one in Figure 2(b).

```
circuit = Circuit(input_labels=['x1', 'x2', 'x3', 'x4', 'x5'])
x1, x2, x3, x4, x5 = circuit.input_labels
a0, a1 = add_sum3(circuit, [x1, x2, x3])
b0, b1 = add_sum3(circuit, [a0, x4, x5])
w1, w2 = add_sum2(circuit, [a1, b1])
circuit.outputs = [b0, w1, w2]
check_sum_circuit(circuit)
circuit.draw('sum5')
```

2.2 Finding Efficient Circuits

The class `CircuitFinder` allows to check whether there exists a circuit of the required size for a given Boolean function. For example, one may discover the full adder as follows. (The function `sum_n` returns the list of $\lceil \log_2(n+1) \rceil$ bits of the binary representation of the sum of n bits.)

```
def sum_n(x):
    return [(sum(x) >> i) & 1 for i in range(ceil(log2(len(x) + 1)))]


circuit_finder = CircuitFinder(dimension=3, number_of_gates=5,
                               function=sum_n)
circuit = circuit_finder.solve_cnf_formula()
circuit.draw('sum3')
```

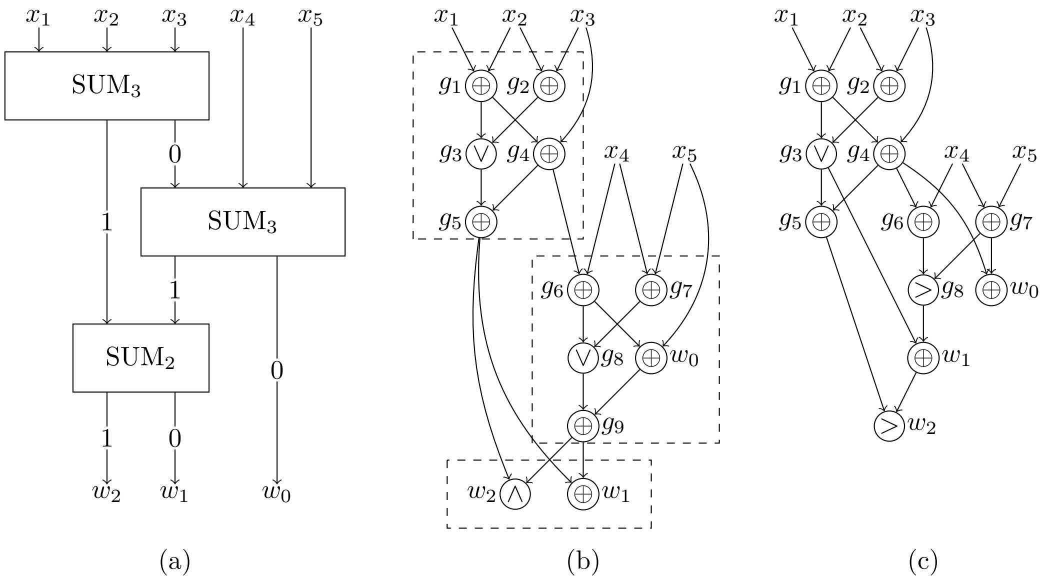

x_1 x_2 x_3 x_4 x_5
SUM$_3$
0
1
SUM$_3$
1
0
SUM$_2$
1
0
w_2 w_1 w_0
(a)
g_1 g_2 g_3 g_4 g_5 g_6 g_7 g_8 g_9
w_0 w_1 w_2
(b)
g_1 g_2 g_3 g_4 g_5 g_6 g_7 g_8
w_0 w_1 w_2
(c)

Figure 2 (a) A schematic circuit for SUM$_5$ composed out of two full adders and one half adder. (b) The corresponding circuit of size 12. (c) An improved circuit of size 11.

This is done by encoding the task as a CNF formula and invoking a SAT solver (via the `pysat` module [9]). The reduction to SAT is described in [13]. Basically, one translates a statement "there exists a circuit of size s comuting a given function $f\colon \{0,1\}^n \to \{0,1\}^m$" to CNF-SAT. To do this, one introduces many auxiliary variables: for example, for every $x \in \{0,1\}^n$ and every $1 \le i \le r$, one uses a variable that is responsible for the value of the i-th gate on the input x.

As mentioned in the introduction, the limits of applicability of this approach (for finding a circuit of size s) are roughly the following: for $s \le 6$, it usually works in less than a minute; for $7 \le s \le 12$, it may already take up to several hours or days; for $s \ge 13$, it becomes almost impractical. The running time may vary a lot for inputs of the same length. In particular, it usually takes much longer to prove that the required circuit does not exist (by proving that the corresponding formula is unsatisfiable). Table 1 reports the running time of this approach on several datasets.

2.3 Improving Circuits

The method `improve_circuit` goes through all subcircuits of a given size of a given circuit and checks whether any of them can be replaced by a smaller subcircuit (computing the same function) via `find_circuit`. For example, applying this method to the circuit from Figure 2(b) gives the circuit from Figure 2(c) in a few seconds.

```
circuit = Circuit(input_labels=[f'x{i}' for i in range(1, 6)], gates={})
circuit.outputs = add_sum5_suboptimal(circuit, circuit.input_labels)
improved_circuit = improve_circuit(circuit, subcircuit_size=5,
                                   connected=True)
print(improved_circuit)
improved_circuit.draw('sum5')
```

Table 1 The running time of `CircuitFinder` on various Boolean functions.

function	circuit size	status	time (sec.)
SUM_5	12	SAT	141.4
SUM_5	11	SAT	337.8
$\mathrm{MOD}_4^{3,0}$	7	SAT	0.2
$\mathrm{MOD}_4^{3,0}$	6	UNSAT	1178.8
$\mathrm{MOD}_4^{3,1}$	7	SAT	0.2
$\mathrm{MOD}_4^{3,1}$	6	UNSAT	1756.5
$\mathrm{MOD}_4^{3,2}$	6	SAT	0.2
$\mathrm{MOD}_4^{3,2}$	5	UNSAT	12.6
$\mathrm{MOD}_5^{3,0}$	10	SAT	90.1
$\mathrm{MOD}_5^{3,1}$	9	SAT	50.1
$\mathrm{MOD}_5^{3,2}$	10	SAT	74.3

Table 2 shows the time taken by `improve_circuit` to improve some of the known circuits for SUM, MOD^3, and THR^4. For SUM, we start from known circuits of size about $5n$ (composed out of full adders and half adders). For MOD^3, we start from circuits of size $3n-4$ presented by Demenkov et al. [4]. For THR^4, we start from circuits of size about $5n$ (we start by computing SUM_n and then compare the resulting $\log n$-bit integer to 4).

Table 2 The running time of `improve_circuit` on various Boolean functions.

function	circuit size	time (sec.)
SUM_5	$12 \to 11$	6.7
SUM_7	$20 \to 19$	5.8
$\mathrm{MOD}_6^{3,0}$	$15 \to 14$	17.0
$\mathrm{MOD}_6^{3,1}$	$15 \to 14$	17.2
$\mathrm{MOD}_6^{3,2}$	$14 \to 13$	16.7
$\mathrm{MOD}_7^{3,0}$	$17 \to 16$	31.3
$\mathrm{MOD}_7^{3,1}$	$17 \to 16$	33.6
$\mathrm{MOD}_7^{3,2}$	$16 \to 15$	30.5
THR_5^4	$23 \to 10$	38.6
THR_6^4	$28 \to 14$	42.1
THR_7^4	$31 \to 17$	43.8
THR_8^4	$40 \to 22$	55.1

3 New Circuits

In this section, we present new circuits for symmetric functions found with the help of the program. A function $f(x_1, \ldots, x_n)$ is called *symmetric* if its value depends on $\sum_{i=1}^n x_i$ only. They are among the most basic Boolean functions:

- to specify an arbitrary Boolean function from B_n, one needs to write down its truth table of length 2^n; symmetric functions allow for more compact representation: it is enough to specify $n+1$ bits (for each of $n+1$ values of $\sum_{i=1}^n x_i$);

- circuit complexity of almost all functions of n variables is exponential ($\Theta(2^n/n)$), whereas any symmetric function can be computed by a linear size circuit ($O(n)$).

Despite simplicity of symmetric functions, we still do not know how optimal circuits look like for most of them. Below, we present new circuits for some of these functions.

3.1 Sum Function

The SUM function is a fundamental symmetric function: for any symmetric $f \in B_n$, $\text{size}(f) \leq \text{size}(\text{SUM}_n) + o(n)$. The reason for this is that any function from B_n can be computed by a circuit of size $O(2^n/n)$ by the results of Muller [18] and Lupanov [17]. This allows to compute any symmetric $f(x_1, \dots, x_n) \in B_n$ as follows: first, compute $\text{SUM}_n(x_1, \dots, x_n)$ using $\text{size}(\text{SUM}_n)$ gates; then, compute the resulting bit using at most $O(2^{\log n}/\log n) = o(n)$ gates. For the same reason, any lower bound $\text{size}(f) \geq \alpha$ for a symmetric function $f \in B_n$ implies a lower bound $\text{size}(\text{SUM}_n) \geq \alpha - o(n)$. Currently, we know the following bounds for SUM_n: $2.5n - O(1) \leq \text{size}(\text{SUM}_n) \leq 4.5n + o(n)$. The lower bound is due to Stockmeyer [25], the upper bound is due to Demenkov et al. [4].

A circuit for SUM_n can be constructed from circuits for SUM_k for some small k. For example, using full and half adders as building blocks, one can compute SUM_n (for any n) by a circuit of size $5n$ as follows. Start from n bits $(x_1, \dots, x_n)$ of weight 0. While there are three bits of the same weight k, replace them by two bits of weights k and $k+1$ using a full adder. This way, one gets at most two bits of each weight $0, 1, \dots, l-1$ ($l = \lceil \log_2(n+1) \rceil$) in at most $5(n-l)$ gates (as each full adder reduces the number of bits). To leave exactly one bit of each weight, it suffices to use at most l half or full adders ($o(n)$ gates). Let us denote the size of the resulting circuit by $s(n)$. The first row of Table 3 shows the values of $s(n)$ for some $n \leq 15$ (see (28) in [11]).

Table 3 The first row gives the size $s(n)$ of a circuit for SUM_n composed out of half and full adders, the second row shows known upper bounds for $\text{size}(\text{SUM}_n)$ (all of them were known before our work, see (28) in [11]).

n	2	3	4	5	6	7	8	9	10	15
$s(n)$	2	5	9	12	17	20	26	29	34	55
$\text{size}(\text{SUM}_n)$	2	5	9	11	≤ 16	≤ 19	≤ 25	≤ 27	≤ 32	≤ 53

In a similar fashion, one can get an upper bound (see Theorem 1 in [14])

$$\text{size}(\text{SUM}_n) \leq \frac{\text{size}(\text{SUM}_k)}{k - \lceil \log_2(k+1) \rceil} \cdot n + o(n)\,. \tag{2}$$

This motivates the search for efficient circuits for SUM_k for small values of k. The bottom row of Table 3 gives upper bounds that we were able to find using the program. The table shows that the first value where $s(n)$ is not optimal is $n = 5$. The best upper bound for SUM_n given by (2) is $4.75n + o(n)$ for $k = 7$. The upper bound for $k = 15$ is $53n/11 + o(n)$ which is worse than the previous upper bound. But if it turned out that $\text{size}(\text{SUM}_{15}) \leq 52$, it would give a better upper bound.

The found circuits for SUM_n for $n \leq 15$ do not allow to improve the strongest known upper bound $\text{size}(\text{SUM}_n) \leq 4.5n + o(n)$ due to Demenkov et al. [4]. Below, we present several interesting observations on the found circuits.

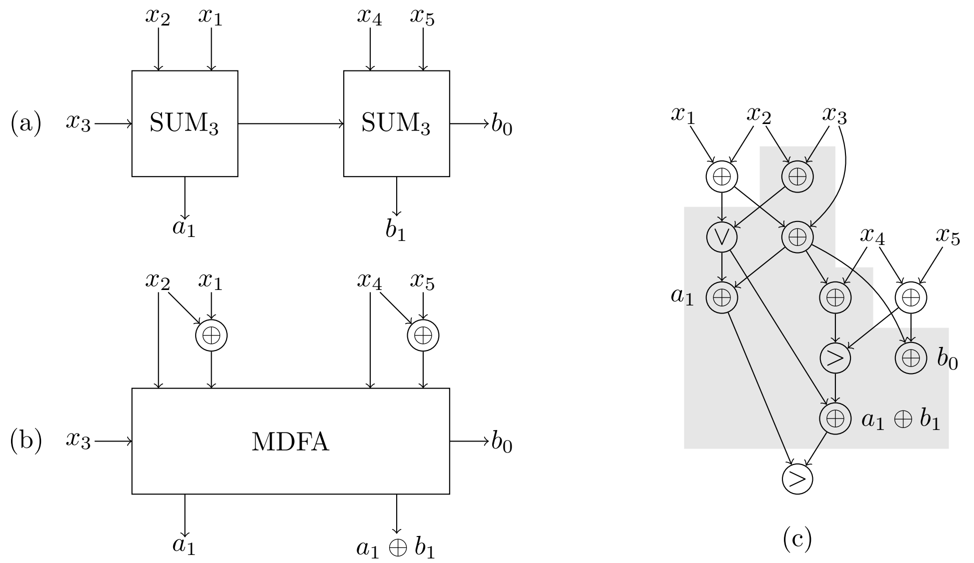


Figure 3 (a) Two consecutive SUM$_3$ blocks. (b) The MDFA block. (c) The highlighted part of the optimal circuit for SUM$_5$ computes MDFA.

3.1.1 Best Known Upper Bound for the SUM Function

The optimal circuit of size 11 for SUM$_5$ shown in Figure 2(c) can be used to get an upper bound $4.5n + o(n)$ for size(SUM$_n$) (though not through (2) directly). To do this, consider two consecutive SUM$_3$ circuits shown in Figure 3(a). They compute a function DFA$(x_1, x_2, x_3, x_4, x_5) = (b_0, b_1, a_1)$ (for double full adder) such that, for every $x_1, \ldots, x_5 \in \{0, 1\}$, $x_1 + \cdots + x_5 = b_0 + 2(b_1 + a_1)$. Figure 3(a) shows that size(DFA) ≤ 10. One can construct a similar block, called MDFA (for modified double full adder), such that

$$\text{MDFA}(x_1 \oplus x_2, x_2, x_3, x_4, x_4 \oplus x_5) = (b_0, a_1, a_1 \oplus b_1)\,,$$

see Figure 3(b).

The fact that MDFA uses the encoding $(p, p \oplus q)$ for pairs of bits (p, q), allows to use it recursively to compute SUM$_n$. As the original construction is presented in [4], below we give a sketch only.

1. Compute $x_2 \oplus x_3, x_4 \oplus x_5, \ldots, x_{n-1} \oplus x_n$ ($n/2$ gates).
2. Apply at most $n/2$ MDFA blocks (no more than $4n$ gates).
3. The last MDFA block outputs two bits: a and $a \oplus b$. Instead of them, one needs to compute $a \oplus b$ and $a \wedge b$. To achieve this, it suffices to apply $x > y = (x \wedge \overline{y})$ operation: $a \wedge b = a > (a \oplus b)$.

Figure 4 shows an example for $n = 17$.

The MDFA block was constructed by Demenkov et al. [4] in a semiautomatic manner. And it turns out that MDFA is just a subcircuit of the optimal circuit for SUM$_5$! See Figure 3(c).

3.1.2 Best Known Circuits for SUM with New Structure

For many upper bounds from the bottom row of Table 3, we found circuits with the following interesting structure: the first thing the circuit computes is $x_1 \oplus x_2 \oplus \cdots \oplus x_n$; moreover the variables $x_2, \ldots, x_n$ are used for this only. This is best illustrated by an example, see Figure 5.

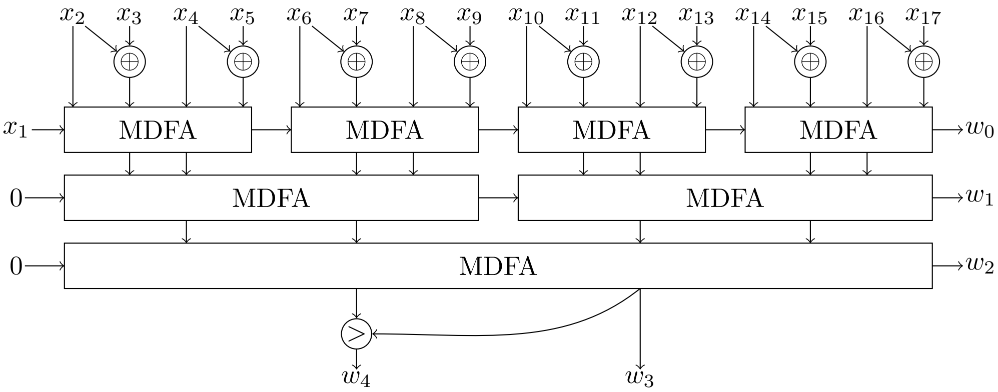

x_1
x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
MDFA
0
w_0
w_1
w_2
w_3
w_4

Figure 4 A circuit computing SUM_{17} composed out of MDFA blocks.

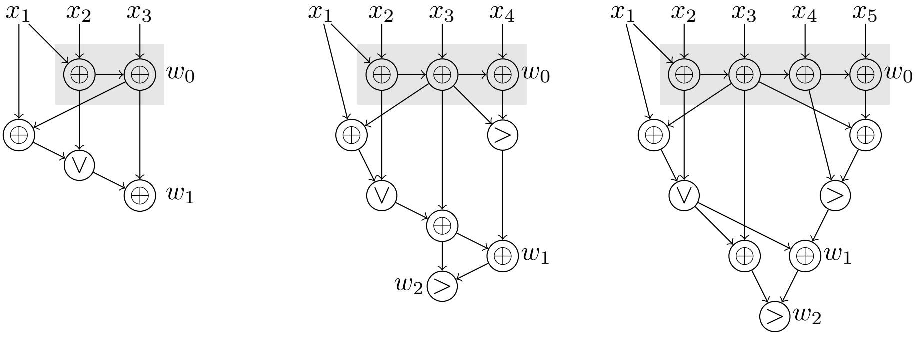


Figure 5 Optimal circuits computing SUM_n for $n = 3, 4, 5$ with a specific structure: every input, except for x_1, has out-degree one.

These circuits can be found using the following code. It demonstrates two new useful features: fixing gates and forbidding wires between some pairs of gates.

```
def sum_n(x):
    return [(sum(x) >> i) & 1 for i in range(ceil(log2(len(x) + 1)))]


for n, size in ((3, 5), (4, 9), (5, 11)):
    circuit_finder = CircuitFinder(dimension=n, number_of_gates=size,
                                   function=sum_n)
    circuit_finder.fix_gate(n, 0, 1, '0110')
    for k in range(n - 2):
        circuit_finder.fix_gate(n + k + 1, k + 2, n + k, '0110')
    for i in range(1, n):
        for j in range(n, n + size):
            if i + n - 1 != j:
                circuit_finder.forbid_wire(i, j)
    circuit = circuit_finder.solve_cnf_formula(verbose=0)
    circuit.draw(f'sum{n}')
```

3.1.3 Optimal Circuits for Counting Modulo 4

The optimal circuit for SUM_5 can be used to construct an optimal circuit of size $2.5n+O(1)$ for $\mathrm{MOD}_n^{4,r}$ due to Stockmeyer [25] (recall that $\mathrm{MOD}_n^{4,r}(x_1,\dots,x_n) = [x_1+\dots+x_n \equiv r \bmod 4]$). To do this, note that there is a subcircuit (of the circuit in Figure 2(c)) of size 9 that computes the two least significant bits (w_0, w_1) of $x_1+\dots+x_5$ (one removes the gates g_5, w_2). To compute $x_1 + \dots + x_n \bmod 4$, one first applies $\frac{n}{4}$ such blocks and then computes the parity of the resulting bits of weight 1 (every block takes four fresh inputs as well as one bit of weight 0 from the previous block). The total size is $9 \cdot \frac{n}{4} + \frac{n}{4} = 2.5n$. Thus, block that Stockmeyer constructed by hand in 1977 to compute MOD_n^4 nowadays can be found automatically in a few seconds.

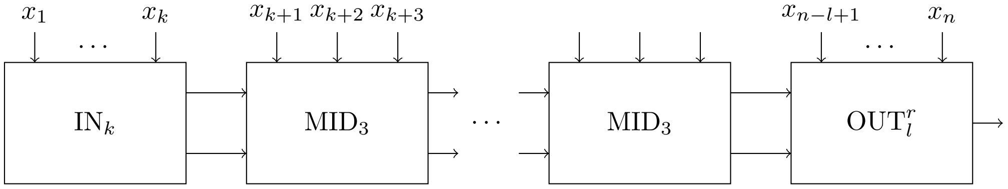
3.2 Modulo-3 Function

In [13], Kojevnikov et al. presented circuits of size $3n + O(1)$ for $\mathrm{MOD}_n^{3,r}$ (for any r). Later, Knuth [12, solution to exercise 480] analyzed their construction and proved an upper bound $3n-4$. Also, by finding the exact values for $\mathrm{size}(\mathrm{MOD}_n^{3,r})$ for all $3 \le n \le 5$ and all $0 \le r \le 2$, he made the conjecture (1). Using our program, we proved the conjectured upper bound for all n.

▶ **Theorem 1.** *For all $n \ge 3$ and all $r \in \{0,1,2\}$,*

$$\mathrm{size}(\mathrm{MOD}_n^{3,r}) \le 3n - 5 - [(n+r) \equiv 0 \bmod 3]\,.$$

Proof. As in [13], we construct the required circuit out of constant size blocks. Schematically, the circuit looks as follows.



Here, the n input bits are passed from above. What is passed from block to block (from left to right) is the pair of bits (r_0, r_1) encoding the current remainder r modulo 3 as follows: if $r = 0$, then $(r_0, r_1) = (0, 0)$; if $r = 1$, then $(r_0, r_1) = (0, 1)$; if $r = 2$, then $r_0 = 1$. The first block IN_k takes the first k input bits and computes the remainder of their sum modulo 3. It is followed by a number of MID_3 blocks each of which takes the current remainder and three new input bits and computes the new remainder. Finally, the block OUT_l^r takes the remainder and the last l input bits and outputs $\mathrm{MOD}_n^{3,r}$. The integers k, l take values in $\{2,3,4\}$ and $\{1,2,3\}$, respectively. Their exact values depend on r and $n \bmod 3$ as described below.

The theorem follows from the following upper bounds on the circuit size of the just introduced functions: $\mathrm{size}(\mathrm{IN}_2) \le 2$, $\mathrm{size}(\mathrm{IN}_3) \le 5$, $\mathrm{size}(\mathrm{IN}_4) \le 7$, $\mathrm{size}(\mathrm{MID}_3) \le 9$, $\mathrm{size}(\mathrm{OUT}_2^0) \le 5$, $\mathrm{size}(\mathrm{OUT}_1^1) \le 2$, $\mathrm{size}(\mathrm{OUT}_3^2) \le 8$. The corresponding circuits are presented in [15] by a straightforward `Python` code that verifies their correctness. (The presented code proves the mentioned upper bounds by providing explicit circuits. We have also verified that no smaller circuits exist meaning that the inequalities above are in fact equalities.) Table 4 shows how to combine the blocks to get a circuit computing $\mathrm{MOD}_n^{3,r}$ of the required size. ◀

Table 4 Choosing parameters k, m, l depending on $n \bmod 3$ and r. The circuit is composed out of blocks as follows: $\mathrm{IN}_k + m \times \mathrm{MID}_3 + \mathrm{OUT}_r^l$. For each pair $(n \bmod 3, r)$ we show three things: the triple (k, m, l); the sizes of two blocks: $\mathrm{size}(\mathrm{IN}_k)$ and $\mathrm{size}(\mathrm{OUT}_r^l)$; the size of the resulting circuit computed as $s = \mathrm{size}(\mathrm{IN}_k) + 9m + \mathrm{size}(\mathrm{OUT}_r^l)$. For example, the top left cell is read as follows: when $r = 0$ and $n = 3t$, we set $k = 4, m = t-2, l = 2$; the resulting circuit is then $\mathrm{IN}_4 + (t-2) \times \mathrm{MID}_3 + \mathrm{OUT}_0^2$; since $\mathrm{size}(\mathrm{IN}_4) = 7$ and $\mathrm{size}(\mathrm{OUT}_0^2) = 5$, the size of the circuit is $7 + 9(t-2) + 5 = 9t - 6 = 3n - 6$. There are three corner cases that are not well-defined as they require the number of MID blocks to be negative $(k = t-2)$: $(n = 3, r = 0)$, $(n = 3, r = 2)$, and $(n = 4, r = 2)$.

	$n = 3t$	$n = 3t+1$	$n = 3t+2$
$r = 0$	$(4, t-2, 2)$, $(7, 5)$, $3n-6$	$(2, t-1, 2)$, $(2, 5)$, $3n-5$	$(3, t-1, 2)$, $(5, 5)$, $3n-5$
$r = 1$	$(2, t-1, 1)$, $(2, 2)$, $3n-5$	$(3, t-1, 1)$, $(5, 2)$, $3n-5$	$(4, t-1, 1)$, $(7, 2)$, $3n-6$
$r = 2$	$(3, t-2, 3)$, $(5, 8)$, $3n-5$	$(4, t-2, 3)$, $(7, 8)$, $3n-6$	$(2, t-1, 3)$, $(2, 8)$, $3n-5$

3.3 Threshold Function

Recall that $\mathrm{THR}_n^2(x_1, \dots, x_n) = [x_1 + \dots + x_n \geq 2]$.

▶ **Theorem 2.** *For any $k \geq 4$,*

$$\mathrm{size}(\mathrm{THR}_n^k) \leq (4.5 - 2^{2-\lceil \log_2 k \rceil})(n + 2^{\lceil \log_2 k \rceil} - k) + o(n)\,.$$

Proof. For a sequence of $2m$ formal variables $y_1, z_1, \dots, y_m, z_m$, consider a function $g \in B_{2m}$ that takes $y_1, y_1 \oplus z_1, y_2, y_2 \oplus z_2, \dots, y_m, y_m \oplus z_m$ as input and outputs $\mathrm{THR}_{2m}^2(y_1, z_1, \dots, y_m, z_m)$. Note that $\mathrm{THR}_{2m}^2(y_1, z_1, \dots, y_m, z_m) = 1$ iff there is a pair containing two 1's or there are two pairs each containing at least one 1: $\mathrm{THR}_{2m}^2(y_1, z_1, \dots, y_m, z_m) = 1$ iff there exists $1 \leq i \leq m$ such that $y_i = z_i = 1$ or $\mathrm{THR}_m^2(y_1 \oplus z_1, \dots, y_m \oplus z_m) = 1$. The condition $y_i = z_i = 1$ can be computed through y_i and $y_i \oplus z_i$ using a single binary gate: $(y_i \wedge z_i) = (y_i \wedge \overline{(y_i \oplus z_i)})$. Thus,

$$g(y_1, y_1 \oplus z_1, \dots, y_m, y_m \oplus z_m) = \mathrm{THR}_m^2(y_1 \oplus z_1, \dots, y_m \oplus z_m) \vee \bigvee_{i=1}^{m} (y_i \wedge \overline{(y_i \oplus z_i)})\,.$$

Now, $\mathrm{size}(\mathrm{THR}_m^2) \leq 2m + o(m)$ as shown by Dunne [5]. Also, clearly,

$$\mathrm{size}\left(\bigvee_{i=1}^{m} (y_i \wedge \overline{(y_i \oplus z_i)})\right) \leq 2m - 1\,.$$

Thus,

$$\mathrm{size}(g) \leq 4m + o(m)\,. \tag{3}$$

To construct a circuit for THR_n^k, first, consider the case $k = 2^t$ where $t \geq 2$ is an integer. Apply $t - 1$ layers of MDFA's (as in Figure 4). It takes

$$\frac{n}{2} + n \sum_{i=1}^{t-1} 2^{2-i} = (4.5 - 2^{3-t})n$$

gates. As a result, we get bits $w_0, \dots, w_{t-2}, a_1, a_1 \oplus b_1, \dots, a_m, a_m \oplus b_m$, where $m = n/2^t$, such that

$$x_1 + \dots + x_n = w_0 + 2w_1 + \dots + 2^{t-2} w_{t-2} + 2^{t-1}(a_1 + b_1 + \dots + a_m + b_m)\,.$$

Note that $w_0 + 2w_1 + \cdots + 2^{t-2}w_{t-2} < 2^{t-1}$. Hence,

$$[x_1 + \cdots + x_n \geq 2^t] = [a_1 + b_1 + \cdots + a_m + b_m \geq 2]\,.$$

Thus, it remains to compute the function g given $2m$ bits $a_1, a_1 \oplus b_1, \ldots, a_m, a_m \oplus b_m$. By (3), it takes $4m + o(m) = 2^{2-t}n + o(n)$ gates. The total size of the constructed circuit is

$$(4.5 - 2^{3-t} + 2^{2-t})n + o(n) = (4.5 - 2^{2-t})n + o(n)\,.$$

Now, assume that $2^{t-1} < k < 2^t$ (hence $\lceil \log_2 k \rceil = t$). Clearly,

$$[x_1 + \cdots + x_n \geq k] = [(2^t - k) + x_1 + \cdots + x_n \geq 2^t]\,.$$

By the previous argument, there exists a circuit C computing $\mathrm{THR}^{2^t}_{n+(2^t-k)}$ of size

$$(4.5 - 2^{2-t})(n + (2^t - k)) + o(n) = (4.5 - 2^{2-\lceil \log_2 k \rceil})(n + 2^{\lceil \log_2 k \rceil} - k) + o(n)\,.$$

By replacing arbitrary $(2^t - k)$ inputs of C by 1's, one gets a circuit computing THR^k_n. ◀

▶ **Corollary 3.** *For* $4 \leq k = O(1)$, $\mathrm{size}(\mathrm{THR}^k_n) \leq (4.5 - 2^{2-\lceil \log_2 k \rceil})n + o(n)$. *In particular,* $\mathrm{size}(\mathrm{THR}^4_n) \leq 3.5n + o(n)$ *and* $\mathrm{size}(\mathrm{THR}^k_n) \leq 4n + o(n)$ *for* $5 \leq k \leq 8$.

We conclude by presenting an example of a reasonably small circuit that our program fails to improve though a better circuit is known. The reason is that these two circuits are quite different. Figures 6 and 7 show circuits of size 31 and 29 for THR^2_{12}. They are quite different and our program is not able to find out that the circuit of size 31 is suboptimal. The code below shows how one can construct the two circuits in the program.

```
c = Circuit(input_labels=[f'x{i}' for i in range(1, 13)], gates={})
c.outputs = add_naive_thr2_circuit(c, c.input_labels)
c.draw('thr2naive')

c = Circuit(input_labels=[f'x{i}' for i in range(1, 13)], gates={})
c.outputs = add_efficient_thr2_circuit(c, c.input_labels, 3, 4)
c.draw('thr2efficient')
```

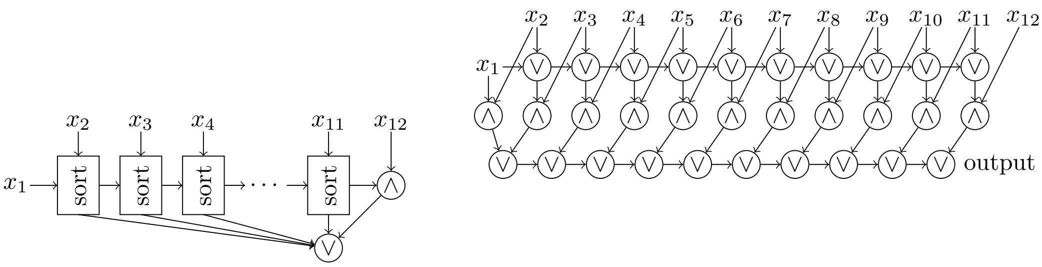


Figure 6 A circuit of size 31 for THR^2_{12}: (a) block structure and (b) gate structure. The $\mathrm{SORT}(u,v)$ block sorts two input bits as follows: $\mathrm{SORT}(u,v) = (\min\{u,v\}, \max\{u,v\}) = (u \wedge v, u \vee v)$. The circuit performs one and a half iterations of the bubble sort algorithm: one first finds the maximum bit among n input bits; then, it remains to compute the disjunction of the remaining $n-1$ bits to check whether there is at least one 1 among them. In general, this leads to a circuit of size $3n-5$.

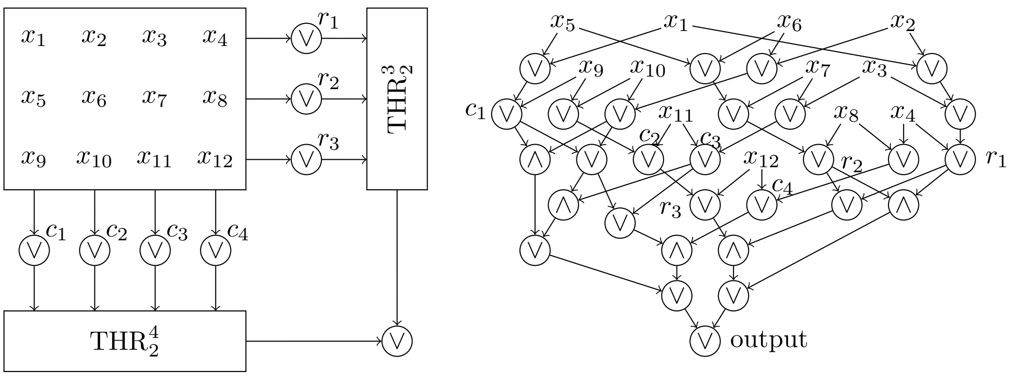


Figure 7 A circuit of size 29 for THR^2_{12}: (a) block structure and (b) gate structure. It implements a clever trick by Dunne [5]. Organize 12 input bits into a 3×4 table. Compute disjunctions r_1, r_2, r_3 of the rows and disjunctions c_1, c_2, c_3, c_4 of the columns. Then, there are at least two 1's among $x_1, \dots, x_{12}$ if and only if there are at least two 1's among either r_1, r_2, r_3 or c_1, c_2, c_3, c_4. This allows to proceed recursively. In general, it leads to a circuit of size $2n + o(n)$. (Sergeev [23] showed recently that the *monotone* circuit size of THR^2_n is $2n + \Theta(\sqrt{n})$.)

4 Further Directions

We focus mainly on proving asymptotic upper bounds for symmetric function families (that is, upper bounds that hold for every input size). A natural further step is to apply the program to specific circuits that are used in practice. It would also be interesting to extend the program so that it is able to discover the circuit from Figure 7. Finally, it would be interesting to generalize circuits for SUM_n presented in Section 3.1.2.

References

1 `https://github.com/berkeley-abc/abc`.

2 Joshua Brakensiek, Marijn Heule, John Mackey, and David Narváez. The resolution of Keller's conjecture. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I*, volume 12166 of *Lecture Notes in Computer Science*, pages 48–65. Springer, 2020. `doi:10.1007/978-3-030-51074-9_4`.

3 `https://github.com/alexanderskulikov/circuit_improvement`.

4 Evgeny Demenkov, Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. New upper bounds on the boolean circuit complexity of symmetric functions. *Inf. Process. Lett.*, 110(7):264–267, 2010. `doi:10.1016/j.ipl.2010.01.007`.

5 Paul E. Dunne. *Techniques for the analysis of monotone Boolean networks.* PhD thesis, University of Warwick, 1984.

6 Johannes Klaus Fichte, Neha Lodha, and Stefan Szeider. Sat-based local improvement for finding tree decompositions of small width. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 401–411. Springer, 2017. `doi:10.1007/978-3-319-66263-3_25`.

7 Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 89–98. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.19.

8 Winston Haaswijk, Alan Mishchenko, Mathias Soeken, and Giovanni De Micheli. SAT based exact synthesis using DAG topology families. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 53:1–53:6. ACM, 2018. doi:10.1145/3195970.3196111.

9 Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018. doi:10.1007/978-3-319-94144-8_26.

10 http://www-cs-faculty.stanford.edu/~knuth/programs.html.

11 Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions (Art of Computer Programming)*. Addison-Wesley Professional, 1 edition, 2008.

12 Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 1st edition, 2015.

13 Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. Finding efficient circuits using SAT-solvers. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 32–44. Springer, 2009. doi:10.1007/978-3-642-02777-2_5.

14 Alexander S. Kulikov. Improving circuit size upper bounds using sat-solvers. In Jan Madsen and Ayse K. Coskun, editors, *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 305–308. IEEE, 2018. doi:10.23919/DATE.2018.8342026.

15 Alexander S. Kulikov, Danila Pechenev, and Nikita Slezkin. Sat-based circuit local improvement. *CoRR*, abs/2102.12579, 2021. arXiv:2102.12579.

16 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. *ACM Trans. Comput. Log.*, 20(3):15:1–15:24, 2019. doi:10.1145/3326159.

17 Oleg Lupanov. A method of circuit synthesis. *Izvestiya VUZov, Radiofizika*, 1:120–140, 1959.

18 David E. Muller. Complexity in electronic switching circuits. *IRE Transactions on Electronic Computers*, EC-5:15–19, 1956.

19 https://pygraphviz.github.io/.

20 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Maxsat-based postprocessing for treedepth. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 478–495. Springer, 2020. doi:10.1007/978-3-030-58475-7_28.

21 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Turbocharging treewidth-bounded bayesian network structure learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3895–3903. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16508.

22 André Schidler and Stefan Szeider. Sat-based decision tree learning for large data sets. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3904–3912. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16509.

23 Igor Sergeev. On monotone circuit complexity of threshold boolean functions. *Diskretnaya Matematika*, 32:81–109, 2020. doi:10.4213/dm1547.

24 Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, and Giovanni De Micheli. The EPFL logic synthesis libraries, November 2019. arXiv:1805.05121v2.

25 Larry J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977. doi:10.1007/BF01683282.

26 Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bluebook/.

Complexity of the Cluster Vertex Deletion Problem on H-Free Graphs

Hoang-Oanh Le
Independent Researcher, Berlin, Germany

Van Bang Le
Institut für Informatik, Universität Rostock, Germany

Abstract

The well-known Cluster Vertex Deletion problem (CLUSTER-VD) asks for a given graph G and an integer k whether it is possible to delete at most k vertices of G such that the resulting graph is a cluster graph (a disjoint union of cliques). We give a complete characterization of graphs H for which CLUSTER-VD on H-free graphs is polynomially solvable and for which it is NP-complete.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory; Mathematics of computing → Graph algorithms

Keywords and phrases Cluster vertex deletion, Vertex cover, Computational complexity, Complexity dichotomy

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.68

1 Introduction and result

A very extensively studied version of graph modification problems asks to modify a given graph to a graph that satisfies a certain property $\mathcal{G}$ by deleting a minimum number of vertices. The case $\mathcal{G}$ being 'edgeless' is the well-known VERTEX COVER problem, one of the classical NP-hard problems. If $\mathcal{G}$ is a 'cluster graph', a graph in which every connected component is a clique, the corresponding problem is another well-known NP-hard problem, the CLUSTER VERTEX DELETION problem (CLUSTER-VD for short). In this paper, we revisit the computational complexity of CLUSTER-VD, formally given below.

CLUSTER-VD
Instance: A graph $G = (V, E)$ and an integer k.
Question: Does there exist a vertex set $S \subseteq V$ of size at most k such that $G - S$ is a cluster graph?

Being an hereditary property on induced subgraphs, CLUSTER-VD is NP-complete [16], even when restricted to planar graphs [20] and to bipartite graphs [21], and to bipartite graphs of maximum degree 3 [13]. Most recent works on CLUSTER-VD deal with exact, fpt and approximation algorithms [1, 2, 14, 19].

It is noticeable that there are only a few known cases where the problem can be solved efficiently: CLUSTER-VD is polynomially solvable on block graphs and split graphs [3], and on graphs of bounded treewidth [18]. On the other hand, the complexity status of CLUSTER-VD on many well-studied graph classes is still open, e.g., chordal graphs discussed in [3] and planar bipartite graphs mentioned in [4].

In this paper we initiate studying the computational complexity of CLUSTER-VD on graphs defined by forbidding certain induced subgraphs. We remark that related approaches for other problems are quite common in the literature, e.g., for VERTEX COVER (aka INDEPENDENT SET) [9, 12] and COLORING [10, 15], and that many popular graph classes are defined or characterized by forbidding induced subgraphs, e.g., chordal and bipartite graphs (by infinitely many forbidden subgraphs), and cographs and line graphs (by finitely many forbidden subgraphs).

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 68; pp. 68:1–68:10
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

All graphs considered are undirected and finite and have no multiple edges or self-loops. Let H be a given graph. A graph G is *H-free* if no induced subgraph in G is isomorphic to H. A path with n vertices and $n-1$ edges is denoted by P_n. The main result of the present paper is the following complexity dichotomy:

▶ **Theorem 1.** *Let H be a fixed graph.* CLUSTER-VD *is polynomially solvable on H-free graphs if H is an induced subgraph of the 4-vertex path P_4, and* NP*-complete otherwise.*

Theorem 1 is remindful of the main result in [15] which characterizes all graphs H for which coloring H-free graphs is easy and for which it is hard. In fact, the present paper was motivated by this H-free coloring theorem.

For a set $\mathcal{H}$ of graphs, $\mathcal{H}$-free graphs are those in which no induced subgraph is isomorphic to a graph in $\mathcal{H}$. We denote by $K_{1,n}$ the tree with $n \geq 3$ vertices and n leaves, by C_n the n-vertex cycle, and as usual, by $\overline{G}$ the complement of a graph G. The union $G + H$ of two vertex-disjoint graphs G and H is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$; we write pG for the union of p copies of G. For a subset $S \subseteq V(G)$, we let $G[S]$ denote the subgraph of G induced by S; $G - S$ stands for $G[V(G) \setminus S]$. By 'G contains an H' we mean G contains H as an induced subgraph.

A graph G is a *cluster graph* if each of its connected components is a clique. Observe that G is a cluster graph if and only if G is P_3-free. If $S \subseteq V(G)$ is a subset of vertices of G such that $G - S$ is P_3-free, then S is called a *cluster vertex deletion set* of G. An *optimal* cluster vertex deletion set is one of minimum size.

We first address the polynomial part of Theorem 1 in the next section. Then we present two new NP-completeness results for CLUSTER-VD in Sections 3 and 4. These hardness results allow us to clear the NP-completeness part of Theorem 1 in Section 5. The last section concludes the paper.

2 Polynomial cases

The polynomial part in Theorem 1 consists of six cases; see also Fig. 1 for all graphs H for which CLUSTER-VD is polynomially solvable on H-free graphs.

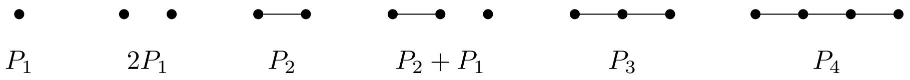


Figure 1 The graphs H for which CLUSTER-VD is polynomially solvable on H-free graphs.

Observe that H-freeness is hereditary, meaning if H' is an induced subgraph of H then H'-free graphs are H-free graphs. Thus, it suffices to prove the polynomial part only for the case where H is the 4-vertex path P_4.

We now are going to describe how to solve CLUSTER-VD in polynomial time when restricted to P_4-free graphs. P_4-free graphs are also called *cographs* [6]. More precisely, for vertex-disjoint graphs $G_i = (V_i, E_i)$, $i = 1, 2$, let $G_1 \circledcirc G_2$ be the union (or *co-join*) of G_1 and G_2,

$$G_1 \circledcirc G_2 = (V_1 \cup V_2, E_1 \cup E_2),$$

and let $G_1 \textcircled{1} G_2$ be the *join* of G_1 and G_2,

$$G_1 \textcircled{1} G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{uv \mid u \in V_1, v \in V_2\}).$$

With these notations, cographs are exactly those graphs that can be constructed from the one-vertex graph by applying the join and co-join operations. Thus, a cograph is the one-vertex graph or is the join of two smaller cographs or is the co-join of two smaller cographs.

Recall that $S \subseteq V(G)$ is a vertex cover if $G - S$ is edgeless and is a cluster vertex deletion set if $G - S$ is a cluster graph. Let $\tau(G)$ and $\varsigma(G)$ denote the vertex cover number and the cluster vertex deletion number of G, respectively,

$$\tau(G) = \min\{|S| : S \text{ is a vertex cover of } G\},$$
$$\varsigma(G) = \min\{|S| : S \text{ is a cluster vertex deletion set of } G\}.$$

We will see that $\tau(G)$ and $\varsigma(G)$ can be computed efficiently when restricted to cographs. The calculation is based on the following fact:

▶ **Lemma 2.** *For any (not necessarily P_4-free) graphs G_1 and G_2, the following relations hold:*

$$\tau(G_1 \textcircled{0} G_2) = \tau(G_1) + \tau(G_2); \tag{1}$$
$$\tau(G_1 \textcircled{1} G_2) = \min\{\tau(G_1) + |V(G_2)|, \tau(G_2) + |V(G_1)|\}; \tag{2}$$
$$\varsigma(G_1 \textcircled{0} G_2) = \varsigma(G_1) + \varsigma(G_2); \tag{3}$$
$$\varsigma(G_1 \textcircled{1} G_2) = \min\{\varsigma(G_1) + |V(G_2)|, \varsigma(G_2) + |V(G_1)|, \tau(\overline{G_1}) + \tau(\overline{G_2})\}. \tag{4}$$

Proof. (1) and (3) are trivial.

(2): Let S_i be a vertex cover of G_i of optimal size $\tau(G_i)$, $i = 1, 2$. Then $S_1 \cup V(G_2)$ and $S_2 \cup V(G_1)$ are vertex covers of $G_1 \textcircled{1} G_2$. Hence $\tau(G_1 \textcircled{1} G_2) \leq \min\{|S_1| + |V(G_2)|, |S_2| + |V(G_1)|\} = \min\{\tau(G_1) + |V(G_2)|, \tau(G_2) + |V(G_1)|\}$.

For the other direction, let S be a vertex cover of $G_1 \textcircled{1} G_2$ of optimal size, and write $S_i = S \cap V(G_i)$. Then S_i is a vertex cover of G_i, and moreover, $S_1 = V(G_1)$ or else $S_2 = V(G_2)$. Hence $\tau(G_1 \textcircled{1} G_2) \geq \min\{|S_1| + |V(G_2)|, |S_2| + |V(G_1)|\} \geq \min\{\tau(G_1) + |V(G_2)|, \tau(G_2) + |V(G_1)|\}$.

(4): Let S_i be a cluster vertex deletion set of G_i of optimal size $\varsigma(G_i)$, $i = 1, 2$. Then $S_1 \cup V(G_2)$ and $S_2 \cup V(G_1)$ are cluster vertex deletion sets of $G_1 \textcircled{1} G_2$. Hence $\varsigma(G_1 \textcircled{1} G_2) \leq \min\{|S_1| + |V(G_2)|, |S_2| + |V(G_1)|\} = \min\{\varsigma(G_1) + |V(G_2)|, \varsigma(G_2) + |V(G_1)|\}$. Let S_i be a vertex cover of $\overline{G_i}$ of optimal size $\tau(\overline{G_i})$, $i = 1, 2$. Then $S_1 \cup S_2$ is a cluster vertex deletion set of $G_1 \textcircled{1} G_2$, hence $\varsigma(G_1 \textcircled{1} G_2) \leq |S_1| + |S_2| = \tau(\overline{G_1}) + \tau(\overline{G_2})$.

For the other direction, let S be a cluster vertex deletion set of $G_1 \textcircled{1} G_2$ of optimal size, and write $S_i = S \cap V(G_i)$. Then S_i is a cluster vertex deletion set of G_i, and moreover,

- if $G_1 - S_1$ is not a clique then $S_2 = V(G_2)$, likewise
- if $G_2 - S_2$ is not a clique then $S_1 = V(G_1)$.

In these two cases, $|S| = \varsigma(G_1 \textcircled{1} G_2) \geq \min\{|S_1| + |V(G_2)|, |S_2| + |V(G_1)|\} \geq \min\{\varsigma(G_1) + |V(G_2)|, \varsigma(G_2) + |V(G_1)|\}$. In the third case where each of $G_1 - S_1$ and $G_2 - S_2$ is a clique, S_1 and S_2 are vertex covers of $\overline{G_1}$ and $\overline{G_2}$, respectively. Hence in this case, $|S| = \varsigma(G_1 \textcircled{1} G_2) = |S_1| + |S_2| \geq \tau(\overline{G_1}) + \tau(\overline{G_2})$. ◀

▶ Remark 3. For any integer $r \geq 2$, Lemma 2 holds accordingly for $G_1 \textcircled{0} G_2 \textcircled{0} \cdots \textcircled{0} G_r = G_1 \textcircled{0} (G_2 \textcircled{0} \cdots \textcircled{0} G_r)$ and $G_1 \textcircled{1} G_2 \textcircled{1} \cdots \textcircled{1} G_r = G_1 \textcircled{1} (G_2 \textcircled{1} \cdots \textcircled{1} G_r)$. We also note that Lemma 2 holds for the weighted version, too.

With each cograph $G = (V, E)$, one can associate a so-called *cotree* T of G as follows.

- The leaves of T are the vertices of G;
- Every internal node of T has a label ⓪ or ①, and has at least two children;
- No two internal nodes of T with the same label are adjacent;
- Two vertices u and v of G are (non-)adjacent if and only if the least common ancestor of u and v in T has label ① (respectively, ⓪).

In particular, the cotree of an n-vertex cograph has at most $2n - 1$ nodes.

Note that, for any internal node v of T, the subtree T_v of T rooted at v is the cotree of the subgraph of G induced by the leaves of T_v. The cograph corresponding to T_v where v has label ⓪ is the disjoint union of the cographs corresponding to the children of v. The cograph corresponding to T_v where v has label ① is the join of the cographs corresponding to the children of v.

In particular, the cotree of $\overline{G}$ can be obtained from the cotree of G by changing the label ⓪ to ① and ① to ⓪.

In [7], a linear time algorithm is given for recognizing if a given graph is a cograph, and if so, constructing its cotree. Note that the cotree can immediately be transformed to an equivalent binary tree; see Fig. 2 for an example of a cograph G and the cotree of G and its binary version. For simplification, we will use the binary cotree in our algorithm below.

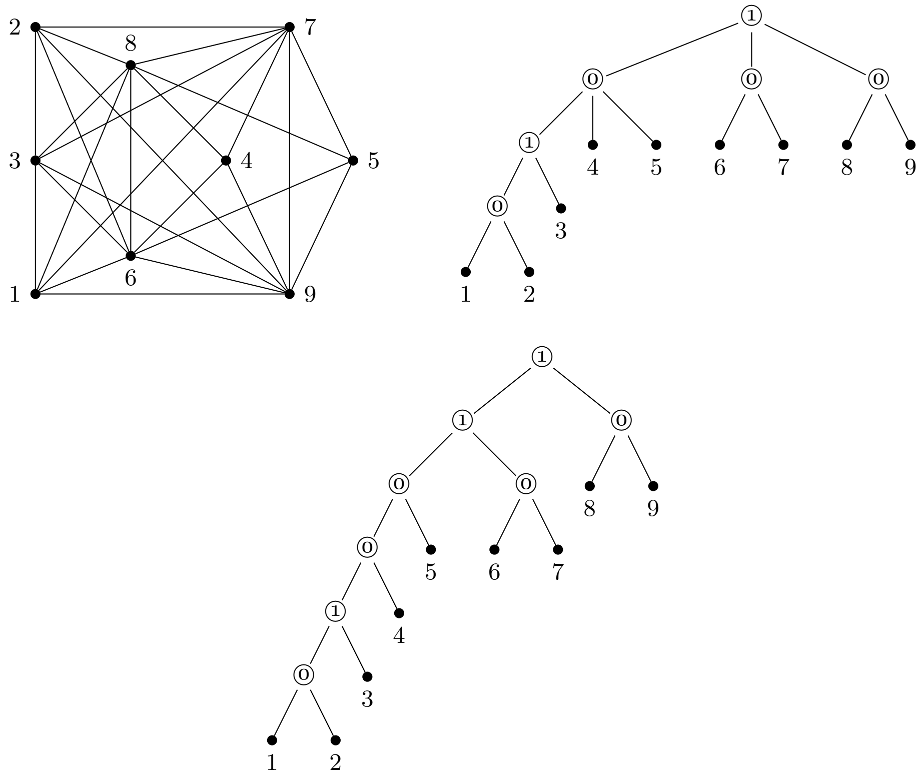


Figure 2 A cograph G, the cotree of G and its binary version.

Now, given a cograph G together with its binary cotree T, the bottom-up Algorithm 1 below computes the cluster vertex deletion number $\varsigma(G)$ of G, as suggested by Lemma 2. The algorithm uses the following notations. For a node v of T,

- if v is an internal node then $\ell(v)$ and $r(v)$ stands for the left child and the right child of v, respectively;

- $n(v)$ denotes the size of the subgraph of G induced by the leaves of T_v. Thus, if v is a leaf then $n(v) = 1$ and if v is the root of T then $n(v) = |V(G)|$;
- $\varsigma(v)$ denotes the cluster vertex deletion number of the subgraph of G induced by the leaves of T_v. Thus, if v is a leaf then $\varsigma(v) = 0$ and if v is the root of T then $\varsigma(v) = \varsigma(G)$;
- $\overline{\tau}(v)$ denotes the vertex cover number of the *complement* of the subgraph of G induced by the leaves of T_v. Thus, if v is a leaf then $\overline{\tau}(v) = 0$ and if v is the root of T then $\overline{\tau}(v) = \tau(\overline{G})$.

Algorithm 1 `computing cluster vertex deletion number.`

```
Input: A cograph $G = (V, E)$ together with its (binary) cotree $T$.
Output: $\varsigma(G)$, the cluster vertex deletion number of $G$

Traverse $T$ by post-order and let $v$ be the current node
if $v$ is a leaf then
    $n(v) \leftarrow 1$; $\overline{\tau}(v) \leftarrow 0$; $\varsigma(v) \leftarrow 0$
end
else
    $n(v) \leftarrow n(\ell(v)) + n(r(v))$
    if $v$ has label ⓪ then
        $\overline{\tau}(v) \leftarrow \min\{\overline{\tau}(\ell(v)) + n(r(v)), \overline{\tau}(r(v)) + n(\ell(v))\}$
        $\varsigma(v) \leftarrow \varsigma(\ell(v)) + \varsigma(r(v))$
    end
    if $v$ has label ① then
        $\overline{\tau}(v) \leftarrow \overline{\tau}(\ell(v)) + \overline{\tau}(r(v))$
        $\varsigma(v) \leftarrow \min\{\varsigma(\ell(v)) + n(r(v)), \varsigma(r(v)) + n(\ell(v)), \overline{\tau}(v)\}$
    end
end
```

▶ **Proposition 4.** *Given a P_4-free n-vertex graph G together with its cotree, Algorithm 1 correctly computes the cluster deletion number $\varsigma(G)$ of G in $O(n)$ time.*

Proof. The correctness of Algorithm 1 directly follows from Lemma 2. Since per node in the cotree a constant number of operations are performed, the algorithm runs in $O(n)$ time. ◀

We remark that Algorithm 1 can be slightly modified for computing a minimum cluster vertex deletion set. Also, since Lemma 2 holds accordingly for the weighted version, the minimum weight cluster vertex deletion number of cographs can be computed in linear time, too.

Another approach is based on the concept of clique-width of graphs in connection with the so-called $LinEMSOL_{\tau_{1,p}}$ problems (problems definable in monadic second-order logic and allowed in searching optimal sets of vertices with respect to some linear objective function). We refer to [8] for details. It is well known that cographs have clique-width at most 2 and a corresponding 2-expression can be constructed in linear time. Hence, every $LinEMSOL_{\tau_{1,p}}$ problem on cographs can be solved in linear time [8, Theorem 4].

Observe that the optimization version of CLUSTER-VD, MINIMUM CLUSTER-VD, is a $LinEMSOL(\tau_1, p)$ problem, since it can be expressed as follows:

minimize $|S|$ with respect to
$\forall u, v, w\big(\neg S(u) \wedge \neg S(v) \wedge \neg S(w) \wedge E(u,v) \wedge E(v,w) \wedge (u \neq w) \rightarrow E(u,w)\big)$,

where $S(x)$ means $x \in S$ and $E(x,y)$ means $xy \in E(G)$.

We remark that MINIMUM WEIGHT CLUSTER-VD is also a $LinEMSOL_{\tau_{1,p}}$ problem, hence it can be solved in linear time on cographs, too.

3 Cluster-VD on sparse graphs

Recall that CLUSTER-VD is NP-complete on bipartite graphs [21], even when restricted to bipartite graphs of maximum degree 3 [13]. In this section, we show that, for any given tree T containing two vertices of degree 3, CLUSTER-VD remains NP-complete when restricted to T-free bipartite graphs of maximum degree 3 and with arbitrarily large girth. The girth $g(G)$ of a graph G is the smallest length of a cycle in G; we set $g(G) = \infty$ if G is a forest. Thus, for any fixed $g \geq 3$, $g(G) > g$ if and only if G is $\{C_3, C_4, \ldots, C_g\}$-free.

▶ **Theorem 5.** *For any given integer $g \geq 3$ and any given tree T containing two degree-3 vertices,* CLUSTER-VD *is* NP*-complete on T-free bipartite graphs of maximum degree 3 and with girth $> g$.*

In particular, CLUSTER-VD *is* NP*-complete on bipartite graphs of maximum degree 3 and with girth $> g$.*

Proof. We give a polynomial reduction from CLUSTER-VD on bipartite graphs of maximum degree 3 to CLUSTER-VD on T-free bipartite graphs of maximum degree 3 and of girth $> g$. (Note that NP-membership of CLUSTER-VD restricted to these graphs is clear.)

First, given a bipartite graph G of maximum degree 3 with m edges, let G' be obtained from G by subdividing each edge $e = xy$ in G with three new vertices e_x, e_{xy} and e_y, thus obtaining the 5-vertex path $xe_xe_{xy}e_yy$ in G' in which all new vertices are of degree 2. Note that like G, G' is bipartite. We claim that G has a cluster vertex deletion set of size at most k if and only if G' has a cluster vertex deletion set of size at most $k + m$. For one direction, extend a cluster vertex deletion set $S \subseteq V(G)$ to a cluster vertex deletion set $S' \subseteq V(G')$ of size $|S| + m$ as follows: initially, set $S' = S$. Then, for each edge $e = xy$ in G,

- if both x and y are in S or outside S, put e_{xy} into S';
- if $x \in S$ and $y \notin S$, put e_y into S';
- if $x \notin S$ and $y \in S$, put e_x into S'.

To see that $G' - S'$ is P_3-free, notice that by construction, for each edge $e = xy$ in G, exactly one of e_x, e_{xy} and e_y is in S', and if $e_x, e_{xy} \notin S'$ then $x \in S$, and if $e_x, x \notin S'$ then $y \in S$. Since each P_3 in G' has the form xe_xe_{xy}, $e_xe_{xy}e_y$ or $e_xxe'_x$ for some edge $e = xy$ and $e' = xz$, it follows from these facts that $G' - S'$ is P_3-free.

For the other direction, suppose that G' has a cluster deletion set of size at most $k + m$, and consider such a set S' of minimum size. Then, we may assume that, for each edge $e = xy$ in G, S' contains exactly one of e_x, e_{xy} and e_y: note that $e_xe_{xy}e_y$ is a P_3, hence $|S' \cap \{e_x, e_{xy}, e_y\}| \geq 1$, and by minimality, $|S' \cap \{e_x, e_{xy}, e_y\}| \leq 2$. Now, if $|S' \cap \{e_x, e_{xy}, e_y\}| = 2$ for some edge $e = xy$ in G, then S' can be modified to a minimum cluster vertex deletion set containing exactly one of e_x, e_{xy} and e_y as follows:

- suppose that $e_x, e_{xy} \in S'$. Then $x, y \notin S'$ (if $x \in S'$ then $S' - e_x$ would be a cluster vertex deletion set of G', and if $y \in S'$ then $S' - e_{xy}$ would be a cluster vertex deletion set of G', contradicting the minimality of S'), and $S'' = S' - e_{xy} + y$ is a desired cluster vertex deletion set of minimum size;
- suppose that $e_y, e_{xy} \in S'$. Then similar to the above case, $x, y \notin S'$, and $S'' = S' - e_{xy} + x$ is a desired cluster vertex deletion set of minimum size;
- suppose that $e_x, e_y \in S'$. Then $x, y \notin S'$ (if $x \in S'$ or $y \in S'$ then $S'' = S' - e_x$, respectively $S' - e_y$, would be a cluster vertex deletion set of G', contradicting the minimality of S'), and $S'' = S' - e_x + x$ is a desired cluster vertex deletion set of minimum size.

Hence, $S = S' \cap V(G)$ has at most k vertices, and $G - S$ is P_3-free: if there would be an induced P_3 xyz in G with edges $e = xy$ and $e' = yz$, then, as $|S' \cap \{e_x, e_{xy}, e_y\}| = 1$ $= |S' \cap \{e'_y, e'_{yz}, e'_z\}|$, one of the 3-paths xe_xe_{xy}, $e_yye'_y$ and $e'_{yz}e'_zz$ would be outside S'.

Thus, G has a cluster vertex deletion set of size at most k if and only if G' has a cluster vertex deletion set of size at most $k + m$, as claimed.

Now, given $g > 0$ and a tree T with two degree-3 vertices, repeating the construction $t = \max\{\log_4(g/g(G)) + 1, |V(T)|\}$ times, the final bipartite graph has girth $4^t g(G) > g$ and maximum degree 3 and contains no induced subgraph isomorphic to T. ◀

4 Cluster-VD on dense graphs

In this section, we give a polynomial reduction from VERTEX COVER to CLUSTER-VD, showing that CLUSTER-VD remains NP-complete when restricted to $\{3P_1, 2P_2\}$-free graphs.

Recall that the VERTEX COVER problem asks, for a given graph G and an integer k, if one can delete a vertex set S of size at most k such that $G - S$ is edgeless. Let (G, k) be an instance for VERTEX COVER. We may assume that

- G is not perfect.[1] This is because VERTEX COVER is polynomially solvable on perfect graphs (see [11]); notice that G is perfect if and only if $\overline{G}$ is perfect and perfect graphs can be recognized in polynomial time [5]),
- G has girth $> g$ for any given integer $g \geq 3$ (see, e.g., [17]), and
- $k \leq |V(G)|/2$. This fact is probably known and can be easily seen as follows: given G with n vertices and an integer k, let G' be obtained from G by adding $p = \max\{0, 2k - n\}$ isolated vertices. Then $k = |V(G')|/2$ and $(G, k) \in$ VERTEX COVER if and only if $(G', k) \in$ VERTEX COVER. Notice that like G, G' satisfies the first two conditions, too.

From (G, k) we construct an equivalent instance (G', k') for CLUSTER-VD as follows: G' is obtained from two disjoint copies of $\overline{G}$, G_1 and G_2, by adding all possible edges between $V(G_1)$ and $V(G_2)$. Set $k' = 2k$.

We argue that $(G, k) \in$ VERTEX COVER if and only $(G', k') \in$ CLUSTER-VD. First, let $S \subset V(G)$ with $|S| \leq k$ be such that $G - S$ is edgeless. Let S_1 and S_2 be the copy of S in G_1 and G_2, respectively. Then $G_i - S_i$ is a clique in $G_i = \overline{G}$, hence $G' - S'$ is a clique in G' where $S' = S_1 \cup S_2$ with $|S'| = 2|S| \leq 2k = k'$. Conversely, let $S' \subseteq V(G')$ be a cluster vertex deletion set of G' with $|S'| \leq k' = 2k$. Observe that $S' \cap V(G_i)$ is a proper nonempty subset of both $V(G_1)$ and $V(G_2)$: if for some i, $S' \cap V(G_i) = \emptyset$ then G_i (hence G) would be perfect, and if $V(G_i) \subset S'$ then $2k \geq |S'| > |V(G_i)| = |V(G)|$, contradicting $k \leq |V(G)|/2$. It follows from the above that $G' - S'$ is a single clique, implying for each i, $G_i - S_i$ is a

[1] Actually, we will only use the fact that $\overline{G}$ contains at least one induced P_3.

clique in G_i where $S_i = S' \cap V(G_i)$. Since $|S'| \leq 2k$, $|S_1| \leq k$ or $|S_2| \leq k$. Let $|S_1| \leq k$, say, and let $S \subseteq V(G)$ be the set of the corresponding vertices in G. Then $G - S$ is edgeless with $|S| \leq k$.

We have seen that G has a vertex cover of size at most k if and only if G' has a cluster vertex deletion set of size at most k', as claimed.

Now, observe that, for any *connected* graph X, if G is X-free then G' is $\overline{X}$-free. Since G has girth $> g$, we obtain:

▶ **Theorem 6.** *For any fixed $g \geq 3$,* CLUSTER-VD *is* NP*-complete on $\{\overline{C_3}, \overline{C_4}, \ldots, \overline{C_g}\}$-free graphs.*

In particular, CLUSTER-VD *is* NP*-complete on $\{3P_1, 2P_2\}$-free graphs.*

5 NP-completeness cases

In this section we give the proof of the NP-completeness part of Theorem 1.

Let H be a fixed graph. By Proposition 4, CLUSTER-VD is polynomially solvable on H-free graphs whenever H is an induced subgraph of the 4-vertex path P_4. The following fact is easy to see:

▶ **Observation 7.** *A graph is an induced subgraph of the 4-path P_4 if and only if it is a $\{3P_1, 2P_2\}$-free forest.*

Thus, it remains to consider the cases where H contains a cycle or a $3P_1$ or a $2P_2$ as an induced subgraph.

Now, if H contains a cycle then graphs of girth $> g = |V(H)|$ are H-free, hence Theorem 5 implies that CLUSTER-VD is NP-complete on H-free graphs. If H contains a $3P_1$ or a $2P_2$ then $\{3P_1, 2P_2\}$-free graphs are H-free graphs, hence Theorem 6 implies that CLUSTER-VD is NP-complete on H-free graphs.

The proof of Theorem 1 is complete.

6 Conclusion

We have found a complete characterization of graphs H for which CLUSTER-VD on H-free graphs is polynomially solvable and for which it is NP-complete (Theorem 1).

We remark that a complexity dichotomy for VERTEX COVER on H-free graphs, like Theorem 1 for CLUSTER-VD, seems very hard to achieve. Indeed, it is a long-standing open problem whether there exists a constant t for which VERTEX COVER is NP-complete on P_t-free graphs. So far it is known that such a constant t, if any, must be at least 7 [12].

Let $\mathcal{H}$ be a set of (possibly infinitely many) graphs. A natural question generalizing the case of one forbidden induced subgraph is: what is the complexity of CLUSTER-VD on $\mathcal{H}$-free graphs? The case $\mathcal{H} = \{H\}$ is completely solved by Theorem 1. The case $\mathcal{H} = \{C_\ell \mid \ell \geq 4\}$ addressed in [3] is still open. The next step may be the case of two-element sets $\mathcal{H} = \{H_1, H_2\}$. This case is more complex and currently we are investigating the case $\mathcal{H} = \{H, \overline{H}\}$. Another interesting problem is to clear the complexity of CLUSTER-VD on line graphs, a well-studied graph class defined by excluding nine small induced subgraphs.

References

1 Manuel Aprile, Matthew Drescher, Samuel Fiorini, and Tony Huynh. A tight approximation algorithm for the cluster vertex deletion problem. *Math. Program.*, 2022. doi:10.1007/s10107-021-01744-w.

2 Anudhyan Boral, Marek Cygan, Tomasz Kociumaka, and Marcin Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory Comput. Syst.*, 58(2):357–376, 2016. doi:10.1007/s00224-015-9631-7.

3 Yixin Cao, Yuping Ke, Yota Otachi, and Jie You. Vertex deletion problems on chordal graphs. *Theor. Comput. Sci.*, 745:75–86, 2018. doi:10.1016/j.tcs.2018.05.039.

4 Dibyayan Chakraborty, L. Sunil Chandran, Sajith Padinhatteeri, and Raji R. Pillai. Algorithms and complexity of s-club cluster vertex deletion. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms - 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021, Proceedings*, volume 12757 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2021. doi:10.1007/978-3-030-79987-8_11.

5 Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul D. Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25(2):143–186, 2005. doi:10.1007/s00493-005-0012-8.

6 Derek G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discret. Appl. Math.*, 3(3):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.

7 Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985. doi:10.1137/0214065.

8 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.

9 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

10 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of coloring graphs with forbidden subgraphs. *J. Graph Theory*, 84(4):331–363, 2017. doi:10.1002/jgt.22028.

11 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

12 Andrzej Grzesik, Tereza Klimosová, Marcin Pilipczuk, and Michal Pilipczuk. Polynomial-time algorithm for maximum weight independent set on P_6-free graphs. *ACM Trans. Algorithms*, 18(1):4:1–4:57, 2022. doi:10.1145/3414473.

13 Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. On the d-claw vertex deletion problem. *CoRR*, abs/2203.06766, 2022. doi:10.48550/arXiv.2203.06766.

14 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 47(1):196–217, 2010. doi:10.1007/s00224-008-9150-x.

15 Daniel Král, Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. 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 254–262. Springer, 2001. doi:10.1007/3-540-45477-2_23.

16 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

17 Owen J. Murphy. Computing independent sets in graphs with large girth. *Discret. Appl. Math.*, 35(2):167–170, 1992. doi:10.1016/0166-218X(92)90041-8.

18 Ignasi Sau and Uéverton dos Santos Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. *Inf. Comput.*, 281:104812, 2021. doi:10.1016/j.ic.2021.104812.

19 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory Comput. Syst.*, 65(2):323–343, 2021. doi:10.1007/s00224-020-10005-w.

20 Mihalis Yannakakis. Node- and edge-deletion np-complete problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 253–264. ACM, 1978. doi:10.1145/800133.804355.

21 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981. doi:10.1137/0210022.

Reducing the Vertex Cover Number via Edge Contractions

Paloma T. Lima
Computer Science Department, IT University of Copenhagen, Denmark

Vinicius F. dos Santos
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Uéverton S. Souza
Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil
Institute of Informatics, University of Warsaw, Warsaw, Poland

Prafullkumar Tale
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

The Contraction(vc) problem takes as input a graph G on n vertices and two integers k and d, and asks whether one can contract at most k edges to reduce the size of a minimum vertex cover of G by at least d. Recently, Lima et al. [MFCS 2020, JCSS 2021] proved, among other results, that unlike most of the so-called *blocker problems*, Contraction(vc) admits an XP algorithm running in time $f(d) \cdot n^{\mathcal{O}(d)}$. They left open the question of whether this problem is FPT under this parameterization. In this article, we continue this line of research and prove the following results:

- Contraction(vc) is W[1]-hard parameterized by $k + d$. Moreover, unless the ETH fails, the problem does not admit an algorithm running in time $f(k + d) \cdot n^{o(k+d)}$ for any function f. In particular, this answers the open question stated in Lima et al. [MFCS 2020] in the negative.
- It is NP-hard to decide whether an instance (G, k, d) of Contraction(vc) is a Yes-instance even when $k = d$, hence enhancing our understanding of the classical complexity of the problem.
- Contraction(vc) can be solved in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$. This XP algorithm improves the one of Lima et al. [MFCS 2020], which uses Courcelle's theorem as a subroutine and hence, the $f(d)$-factor in the running time is non-explicit and probably very large. On the other hand, this shows that when $k = d$, the problem is FPT parameterized by d (or by k).

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

Keywords and phrases Blocker problems, edge contraction, vertex cover, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.69

Related Version *Full Version*: https://arxiv.org/abs/2202.03322

Funding *Vinicius F. dos Santos*: Grant APQ-01707-21 Minas Gerais Research Support Foundation (FAPEMIG) and Grant 312069/2021-9 National Council for Scientific and Technological Development (CNPq).
Ignasi Sau: CAPES-PRINT Institutional Internationalization Program, process 88887.371209/2019-00, and projects DEMOGRAPH (ANR-16-CE40-0028), ESIGMA (ANR-17-CE23-0010), ELIT (ANR-20-CE48-0008-01), and UTMA (ANR-20-CE92-0027).
Uéverton S. Souza: This research has received funding from Rio de Janeiro Research Support Foundation (FAPERJ) under grant agreement E-26/201.344/2021, National Council for Scientific and Technological Development (CNPq) under grant agreement 309832/2020-9, and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement CUTACOMBS (No. 714704).

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 69; pp. 69:1–69:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Prafullkumar Tale: The author has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement SYSTEMATICGRAPH (No. 725978).

Acknowledgements The last author would like to thank Roohani Sharma for insightful discussions.

1 Introduction

Graph modification problems have been extensively studied in theoretical computer science, in particular for their vast expressive power and their applicability in a number of scenarios. Such problems can be generically defined as follows. For a fixed graph class $\mathcal{F}$ and a fixed set $\mathcal{M}$ of allowed graph modification operations, such as vertex deletion, edge deletion, edge addition, edge editing or edge contraction, the $\mathcal{F}$-$\mathcal{M}$-Modification problem takes as input a graph G and a positive integer k, and the goal is to decide whether at most k operations from $\mathcal{M}$ can be applied to G so that the resulting graph belongs to the class $\mathcal{F}$. For most natural graph classes $\mathcal{F}$ and modification operations $\mathcal{M}$, the $\mathcal{F}$-$\mathcal{M}$-Modification problem is NP-hard [15, 16]. To cope up with this hardness, these problems have been examined via the lens of parameterized complexity [1, 3]. With an appropriate choice of $\mathcal{F}$ and the allowed modification operations $\mathcal{M}$, $\mathcal{F}$-$\mathcal{M}$-Modification can encapsulate well-studied problems like Vertex Cover, Feedback Vertex Set (FVS), Odd Cycle Transversal (OCT), Chordal Completion, or Cluster Editing, to name just a few.

The most natural and well-studied modification operations are, probably in this order, vertex deletion, edge deletion, and edge addition. In recent years, the *edge contraction* operation has begun to attract significant scientific attention. (When contracting an edge uv in a graph G, we delete u and v from G, add a new vertex and make it adjacent to vertices that were adjacent to u or v.) In parameterized complexity, $\mathcal{F}$-Contraction problems, i.e., $\mathcal{F}$-$\mathcal{M}$-Modification problems where the only modification operation in $\mathcal{M}$ is edge contraction, are usually studied with the number of edges allowed to contract, k, as the parameter. A series of more than 15 recent papers studied the parameterized complexity of $\mathcal{F}$-Contraction for various graph classes $\mathcal{F}$ (see [14] and the references therein, as well as the full version of this article, for the precise list of these classes $\mathcal{F}$).

For all the $\mathcal{F}$-$\mathcal{M}$-Modification problems mentioned above, a typical definition of the problem contains a description of the target graph class $\mathcal{F}$. For example, Vertex Cover, FVS, and OCT are $\mathcal{F}$-$\mathcal{M}$-Modification problems where $\mathcal{F}$ is the collection of edgeless graphs, forests, and bipartite graphs, respectively, and $\mathcal{M}$ contains only vertex deletion. Recently, a different formulation of these graph modification problems, called *blocker problems*, has been considered. In this formulation, the target graph class is defined in a *parametric way* from the input graph. To make the statement of such problems precise, consider an invariant $\pi : \mathcal{G} \mapsto \mathbb{N}$, where $\mathcal{G}$ is the collection of all graphs. For a fixed invariant π, a typical input of a blocker problem consists of a graph G, a budget k, and a threshold value d, and the question is whether G can be converted into a graph G' using at most k allowed modifications such that $\pi(G') \leq \pi(G) - d$. This is the same as determining whether (G, k, d) is a Yes-instance of $\mathcal{F}^{\pi}_{G,d}$-$\mathcal{M}$-Modification where $\mathcal{F}^{\pi}_{G,d} = \{G' \in \mathcal{G} \mid \pi(G') \leq \pi(G) - d\}$.

Consider the following examples of this formulation. For the invariant $\pi(G) = |E(G)|$, threshold $d = |E(G)|$, and vertex deletion as the modification operation in $\mathcal{M}$, $\mathcal{F}^{\pi}_{G,d}$-$\mathcal{M}$-Modification is the same as Vertex Cover. Setting the threshold d to a fixed integer p leads to Partial Vertex Cover. In a typical definition of this problem, the input is a graph G and two integers k, p, and the objective is to decide whether there is a set of vertices of size at most k that has at least p edges incident on it. Consider another example when

$\pi(G) = \texttt{vc}(G)$, the size of a minimum vertex cover of G, the threshold value $d = \texttt{vc}(G) - 1$, and the allowed modification operation is edge contraction. To reduce the size of a minimum vertex cover from $\texttt{vc}(G)$ to 1 by k edge contractions, we need to find a connected vertex cover of size $k + 1$. Hence, in this case $\mathcal{F}^{\pi}_{G,d}$-$\mathcal{M}$-Modification is the same as Connected Vertex Cover. In all these cases, we can think of the set of vertices or edges involved in the modifications as "blocking" the invariant π, that is, preventing π from being smaller.

With "vertex deletion" or "edge deletion" as the allowed graph modification operation, blocker problems have been investigated for numerous graph invariants (see the full version of this article for an exhaustive list of these invariants, along with the appropriate references). Blocker problems for edge contraction have already been studied with respect to the chromatic number, clique number, independence number [6, 13], the domination number [8, 10], total domination number [9], and the semitotal domination number [7].

This article is strongly motivated by the results in [12]. They proved, in particular, that it is coNP-hard to test whether we can reduce the size of a minimum feedback vertex set or of a minimum odd cycle transversal of a graph by one, i.e., $d = 1$, by performing one edge contraction, i.e., $k = 1$. This is consistent with earlier results, as blocker problems are generally very hard, and become polynomial-time solvable only restricted to specific graph classes. However, the notable exception is when the invariant is the size of a minimum vertex cover of the input graph. We define the problem before mentioning existing results and our contribution (G/F denotes the graph obtained from G by contracting the edges in F).

Contraction($\texttt{vc}$)
Input: An undirected graph G and two non-negative integers k and d.
Question: Does there exist a set $F \subseteq E(G)$ such that $|F| \leq k$ and $\texttt{vc}(G/F) \leq \texttt{vc}(G) - d$?

Our results. A simple reduction, briefly mentioned in [12], shows that the above problem is NP-hard for *some* k in $\{d, d+1, \ldots, 2d\}$. In our first result, we enhance our understanding of the classical complexity of the problem and prove that the problem is NP-hard even when $k = d$. As any edge contraction can decrease $\texttt{vc}(G)$ by at most one, if $k < d$ then the input instance is a trivial No-instance. To state our first result, we introduce the notation of $\mathsf{rank}(G)$, which is the number of vertices of G minus its number of connected components (or equivalently, the number of edges of a set of spanning trees of each of the connected components of G). Note that it is sufficient to consider values of k that are at most $\mathsf{rank}(G)$, as otherwise it is possible to transform G to an edgeless graph with at most k contractions, and therefore in this case G is a yes-instance for Contraction($\texttt{vc}$) if and only if $\texttt{vc}(G) \geq d$.

▶ **Theorem 1.** *To decide whether an instance (G, k, d) of* Contraction($\texttt{vc}$) *is a* Yes-*instance is*
- coNP-hard *if $k = \mathsf{rank}(G)$,*
- coNP-hard *if $k < \mathsf{rank}(G)$ and $2d \leq k$, and*
- NP-hard *if $k < \mathsf{rank}(G)$ and $k = d + \frac{\ell-1}{\ell+3} \cdot d$ for any integer $\ell \geq 1$ such that k is an integer.*

As one needs to contract at least d edges to reduce the size of a minimum vertex cover by d, the above theorem, for $\ell = 1$, implies that the problem is para-NP-hard when parameterized by the "excess over the lower bound", i.e., by $k - d$. Since we can assume that $d \leq k$, d is a "stronger" parameter than k. One of the main results of [12] is an XP algorithm for Contraction($\texttt{vc}$) with running time $f(d) \cdot n^{\mathcal{O}(d)}$. Here, and in the rest of the article, we denote by n the number of vertices of the input graph. The authors explicitly asked whether the problem admits an FPT algorithm parameterized by d. As our next result, we answer this question in the negative by proving that such an algorithm is highly unlikely, even when parameterized by the larger parameter $d + k$ (or equivalently, just k, as discussed above).

▶ **Theorem 2.** Contraction(vc) *is* W[1]-hard *parameterized by $k + d$. Moreover, unless the* ETH *fails*[1]*, it does not admit an algorithm running in time $f(k + d) \cdot n^{o(k+d)}$ for any computable function $f : \mathbb{N} \mapsto \mathbb{N}$. The result holds even if we assume that the input (G, k, d) is such that $k < \mathsf{rank}(G)$ and $d \le k < 2d$, and G is a bipartite graph with a bipartition $\langle X, Y\rangle$ such that X is a minimum vertex cover of G.*

For the XP algorithm in [12], the authors did not explicitly mention an upper bound on the corresponding function f, but it is expected to be quite large since it uses Courcelle's theorem [2] as a subroutine. Our next result provides a concrete upper bound on the running time, and distinguishes in a precise way the contribution of k and d.

▶ **Theorem 3.** *There exists an algorithm that solves* Contraction(vc) *in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$. Moreover, for an input (G, k, d), the algorithm runs in time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$ unless $k < \mathsf{rank}(G)$ and $d \le k < 2d$.*

Note that the above result implies, in particular, that the problem is FPT parameterized by d when $k - d$ is a constant.

Our methods. A central tool in both our negative and positive results is Lemma 6, which allows us to reformulate the problem as follows. As discussed later, by applying appropriate FPT reductions to the input graph G, it is possible to assume that we have at hand a minimum vertex cover X of G. We say that a set of edges F is a *solution* of (G, k, d) if $|F| \le k$ and $\mathsf{vc}(G/F) \le \mathsf{vc}(G) - d$. Lemma 6 implies that there exists such a solution (i.e., an edge set) if and only if there exist vertex subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that the pair $\langle X_s, Y_s\rangle$, which we call a *solution pair*, satisfies the technical conditions mentioned in its statement (and which we prefer to omit here). This reformulation allows us to convert the problem of finding a subset F of edges to the problem of modifying the given minimum vertex cover X to obtain another vertex cover $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ such that $|X_{\mathsf{el}}| \le |X| + (k - d)$ and $\mathsf{rank}(X_{\mathsf{el}}) \ge k$. Here, we define $\mathsf{rank}(X_{\mathsf{el}}) := \mathsf{rank}(G[X_{\mathsf{el}}])$. See the full version for an illustration.

In our hardness reductions, another simple, yet critical, tool is Lemma 7, which states that if there is a vertex which is adjacent to a pendant vertex (i.e., a vertex of degree one), then there is a solution pair that does not contain this vertex. We present overviews of the reductions in Section 3 and in the full version to demonstrate the usefulness of these two lemmas in the respective hardness results. The reduction that we use to prove the third item in the statement of Theorem 1 (which is the most interesting case) is from a variant of Multicolored Independent Set, while the one in the proof of Theorem 2 is from Edge Induced Forest, a problem that we define and that we prove to be W[1]-hard in Theorem 8, by a parameter preserving reduction from, again, Multicolored Independent Set. It is worth mentioning that the W[1]-hardness in Theorem 8 holds even if we assume that the input graph G is a bipartite graph with a bipartition $\langle X, Y\rangle$ such that X is a minimum vertex cover of G, and such that $k < \mathsf{rank}(G)$ and $d \le k < 2d$. This case is the crux of the difficulty of the problem. This becomes clear in the XP algorithm of Theorem 3 that we proceed to discuss.

[1] The Exponential Time Hypothesis (ETH) is a conjecture stating that N-variable 3-SAT cannot be solved in time $2^{o(N)}$. We refer the reader to [4, Chapter 14] for the formal definition and related literature.

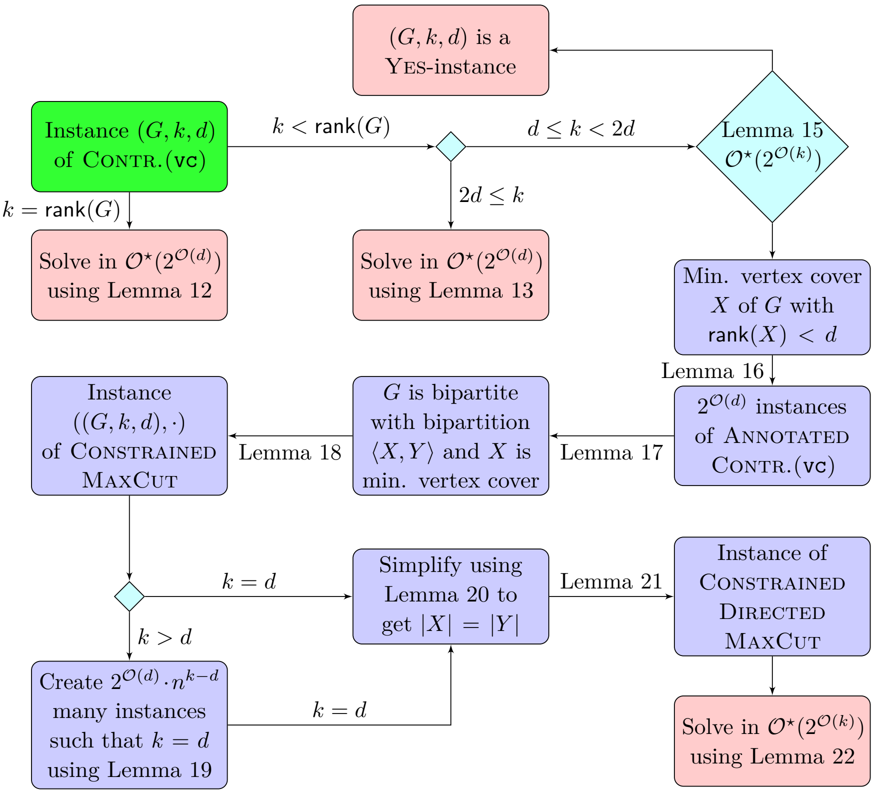

(G, k, d) is a Yes-instance
Instance (G, k, d) of Contr.(vc)
$k < \mathsf{rank}(G)$
$d \leq k < 2d$
Lemma 15 $\mathcal{O}^\star(2^{\mathcal{O}(k)})$
$k = \mathsf{rank}(G)$
$2d \leq k$
Solve in $\mathcal{O}^\star(2^{\mathcal{O}(d)})$ using Lemma 12
Solve in $\mathcal{O}^\star(2^{\mathcal{O}(d)})$ using Lemma 13
Min. vertex cover X of G with $\mathsf{rank}(X) < d$
Lemma 16
Instance $((G, k, d), \cdot)$ of Constrained MaxCut
Lemma 18
G is bipartite with bipartition $\langle X, Y\rangle$ and X is min. vertex cover
Lemma 17
$2^{\mathcal{O}(d)}$ instances of Annotated Contr.(vc)
$k = d$
Simplify using Lemma 20 to get $|X| = |Y|$
Lemma 21
Instance of Constrained Directed MaxCut
$k > d$
Create $2^{\mathcal{O}(d)} \cdot n^{k-d}$ many instances such that $k = d$ using Lemma 19
$k = d$
Solve in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ using Lemma 22

Figure 1 Diagram of the algorithm for Contraction(vc) given by Theorem 3. Recall that we can assume that $d \leq k \leq \mathsf{rank}(G)$, hence the case distinction considered in the beginning is exhaustive. Note also that, in the case where $d \leq k < 2d$, it holds that $\mathcal{O}^\star(2^{\mathcal{O}(k)}) = \mathcal{O}^\star(2^{\mathcal{O}(d)})$.

The algorithm for Contraction(vc), which is our main technical contribution, is provided in Section 4 and summarized in the diagram of Figure 1. By a standard Knapsack-type dynamic programming, mentioned in [12], we can assume that the input graph G is connected. We distinguish three cases depending on the relation between k, d, and $\mathsf{rank}(G)$. The first two cases are easy, and can be solved in time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$, by essentially running an FPT algorithm to determine whether $\mathsf{vc}(G) < d$; see Lemma 12 and Lemma 13. We now present an overview of the algorithm for the third case, namely when its input (G, k, d) is with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$ (cf. Lemma 14). Inspired by Lemma 6, we introduce an annotated version of the problem called Annotated Contraction(vc). We first argue (cf. Lemma 15) that there is a $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ algorithm that either correctly concludes that (G, k, d) is a Yes-instance of Contraction(vc) or finds a minimum vertex cover X of G such that $\mathsf{rank}(X) < d$. Using X, we can construct $2^{\mathcal{O}(d)}$ many instances of Annotated Contraction(vc) such that (G, k, d) is a Yes-instance of Contraction(vc) if and only if at least one of these new instances is a Yes-instance of Annotated Contraction(vc) (cf. Lemma 16). Hence, it suffices to design an algorithm to solve Annotated Contraction(vc). We show that we can apply a simple reduction rule (cf. Lemma 17) that allows us to assume that the input graph G of Annotated Contraction(vc) is bipartite with bipartition $\langle X, Y\rangle$ such that X is a minimum vertex cover of G, as mentioned above.

A solution of an instance of Annotated Contraction(vc) is a solution pair $\langle X_s, Y_s\rangle$ as stated in Lemma 6. We find convenient to present an algorithm that finds a partition $\langle V_L, V_R\rangle$ of $V(G)$ instead of a solution pair $\langle X_s, Y_s\rangle$. To formalize this, we introduce the problem called Constrained MaxCut and we show it to be equivalent to Annotated Contraction(vc) (cf. Lemma 18). We partition the input instances of Constrained MaxCut into the following two types: (1) $k = d$, and (2) $k > d$. For the instances of the second type, we construct $2^{\mathcal{O}(d)} \cdot n^{k-d}$ many instances of the first type such that the input instance is a Yes-instance if and only if at least one of these newly created instances is a Yes-instance (cf. Lemma 19). We remark that this is the only step in the whole algorithm where an n^{k-d}-factor appears (note that this is unavoidable by Theorem 8).

Finally, to handle the instances of the first type (i.e., with $k = d$), we apply a simplification based on the existence of a matching saturating X (cf. Lemma 20), we introduce a directed variation of the problem called Constrained Directed MaxCut, and we prove it to be equivalent to its undirected version (cf. Lemma 21). We then present a dynamic programming algorithm, with running time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, that critically uses the fact that $k = d$ (cf. Lemma 22), in particular to "merge" appropriately some directed cycles in order to obtain a directed *acyclic* graph, whose topological ordering gives a natural way to process the vertices of the input graph in a dynamic programming fashion. At the end of Section 4 we present an overview of this algorithm.

Organization. Due to space limitations, in this extended abstract most of the technical contents of the paper can only be found in the full version, in particular the proofs of the statements marked with "($\star$)". In Section 2 we present some notations and preliminary results about Contraction(vc). The proof of Theorem 1 can be found in the full version. In Section 3 we present the proof of Theorem 2. Section 4 is the most technical part of the paper, and contains the description of the algorithm to solve Contraction(vc) mentioned in Theorem 3. We conclude the article in Section 5 with some open problems.

2 Preliminaries

We use standard graph-theoretic notation, and we refer the reader to [5] for any undefined notation. Similarly, we use standard terminology of parameterized complexity, and we refer the reader to [4]. For completeness, we present in the full version the required basic preliminaries about graph theory and parameterized complexity, and we provide here some non-standard definitions and useful properties of the Contraction(vc) problem.

For a positive integer q, we denote the set $\{1, 2, \ldots, q\}$ by $[q]$. We use $\mathbb{N}$ to denote the collection of all non-negative integers. A *spanning forest* of a graph is a collection of spanning trees of its connected components. As already mentioned in Section 1, the *rank* of a graph G, denoted by $\mathsf{rank}(G)$, is the number of edges of a spanning forest of G. The *rank* of a set $X \subseteq V(G)$ of vertices, denoted by $\mathsf{rank}(X)$, is the rank of $G[X]$. The *rank* of a set $F \subseteq V(G)$ of edges, denoted by $\mathsf{rank}(F)$, is the rank of $V(F)$. Note that an edge contraction decreases the rank of a graph G by exactly one. We present a couple of observations regarding an instance (G, k, d) of the Contraction(vc) problem. Later, we present a lemma that helps us to characterize the problem as finding a vertex cover with special properties.

▶ **Observation 4** ($\star$)**.** *Consider an instance (G, k, d) of* Contraction(vc) *such that $k = \mathsf{rank}(G)$. Then, (G, k, d) is a* Yes*-instance if and only if $d \leq \mathsf{vc}(G)$.*

▶ **Observation 5** ($\star$)**.** *Consider an instance (G, k, d) of* Contraction(vc) *such that G is a connected graph, $k < \mathsf{rank}(G)$, and $2d \leq k$. Then, (G, k, d) is a* Yes*-instance if and only if $d < \mathsf{vc}(G)$.*

Suppose that (G, k, d) is a YES-instance of CONTRACTION(vc). We say that a set $F \subseteq E(G)$ is a *solution* of (G, k, d) if $|F| \leq k$ and $\mathsf{vc}(G/F) \leq \mathsf{vc}(G) - d$. Fix a minimum vertex cover X of G. As X is a vertex cover, for every edge in F, *at least one* of its endpoints is in X. We argue that one can construct an *enlarged vertex cover* X_{el} of G such that for every edge in F, *both* of its endpoints are in X_{el}. Also, X_{el} is not much larger than X. In order to construct X_{el} from X, one needs to remove and add some vertices to X. We denote the removed and added vertices by X_s and Y_s, respectively, and call $\langle X_s, Y_s \rangle$ a *solution pair*. See the corresponding figure in the full version for an illustration. The following lemma relates a solution (a set of edges) to a solution pair (a tuple of disjoint vertex sets).

▶ **Lemma 6** ($\star$). *Consider a connected graph G, a minimum vertex cover X of G, a proper subset F of edges of a spanning forest of G (i.e., $|F| < \mathsf{rank}(G)$), and a non-negative integer d. Then, $\mathsf{vc}(G/F) \leq \mathsf{vc}(G) - d$ if and only if there exists subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $X_{\mathsf{el}} := (X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq |F|$, and (iii) $|Y_s| - |X_s| \leq |F| - d$, i.e., $|X_{\mathsf{el}}| \leq |X| + |F| - d$.*

In the following lemma, we argue that there exists a solution pair $\langle X_s, Y_s \rangle$ such that X_s does not contain any vertex in X which is adjacent to a pendant vertex. For example, in the aforementioned figure in the full version, there exists a solution pair $\langle X_s, Y_s \rangle$ such that $x_1 \notin X_s$.

▶ **Lemma 7** ($\star$). *Consider a connected graph G, a minimum vertex cover X of G, and two integers ℓ and d. Suppose that there exists a vertex x° in X which is adjacent to a pendant vertex. Suppose that there are subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq \ell$, and (iii) $|Y_s| - |X_s| \leq \ell - d$. Then, there are subsets $X'_s \subseteq X$ and $Y'_s \subseteq V(G) \setminus X$ that satisfy these three conditions and $x^\circ \notin X'_s$.*

3 W[1]-hardness results

In this section we prove Theorem 2. To do so, we introduce the EDGE INDUCED FOREST problem, defined below, and present a parameter preserving reduction from MULTICOLORED INDEPENDENT SET to it. This reduction, along with known results about MULTICOLORED INDEPENDENT SET (cf. the preliminaries in the full version), imply the corresponding result for EDGE INDUCED FOREST. We then present a parameter preserving reduction from EDGE INDUCED FOREST to CONTRACTION(vc). This reduction, along with Theorem 8, yield Theorem 2.

EDGE INDUCED FOREST
Input: A graph G and an integer ℓ.
Question: Is there a set F of at least ℓ edges in G such that $G[V(F)]$ is a forest?

We note that a similar problem called INDUCED FOREST has already been studied. In this problem, the input is the same but the objective is to find a subset X of *vertices* of G of size at least ℓ such that $G[X]$ is a forest. The general result of Khot and Raman [11] implies that INDUCED FOREST is W[1]-hard when parameterized by the size of the solution ℓ. As expected, we can prove a similar result for EDGE INDUCED FOREST.

▶ **Theorem 8** ($\star$). EDGE INDUCED FOREST, *parameterized by the size of the solution ℓ, is* W[1]-hard. *Moreover, unless the* ETH *fails, it does not admit an algorithm running in time $f(\ell) \cdot n^{o(\ell)}$ for any computable function $f : \mathbb{N} \mapsto \mathbb{N}$.*

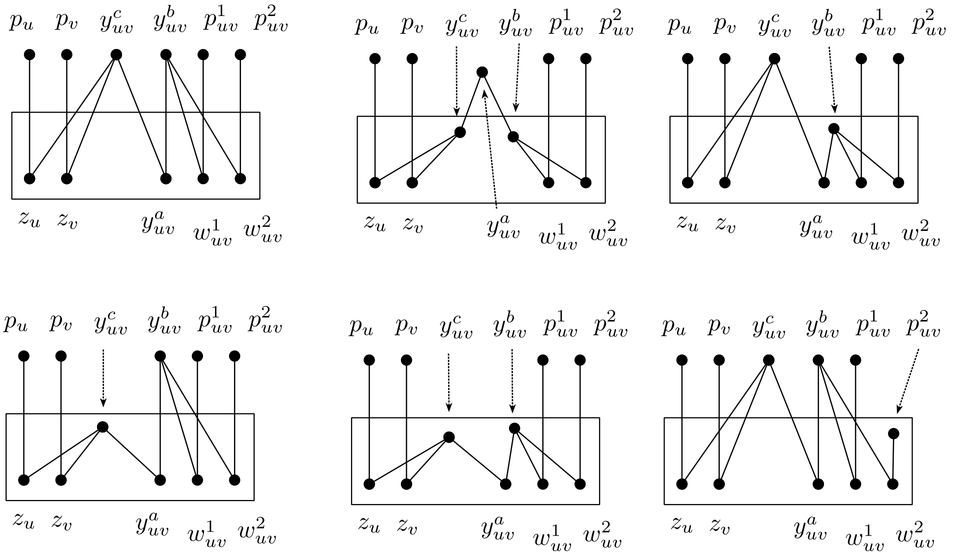


Figure 2 The top-left figure illustrates an encoding of edge uv in G while reducing from an instance of Edge Induced Forest to an instance of Contraction(vc). The remaining five figures correspond to the partition of Y_s mentioned in the proof of Lemma 11.

We now present a parameter preserving reduction from Edge Induced Forest to Contraction(vc), which takes an instance (G, ℓ) of Edge Induced Forest and returns an instance (G', k, d) of Contraction(vc). It constructs a graph G' from G as follows:

- Initialize $V(G') = E(G') = \emptyset$.
- For every vertex u in $V(G)$, add two vertices z_u, p_u to $V(G')$ and the edge $z_u p_u$ to $E(G')$.
- For every edge uv in $E(G)$, add the vertex set $\{y^a_{uv}, y^b_{uv}, y^c_{uv}, w^1_{uv}, w^2_{uv}, p^1_{uv}, p^2_{uv}\}$ to $V(G')$. Add edges $\{z_u y^c_{uv}, z_v y^c_{uv}\}$ to $E(G')$. These edges encode adjacency relations in G. Add also the edges $\{y^a_{uv} y^b_{uv}, y^a_{uv} y^c_{uv}, y^b_{uv} w^1_{uv}, y^b_{uv} w^2_{uv}, w^1_{uv} p^1_{uv}, w^2_{uv} p^2_{uv}\}$ to $E(G')$. These edges are part of a gadget which is private to edge uv.

This completes the construction of G'. The reduction sets $k = 4 \cdot \ell$, $d = 3 \cdot \ell$, and returns (G', k, d) as the constructed instance. Note that, indeed, $k < \mathsf{rank}(G')$ and $d \le k < 2d$ (more precisely, $k - d = \frac{d}{3}$). See Figure 2 for an illustration. Before proving the correctness of the reduction, we first note some properties of the graph G'. We define the following sets:

- $Z := \{z_u \in V(G') \mid u \in V(G)\}$,
- $Y^{abc} := Y^a \cup Y^b \cup Y^c$ where $Y^a := \{y^a_{uv} \in V(G') \mid uv \in E(G)\}$, $Y^b := \{y^b_{uv} \in V(G') \mid uv \in E(G)\}$, and $Y^c := \{y^c_{uv} \in V(G') \mid uv \in E(G)\}$,
- $W := \{w^1_{uv}, w^2_{uv} \in V(G') \mid uv \in E(G)\}$, and
- $P := \{p_u \in V(G') \mid u \in V(G)\} \cup \{p^1_{uv}, p^2_{uv} \mid uv \in E(G)\}$.

Note that $\langle Z, Y^{abc}, W, P\rangle$ is a partition of $V(G')$ where each vertex in P is a pendant vertex and each vertex in $Z \cup W$ is adjacent to a pendant vertex in P. Moreover, $\mathsf{rank}(G') > k$. Note that $X := Z \cup W \cup Y^a$ is an independent set in G'. In the next lemma, we argue that it is also a minimum vertex cover of G', which implies that, as claimed in the statement of Theorem 2, G' is a bipartite graph with a bipartition $\langle X, Y\rangle$ such that X is a minimum vertex cover of G'.

▶ **Lemma 9** ($\star$)**.** *The set $X = Z \cup W \cup Y^a$ is a minimum vertex cover of G'.*

The following lemma corresponds to the "easy" direction of the reduction.

▶ **Lemma 10** ($\star$). *If* (G, ℓ) *is a* Yes*-instance of* Edge Induced Forest*, then* (G', k, d) *is a* Yes*-instance of* Contraction(vc)*.*

We now present a brief overview of the proof of the correctness in the backward direction, corresponding to Lemma 11, whose full proof can be found in the full version. By Lemma 6, there is a solution F of (G', k, d) if and only if there exists a solution pair $\langle X_s, Y_s\rangle$ such that (i) $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ is a vertex cover of G', (ii) $\mathsf{rank}(X_{\mathsf{el}}) \geq |F| = k = 4 \cdot \ell$, and (iii) $|Y_s| - |X_s| \leq k - d = \ell$. Note that as X and $Y = V(G') \setminus X$ are independent sets in G', every edge in $E(X_{\mathsf{el}})$ is incident on exactly one vertex in Y_s. We can interpret the second condition as a *value function* and the third condition as a *cost function*. In other words, our objective is to find sets X_s, Y_s such that their cost, i.e., $|Y_s| - |X_s|$, is at most ℓ whereas their value, i.e., the rank of edges in $E(X_{\mathsf{el}})$ that are incident on Y_s, is at least $4 \cdot \ell$. Lemma 7 implies that the vertices of the form z_u, w^1_{uv}, and w^2_{uv} are in X_{el}. The first condition implies that only the five configurations shown in Figure 2 are possible (the top-left is *not* a configuration). Starting from top-middle and moving row-wise, the individual value and cost of these configurations are $(4, 1)$, $(3, 1)$, $(3, 1)$, $(6, 2)$, and $(1, 1)$, respectively. To meet both the value and budget constraints, every vertex in X_s, Y_s needs to be the of first type. This implies there are ℓ vertices in X_s that are of the form y^a_{uv}, and Y_s contains the corresponding vertices of the form y^b_{uv} and y^c_{uv}. We argue that the edges corresponding to vertices in Y^c_{uv} form a solution of (G, ℓ) and formalize these ideas in the next lemma.

▶ **Lemma 11** ($\star$). *If* (G', k, d) *is a* Yes*-instance of* Contraction(vc)*, then* (G, ℓ) *is a* Yes*-instance of* Edge Induced Forest*.*

We are ready to present the proof of Theorem 2.

Proof of Theorem 2. Consider the reduction presented in this subsection. Lemma 10 and Lemma 11 imply that the reduction is safe. By the description of the reduction, it outputs the constructed instance in polynomial time. The W[1]-hardness of Contraction(vc) follows from Theorem 8. As $k = 4 \cdot \ell$ and $d = 3 \cdot \ell$, if Contraction(vc) admits an algorithm with running time $f(k + d) \cdot n^{o(k+d)}$, then Edge Induced Forest also admits an algorithm with running time $f(\ell) \cdot n^{o(\ell)}$, which contradicts Theorem 8. ◀

4 Algorithm for Contraction(vc)

In this section we present the main ideas of the proof of Theorem 3, following the sketch provided in Section 1 and summarized in Figure 1. Due to space limitations, most of the technical content can be found in the full version Namely, we present an algorithm that takes as input an instance (G, k, d) of Contraction(vc) and returns either Yes or No, and whose high-level description is as follows (cf. Figure 1):

- If $k = \mathsf{rank}(G)$, then it uses the algorithm mentioned in Lemma 12.
- If $k < \mathsf{rank}(G)$ and $2d \leq k$, then it uses the algorithm mentioned in Lemma 13.
- If $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, then it uses the algorithm mentioned in Lemma 14.

Note that, since we can safely assume that $d \leq k \leq \mathsf{rank}(G)$, the above three cases are exhaustive. The first two cases turn out to be quite easy to handle.

▶ **Lemma 12** ($\star$). *There exists an algorithm that, given as input an instance* (G, k, d) *of* Contraction(vc) *with a guarantee that* $k = \mathsf{rank}(G)$*, runs in time* $1.2738^d \cdot n^{\mathcal{O}(1)}$*, and correctly determines whether it is a* Yes*-instance.*

▶ **Lemma 13** (⋆). *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction(vc) *with guarantees that $k < \mathsf{rank}(G)$ and $2d \leq k$, runs in time $1.2738^d \cdot n^{\mathcal{O}(1)}$, and correctly determines whether it is a* Yes-*instance.*

We may assume that G is a connected graph; we justify this assumption in the full version. In order to deal with the third case above, our objective is to prove the following lemma.

▶ **Lemma 14.** *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction(vc) *with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, runs in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$, and correctly determines whether it is a* Yes-*instance.*

We start with the following result, which will allow us to assume henceforth that we are equipped with a minimum vertex cover of the input graph with small rank.

▶ **Lemma 15** (⋆). *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction(vc) *with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, runs in time $2.6181^k \cdot n^{\mathcal{O}(1)}$, and either correctly concludes that (G, k, d) is a* Yes-*instance, or computes a minimum vertex cover X of G such that $\mathsf{rank}(\mathrm{X}) < d$.*

Our next step is to provide an FPT-reduction from Contraction(vc) to the Annotated Contraction(vc) problem, defined as follows.

Annotated Contraction(vc)
Input: An instance (G, k, d) of Contraction(vc), a minimum vertex cover X of G, and a tuple $\langle X_L, X_R \rangle$ such that X_L, X_R are disjoint subsets of X.
Question: Do there exist sets $X_s \subseteq X$ and $Y_s \subseteq Y$ $(= V(G) \setminus X)$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k$, (iii) $|Y_s| - |X_s| \leq k - d$, and (iv) $X_L \cap X_s = \emptyset$ and $X_R \subseteq X_s$?

The first three conditions correspond to the three conditions mentioned in Lemma 6. We remark that there is a small technical caveat while using Lemma 6. Consider an instance (G, k, d) of Contraction(vc), and let F be a solution. Lemma 6 implies that there are subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq |F|$, and (iii) $|Y_s| - |X_s| \leq |F| - d$. However, the statement of Annotated Contraction(vc) specifies the integer k and not the actual size of a minimum solution F. For example, if there exists a solution F of size, say, $k/2$, then Lemma 6 ensures that $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k/2$, however $\mathsf{rank}((X \setminus X_s) \cup Y_s)$ can be smaller than k. To overcome this, we assume that $(G, k-1, d)$ is a No-instance of Contraction(vc). This implies that if there is a subset F of $E(G)$ of size *at most* k such that $\mathsf{vc}(G/F) \leq \mathsf{vc}(G) - d$, then F is of size *exactly* k. We summarize below all the assumptions on the input instance.

▶ **Guarantee 4.1.** *Consider an instance (G, k, d) of* Contraction(vc) *that satisfies the following conditions.*

- *G is a connected graph, $k < \mathsf{rank}(G)$, and $d \leq k$.*
- *A minimum vertex cover X of G is provided as an additional part of the input.*
- $\mathsf{rank}(X) < d$.
- *$(G, k-1, d)$ is a* No-*instance of* Contraction(vc).

Unless stated otherwise, we denote the independent set $V(G) \setminus X$ by Y.

Consider an instance (G, k, d) of Contraction(vc) with Guarantee 4.1. Using Lemma 6, we construct $2^{\mathcal{O}(d)}$ many instances of Annotated Contraction(vc) such that (G, k, d) is a Yes-instance if and only if at least one of these newly created instances is a Yes-instance.

Informally, let F be the set of edges in a spanning forest of $G[X]$. As $\mathsf{rank}(X) < d$, we have $|F| < d$. We iterate over all "valid" partitions $\langle X_L, X_R \rangle$ of $V(F)$. We construct an instance of Annotated Contraction(vc) for each such a partition. We formalize this intuition and prove its correctness in the following lemma.

▶ **Lemma 16** ($\star$). *Suppose that there is an algorithm that solves* Annotated Contraction(vc) *in time $f(n, k, d)$. Then, there exists an algorithm that given as input an instance (G, k, d) of* Contraction(vc) *with Guarantee 4.1, runs in time $3^d \cdot n^{\mathcal{O}(1)} \cdot f(n, k, d)$, and correctly determines whether it is a* Yes*-instance.*

To solve an instance of Annotated Contraction(vc), we reduce it to an equivalent instance of the Constrained MaxCut problem. To present such a reduction, it is convenient to work with an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Annotated Contraction(vc) where X is an independent set. This is guaranteed by the following reduction rule.

▶ **Reduction Rule 4.1.** *Let $((G, k, d), X, \langle X_L, X_R \rangle)$ be an instance of* Annotated Contraction(vc)*, $F_1 = E(X_L, X_R)$, and F_2 be the set of edges in a spanning forest of $G[X_L]$.*

- *Delete the edges in F_1.*
- *Contract the edges in F_2 and reduce both k and d by $|F_2|$.*

Return the instance $((G', k', d'), X', \langle X'_L, X_R \rangle)$ where $G' = (G - F_1)/F_2$, $k' = k - |F_2|$, $d' = d - |F_2|$, $X' = V(G[X]/F_2)$, and $X'_L = V(G[X_L]/F_2)$.

▶ **Lemma 17** ($\star$). *Reduction Rule 4.1 is safe. Therefore, it is safe to assume that we are given an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of* Annotated Contraction(vc) *such that X is an independent set and a minimum vertex cover of G.*

We find the following reformulation of Annotated Contraction(vc) convenient to present an algorithm to solve it.

Constrained MaxCut
Input: An instance (G, k, d) of Contraction(vc), a minimum vertex cover X of G, and a tuple $\langle X_L, X_R \rangle$ such that X_L, X_R are disjoint subsets of X.
Question: Does there exist a partition $\langle V_L, V_R \rangle$ of $V(G)$ such that (i) $E(V_L \cap Y, V_R \cap X) = \emptyset$, (ii) $\mathsf{rank}(E(V_L \cap X, V_R \cap Y)) \geq k$, (iii) $|V_R \cap Y| - |V_R \cap X| \leq k - d$, and (iv) $X_L \subseteq V_L$ and $X_R \subseteq V_R$?

Note that in Annotated Contraction(vc) we are seeking for a pair of subsets, whereas in Constrained MaxCut we are looking for a partition of $V(G)$. Such a formulation allows us to handle vertices that we have decided to keep out of a solution pair. Note that the input instances for both of these problems are the same. Hence, due to Lemma 17, it is safe to assume that X is a minimum vertex cover and an independent set in G. In the next lemma we show that both problems are in fact equivalent.

▶ **Lemma 18** ($\star$). *An instance $((G, k, d), X, \langle X_L, X_R \rangle)$ is a* Yes*-instance of* Annotated Contraction(vc) *if and only if it is a* Yes*-instance of* Constrained MaxCut.

Consider an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Constrained MaxCut. We consider the following two cases: (1) $k = d$, and (2) $d < k < 2d$. (Recall that we are in the case where $k < 2d$.) The first case, as we will see, allows us to impose additional restrictions on the vertices that are in V_R. It also helps us to set up some conditions such that, if they are satisfied while running the algorithm, then it can terminate and safely conclude that the input is a Yes-instance. In the second case, even for $k = d + 1$, we do not have these privileges. We deal with each of the two cases separately. Namely, Lemma 19 states that if an input instance is of the second type, then we can construct a collection of $2^{\mathcal{O}(d)} \cdot n^{k-d}$ many instances of the first type such that the input instance is a Yes-instance if and only if at least one of these newly created instances is a Yes-instance. We remark that this is the only place, in the whole algorithm, where an n^{k-d}-factor appears in the running time. Recall that Theorem 2 implies that this factor is unavoidable. After this lemma we present an algorithm to solve the instances that are of the first type.

▶ **Lemma 19** ($\star$). *Suppose that there is an algorithm that, given an instance* $((G,k,d),X,\langle X_L,X_R\rangle)$ *of* Constrained MaxCut *with a guarantee that* $k = d$*, runs in time* $f(n,k,d)$ *and correctly determines whether it is a* Yes*-instance. Then, there is an algorithm that solves* Constrained MaxCut *in time* $f(n,k,d) \cdot 2^{\mathcal{O}(d)} \cdot n^{k-d+1}$.

We proceed to present an algorithm to solve an instance $((G,k,d),X,\langle X_L,X_R\rangle)$ of Constrained MaxCut with a guarantee that $k = d$. We first present a reduction rule to simplify these instances under the presence of a matching saturating X.

▶ **Reduction Rule 4.2.** *Consider an instance* $((G,k,d),X,\langle X_L,X_R\rangle)$ *of* Constrained MaxCut *such that* $k = d$ *and* X *is an independent set in* G*. Let* M *be a matching in* G *saturating* X*.*

- *If there exists* $x \in X \setminus X_L$ *such that* $N(x) \setminus V(M) \neq \emptyset$*, then add* x *to* X_L*.*
- *If there exists* $x \in X_L$ *such that* $N(x) \setminus V(M) \neq \emptyset$*, then delete all vertices in* $N(x) \setminus V(M)$*.*

Return instance $((G',k,d),X,\langle X'_L,X_R\rangle)$ *where* $G' = G - (N(x) \setminus V(M))$ *and* $X'_L = X_L \cup \{x\}$*.*

▶ **Lemma 20** ($\star$). *Reduction Rule 4.2 is safe.*

The full version contains an informal description of the algorithm for Constrained MaxCut with a guarantee that $k = d$. We now briefly describe the algorithm formally.

We consider directed graphs that can have parallel arcs. We define the *rank* of a digraph, and the rank of a subset of its vertices or arcs using its underlying undirected graph.

Constrained Digraph MaxCut
Input: A digraph D, a tuple $\langle X_L,X_R\rangle$ of disjoint subsets of X, and an integer k.
Question: Does there exist a partition (V_L,V_R) of $V(G)$ such that (i) $A(V_R,V_L) = \emptyset$, (ii) $\mathsf{rank}(A(V_L,V_R)) \geq k$, and (iii) $X_L \subseteq V_L$ and $X_R \subseteq V_R$?

We say that a partition $\langle V_L,V_R\rangle$ is a *solution* of $(D,\langle X_L,X_R\rangle,k)$ if it satisfies all the three conditions in the statement of the problem. We present a reduction that, given an instance $((G,k,d),X,\langle X_L,X_R\rangle)$ of Constrained MaxCut, returns an instance $(D,\langle X_L,X_R\rangle,k)$ of Constrained Directed MaxCut. The reduction takes as input an instance $((G,k,d),X,\langle X_L,X_R\rangle)$ of Constrained MaxCut on which Reduction Rule 4.2 is not applicable. It starts with a copy of the graph G and constructs a digraph D. The reduction finds (in polynomial time) a matching M in G that saturates all vertices in X. For every $xy \in E(G)$, where $x \in X$ and $y \in Y$, it deletes edge xy and adds arc (x,y) (i.e., it directs edges from X to Y). For every arc (x,y) in M, it adds arc (x_1,x) for every in-neighbour x_1 of y, and then deletes vertex y. This completes the construction of D. The reduction returns $(D,\langle X_L,X_R\rangle,k)$ as the instance of Constrained Digraph MaxCut.

▶ **Lemma 21** ($\star$). $((G,k,d),X,\langle X_L,X_R\rangle)$ *is a* Yes*-instance of* Constrained MaxCut *if and only if* $(D,\langle X_L,X_R\rangle,k)$ *is a* Yes*-instance of* Constrained Digraph MaxCut*.*

We present a dynamic programming algorithm for Constrained Directed MaxCut.

▶ **Lemma 22** ($\star$)**.** *There is an algorithm that, given an instance* $(G, \langle X_L, X_R\rangle, k)$ *of* Constrained Directed MaxCut*, runs in time* $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *and correctly determines whether it is a* Yes*-instance.*

We finally have all the ingredients to prove Lemma 14, and consequently Theorem 3 as well. The proofs are provided in the full version.

5 Conclusion

In this article we considered the problem of reducing the size of a minimum vertex cover of a graph G by at least d using at most k edge contractions. Note that the problem is trivial when $k < d$. A few simple observations prove that when $d \leq 2k$, the problem is coNP-hard and FPT when parameterized by $k + d$. Almost all of our technical work is to handle the case when $d \leq k < 2d$. We proved that the problem is NP-hard when $k = d + \frac{\ell-1}{\ell+3} \cdot d$ for any integer $\ell \geq 1$ such that k is an integer (in particular, $\ell = 1$). This implies that the problem is hard for various values of $k - d$ in the set $\{0, 1, \ldots, d - 1\}$. We were able to prove that if $(k - d)$ is a constant then the problem is FPT when parameterized by $k + d$. However, if no such a condition is imposed, then the problem is W[1]-hard. More precisely, we presented an algorithm with running time $2^d \cdot n^{k-d+\mathcal{O}(1)}$ and proved that the problem is W[1]-hard when parameterized by $k + d$ in the case where $k - d = \frac{d}{3}$ (see the proof of Theorem 2).

We believe that it should be possible to prove that the problem is NP-hard for *every* value of $k - d$ in the set $\{0, 1, \ldots, d - 1\}$. Such a reduction has the potential to sharpen the distinction between FPT and W[1]-hard cases as $k - d$ varies in this range. It might also simplify the analysis of our XP algorithm or lead to a simpler algorithm. It would be interesting to analyze the parameterized complexity of the problem with respect to structural parameters like the vertex cover number or the treewidth of the input graph. Note that the problem is trivially FPT when parameterized by the vertex cover number. Finally, it is worth mentioning that we did not focus on optimizing the degree of the polynomial term $n^{\mathcal{O}(1)}$ in our XP algorithm, although it is reasonably small.

References

1 Hans L. Bodlaender, Pinar Heggernes, and Daniel Lokshtanov. Graph Modification Problems (Dagstuhl Seminar 14071). *Dagstuhl Reports*, 4(2):38–59, 2014. doi:10.4230/DagRep.4.2.38.

2 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

3 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification, 2020. arXiv:2001.06867.

4 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.

5 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. URL: https://dblp.org/rec/books/daglib/0030488.bib.

6 Öznur Yaşar Diner, Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Contraction and deletion blockers for perfect graphs and H-free graphs. *Theoretical Computer Science*, 746:49–72, 2018. doi:10.1016/j.tcs.2018.06.023.

7 Esther Galby, Paloma T. Lima, Felix Mann, and Bernard Ries. Using edge contractions to reduce the semitotal domination number, 2021. arXiv:2107.03755.

8 Esther Galby, Paloma T. Lima, and Bernard Ries. Reducing the domination number of graphs via edge contractions and vertex deletions. *Discrete Mathematics*, 344(1):112169, 2021. doi:10.1016/j.disc.2020.112169.

9 Esther Galby, Felix Mann, and Bernard Ries. Blocking total dominating sets via edge contractions. *Theoretical Computer Science*, 877:18–35, 2021. doi:10.1016/j.tcs.2021.03.028.

10 Esther Galby, Felix Mann, and Bernard Ries. Reducing the domination number of (P_3+kP_2)-free graphs via one edge contraction. *Discrete Applied Mathematics*, 305:205–210, 2021. doi:10.1016/j.dam.2021.09.009.

11 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

12 Paloma T. Lima, Vinícius Fernandes dos Santos, Ignasi Sau, and Uéverton S. Souza. Reducing graph transversals via edge contractions. *Journal of Computer and System Sciences*, 120:62–74, 2021. doi:10.1016/j.jcss.2021.03.003.

13 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Critical vertices and edges in H-free graphs. *Discrete Applied Mathematics*, 257:361–367, 2019. doi:10.1016/j.dam.2018.08.016.

14 Saket Saurabh, Uéverton dos Santos Souza, and Prafullkumar Tale. On the parameterized complexity of grid contraction. In *Proc. of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 162 of *LIPIcs*, pages 34:1–34:17, 2020. doi:10.4230/LIPIcs.SWAT.2020.34.

15 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the NP-hardness of edge-deletion and -contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983. doi:10.1016/0166-218X(83)90101-4.

16 Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978. doi:10.1145/800133.804355.

Generalized Bundled Fragments for First-Order Modal Logic

Mo Liu
LORIA, Unviersity of Lorraine, Nancy, France

Anantha Padmanabha
DI ENS, École Normale Supérieure, Université PSL, CNRS, Inria, Paris, France

R. Ramanujam
Institute of Mathematical Sciences, HBNI, Chennai, India (Retired)
Azim Premji University, Bengaluru (Visiting)

Yanjing Wang
Department of Philosophy, Peking University, Beijing, China

Abstract

When we bundle quantifiers and modalities together (as in $\exists x\Box$, $\Diamond\forall x$ etc.) in first-order modal logic (FOML), we get new logical operators whose combinations produce interesting *bundled* fragments of FOML. It is well-known that finding decidable fragments of FOML is hard, but existing work shows that certain bundled fragments are decidable [14], without any restriction on the arity of predicates, the number of variables, or the modal scope. In this paper, we explore generalized bundles such as $\forall x\forall y\Box, \forall x\exists y\Diamond$ etc., and map the terrain with regard to decidability, presenting both decidability and undecidability results. In particular, we propose the loosely bundled fragment, which is decidable over increasing domains and encompasses all known decidable bundled fragments.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Logic and verification

Keywords and phrases bundled fragments, first-order modal logic, decidability, tableaux

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.70

Related Version *Full Version*: `https://arxiv.org/abs/2202.01581`

Funding This work is supported by NSSF grant 19BZX135.
Anantha Padmanabha: The author is funded by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

Acknowledgements The authors thank the anonymous reviewers of MFCS2022 for their comments that improved the presentation of the paper.

1 Introduction

While propositional modal logic (ML) has had extensive applications in system verification and artificial intelligence, and first-order logic (FOL) in finite model theory and database theory, first-order modal logic (FOML) has been studied much less, as it seems to combine the worst of both computationally, leading to undecidability. FOML is a natural specification language for state transition systems where states are given by first-order descriptions of computational domains, with applicability in the realm of database updates, in the control of infinite-state systems, networks with unbounded parallelism and cryptographic protocols. This motivates the study of *decidable fragments* of FOML.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 70; pp. 70:1–70:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This is a challenge, since even the two-variable fragment of FOML with one unary predicate is undecidable over almost all useful model classes [15]. This situation is to be contrasted against the robust decidability of propositional modal logics ([16, 1, 7]), and the many decidable fragments of first-order logic ([4]).

Despite such discouragement, there have been a few successful attempts: for instance, the *monodic restriction*, mandating only one free variable in the scope of any modal subformula, yields decidability when combined with a decidable fragment of FOL ([8, 19]). This idea, arising originally from description logics (cf. [9]) has led to applications in temporal and epistemic logics ([6], [10], [2, 3]).

Rather than placing a restriction on variables, or on *quantification scope* as in the case of guarded fragments [1], Wang ([17]) suggested a fragment in which the existential quantifier and the box modality were always bundled together to appear as a single quantifier-modality pair ($\exists x\Box$). The resulting fragment of FOML enjoys many attractive properties: finite tree model property, PSpace decision procedure and a simple axiom system, without any restriction on predicates or the occurrences of variables. The new operator $\exists x\Box$ captures the logical structure of various *knowing-wh* expressions such as *knowing what, knowing how, knowing why*, and so on (cf. [18]), e.g., *knowing how to achieve* φ can be rendered as *there is* a plan x such that the agent *knows that* x can be executed and will guarantee φ.

In [14], we took the next step by considering not only the combination $\exists x\Box$ but also its companion $\forall x\Box$: the logic with both of these combinations continued to be decidable (over increasing domain models). Such modal-quantifier combinations were thus called *bundled fragments* of FOML. Over models where the domain remains the same in all states, the fragment with both $\exists x\Box$ and $\forall x\Box$ is undecidable, while the $\exists x\Box$-fragment is still decidable.

Clearly, we can define more bundles such as $\Box\forall x$, $\Box\exists x$, etc., and indeed further combinations such as $\forall x\forall y\Box$, and combinations thereof. These (generalized) bundled fragments offer us many interesting possibilities for system specification:

- $\neg\exists x\Box\ (x < c)$: No element is guaranteed to be bounded by constant c (after update).
- $\exists x\Box\ \Box\exists y\ (x > y)$: There is an element that dominates some element after every update.
- $\Box\exists x\ \Box\forall y\ (x \leq y)$: All updates admit a local minimum.
- $\exists x\Box\ (\exists y\Box\ (x > y) \land \exists y\Box\ (x < y))$: There is an element that dominates another no matter the update and is dominated by another no matter the update.

Computationally this raises the natural question of what is the most general *bundled fragment* that is decidable. This is the project taken up in this paper. We consider all possible combinations of the bundled formulas and classify their decidability status. The classification is described in Table 1. We provide a trichotomy: decidable fragments, undecidable fragments and fragments that do not have a finite model property (but where decidability is open).

Towards proving the trichotomy, we first define the notion of *loosely bundled fragment* which subsumes many decidable bundled fragments of FOML and prove its decidability via a tableau method. We also prove the decidability of another combination of bundled operators where we allow only formulas of the form $\forall x\Box + \Box\forall x + \Box\exists x$ (but is not included in the loosely bundled fragment). This requires us to introduce a new proof technique that helps us switch quantifiers in a specific context.

Due to space restrictions, we present only the main ideas and proof techniques of the decidable fragments over increasing domain models in the paper. The proof details, as well as undecidability and lack of finite model property for the other fragments can be found in the detailed technical report in the arXiv [13].

Table 1 Satisfiability problem classification for combinations of bundled fragments over increasing domain models. A “$*$” means no matter the corresponding bundle is included or not.

$\forall\Box$	$\exists\Box$	$\Box\forall$	$\Box\exists$	Decidability
✓	✗	✗	✗	Subsumed by the Loosely Bundled Fragment
✗	✓	✗	✗	
✗	✗	✓	✗	
✗	✗	✗	✓	
✓	✓	✗	✗	
✗	✗	✓	✓	
$*$	✓	✓	$*$	Undecidable
✗	✓	✗	✓	No Finite Model Property
✓	✓	✗	✓	Undecidable
✓	✗	✓	✓	ExpSpace
Loosely Bundled Fragment				ExpSpace

2 Syntax and Semantics

The syntax of first-order modal logic is given by extending the first-order logic with modal operators. Note that we exclude equality, constants and function symbols from the syntax.

▶ **Definition 1** (FOML syntax). *Given a countable set of predicates $\mathcal{P}$ and a countable set of variables* Var*, the syntax of* FOML *is given by:*

$$\alpha ::= P(x_1, \ldots, x_n) \mid \neg\alpha \mid \alpha \wedge \alpha \mid \exists x\alpha \mid \Box\alpha$$

where $P \in \mathcal{P}$ has arity n and $x, x_1, \ldots, x_n \in$ Var.

The boolean connectives $\vee, \rightarrow, \leftrightarrow$, and the modal operator $\Diamond$ which is the dual of $\Box$, and the quantifier $\forall$ are all defined in the standard way. The notion of free variables, denoted by $\mathsf{FV}(\alpha)$ is similar to what we have for first-order logic with $\mathsf{FV}(\Box\alpha) = \mathsf{FV}(\alpha)$. We write $\alpha(x)$ to mean that x occurs as a free variable of α. Also, $\alpha[y/x]$ denotes the formula obtained from α by replacing every free occurrence of x by y.

▶ **Definition 2** (FOML structure). *An increasing domain model for* FOML *is a tuple $\mathcal{M} = (\mathcal{W}, \mathcal{D}, \delta, \mathcal{R}, \rho)$ where $\mathcal{W}$ is a non-empty countable set called* worlds*;*[1] *$\mathcal{D}$ is a non-empty countable set called* domain*; $\mathcal{R} \subseteq (\mathcal{W} \times \mathcal{W})$ is the* accessibility relation*. The map $\delta : \mathcal{W} \mapsto 2^{\mathcal{D}}$ assigns to each $w \in \mathcal{W}$ a non-empty local domain set such that whenever $(w, v) \in \mathcal{R}$ we have $\delta(w) \subseteq \delta(v)$ and $\rho : (\mathcal{W} \times \mathcal{P}) \mapsto \bigcup_n 2^{\mathcal{D}^n}$ is the* valuation function*, which specifies the interpretation of predicates at every world over the local domain with appropriate arity. The model $\mathcal{M}$ is said to be a* constant domain model *if for all $w \in \mathcal{W}$ we have $\delta(w) = \mathcal{D}$.*

The monotonicity condition requiring $\delta(w) \subseteq \delta(v)$ for $(w, v) \in \mathcal{R}$ is required for evaluating the free variables present in the formula [11]. Because of this, the models are called *increasing domain models.*

For a given model $\mathcal{M}$ we denote $\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}$ etc to indicate the corresponding components. We simply use $\mathcal{W}, \mathcal{R}, \delta$ etc when $\mathcal{M}$ is clear from the context.

[1] Note that FOML can be translated into two-sorted FOL, and due to the Löwenheim–Skolem theorem for countable languages, every model has an equivalent countable model, cf. [5].

To evaluate formulas, we need an assignment function for variables. For a given model $\mathcal{M}$, an assignment function $\sigma : \mathsf{Var} \mapsto \mathcal{D}$ is *relevant* at $w \in \mathcal{W}$ if $\sigma(x) \in \delta(w)$ for all $x \in \mathsf{Var}$.

▶ **Definition 3** (FOML semantics). *Given an* FOML *model* $\mathcal{M} = (\mathcal{W}, \mathcal{D}, \delta, \mathcal{R}, \rho)$ *and* $w \in \mathcal{W}$, *and* σ *relevant at* w, *for all* FOML *formulas* α *define* $\mathcal{M}, w, \sigma \models \alpha$ *inductively as follows:*

$\mathcal{M}, w, \sigma \models P(x_1, \ldots, x_n)$	$\Leftrightarrow$	$(\sigma(x_1), \ldots, \sigma(x_n)) \in \rho(w, P)$
$\mathcal{M}, w, \sigma \models \neg\alpha$	$\Leftrightarrow$	$\mathcal{M}, w, \sigma \not\models \alpha$
$\mathcal{M}, w, \sigma \models \alpha \wedge \beta$	$\Leftrightarrow$	$\mathcal{M}, w, \sigma \models \alpha$ *and* $\mathcal{M}, w, \sigma \models \beta$
$\mathcal{M}, w, \sigma \models \exists x\alpha$	$\Leftrightarrow$	*there is some* $d \in \delta(w)$ *such that* $\mathcal{M}, w, \sigma_{[x \mapsto d]} \models \alpha$
$\mathcal{M}, w, \sigma \models \Box\alpha$	$\Leftrightarrow$	*for every* $u \in \mathcal{W}$ *if* $(w, u) \in \mathcal{R}$ *then* $\mathcal{M}, u, \sigma \models \alpha$

We sometimes write $\mathcal{M}, w \models \alpha(a)$ to mean $\mathcal{M}, w, [x \mapsto a] \models \alpha(x)$.

A formula α is *satisfiable* if there is some FOML structure $\mathcal{M}$ and $w \in \mathcal{W}$ and some assignment σ relevant at w such that $\mathcal{M}, w, \sigma \models \alpha$. In the sequel, we will only talk about the relevant σ for a given pointed model. Also, while evaluating α, it is enough to consider σ to be a partial function that gives an interpretation for the free variables of α. A formula α is *valid* if $\neg\alpha$ is not satisfiable.

2.1 Bundled fragments

The motivation for "bundling" is to restrict the occurrences of quantifiers using modalities. For instance, allowing only formulas of the form $\forall x \Box \alpha$ is one such bundling. We could also have $\Diamond \exists y \alpha$. Thus, there are many ways to "bundle" the quantifiers and modalities. We call these the "bundled operators/modalities". The following syntax defines all possible bundled operators of one quantifier and one modality:

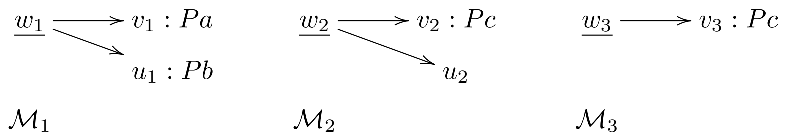
▶ **Definition 4** (Bundled-FOML syntax). *The bundled fragment of* FOML *is the set of all formulas constructed by the following syntax:*

$$\alpha ::= P(x_1, \ldots, x_n) \mid \neg\alpha \mid \alpha \wedge \alpha \mid \Box\alpha \mid \exists x \Box\alpha \mid \forall x \Box\alpha \mid \Box\exists x\alpha \mid \Box\forall x\alpha$$

where $P \in \mathcal{P}$ *has arity* n *and* $x, x_1, \ldots, x_n \in$ *Var.*

Note that the duals of the bundled operators give us the formulas of the form $\forall x \Diamond \alpha$, $\exists x \Diamond \alpha$, $\Diamond \forall x \alpha$, $\Diamond \exists x \alpha$. Also, note that $\Box\alpha$ can be defined using any one of the bundled operators where the quantifier is applied to a variable that does not occur in α. However, we retain $\Box\alpha$ in the syntax for technical convenience.

The following *constant domain* models may help to get familiar with bundles.



Let $\mathcal{D}^{\mathcal{M}_1} = \{a, b\}$, $\mathcal{D}^{\mathcal{M}_2} = \mathcal{D}^{\mathcal{M}_3} = \{c\}$. $\Box\exists x Px$ holds at w_1 and w_3 but not at w_2; $\exists x \Box Px$ holds only at w_3; $\neg\forall x \Box Px$ holds at w_1 and w_2; $\neg\Box\forall x \neg Px$ holds at all the w_i.

We denote $\forall\Box$-fragment to be the language that allows only atomic formulas, negation, conjunction, $\Box\alpha$ and $\forall x \Box\alpha$ (dually $\exists x \Diamond \alpha$) formulas, similarly for $\exists\Box$-fragment and so on. In general, these fragments are not equally expressive, e.g., as shown by [17], the $\exists\Box$-fragment cannot express $\Box\exists$, $\forall\Box$ and $\Box\forall$ bundles over models with increasing (or constant) domains.

In [14, 12] we proved that the $\forall\Box + \exists\Box$ and $\Box\forall + \Box\exists$ fragments are decidable over increasing domain models, while $\forall\Box$-fragment and $\Box\forall$-fragment are undecidable over constant domain models. Since simple bundles become undecidable over constant domain models, in this paper we focus on generalized bundles over increasing domain models.

Note that we can have more general bundled operators of the form $\forall x \forall y \Box \alpha$ etc. This naturally raises the question of what is the most general fragment of this form that is decidable. Towards this, we define a general fragment that subsumes all known decidable bundled fragments so far.

2.2 The Loosely Bundled Fragment

Note that a bundled formula of the form $\exists x \Box \alpha$ imposes a restriction that there is exactly one modal formula in the scope of $\exists x$. But this is a strong requirement. We weaken this condition to allow formulas of the form $\exists x \beta$ where β is a boolean combination of atomic formulas and modal formulas. Moreover, we can allow a quantifier alternation of the form $\exists x_1 \cdots \exists x_n \, \forall y_1 \cdots \forall y_m \, \beta$. As we will see, the fact that the existential quantifiers are outside the scope of universal quantifiers can help us to obtain decidability results over increasing domain models.

▶ **Definition 5** (LBF syntax). *The loosely bundled fragment of* FOML *is the set of all formulas constructed by the following syntax:*

$$\begin{aligned}\psi ::= & P(z_1,\dots z_n) \mid \neg P(z_1,\dots z_n) \mid \psi \wedge \psi \mid \psi \vee \psi \mid \Box\alpha \mid \Diamond\alpha \\ \alpha ::= & \psi \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \exists x_1 \dots \exists x_k \forall y_1 \dots \forall y_l \ \psi\end{aligned}$$

where $k, l, n \geq 0$ and $P \in \mathcal{P}$ has arity n and $x_1, \dots x_k$, $y_1, \dots y_l$, $z_1, \dots, z_n \in$ Var.

Let LBF be the set of all formulas that can be obtained from the grammar of α above. Note that the syntax does not allow a quantifier alternation of the form $\forall x \exists y \ \alpha$. Also, inside the scope of quantifier prefix $\exists^*\forall^*$, we can only have boolean combinations of atomic and modal formulas. The $\exists^*\forall^*$ fragment in FO, (Bernays-Schönfinkel-Ramsey class) has a similar quantifier prefix structure but is different in spirit since there is no modality.

The loosely bundled fragment subsumes some of the combinations of bundled fragments. Hence proving the decidability for LBF implies the decidability for these combinations as well.

▶ **Proposition 6.** *The fragments $\forall\Box + \exists\Box$ and $\Box\forall + \Box\exists$ are subfragments of LBF.*

Note that many combinations of bundled operators (for instance $\Box\forall + \exists\Box$) do not form a subfragment of LBF.

3 Tableau Procedure

Note that the formulas of LBF are in negation normal form (where $\neg$ appears only in front of atomic formulas). We first define some useful terms and notations.

▶ **Definition 7.** *For any* FOML *formula φ:*

- *φ is a literal if φ is of the form $P(x_1,\dots x_n)$ or of the form $\neg P(x_1,\dots x_n)$*
- *φ is a module if φ is a literal or φ is of the form $\Delta\alpha$ where $\Delta \in \{\Box, \Diamond\}$*
- *The component of φ is defined inductively as follows:*
  - *If φ is a module then $C(\varphi) = \{\varphi\}$*
  - *If φ is of the form $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ then $C(\varphi) = C(\varphi_1) \cup C(\varphi_2)$*
  - *If φ is of the form $\forall x \ \varphi_1$ or $\exists x \ \varphi_1$ then $C(\varphi) = \{\varphi\} \cup C(\varphi_1)$*
- *A formula φ is called Existential-safe if every $\psi \in C(\varphi)$ is a module or of the form $\forall x \ \psi'$. A finite set of formulas Γ is Existential-safe if every $\varphi \in \Gamma$ is Existential-safe.*

Intuitively, $\mathsf{C}(\varphi)$ is the set of all subformulas of φ that are "to be evaluated" at the current world. An $\mathsf{Existential\text{-}safe}$ formula φ does not need any witness from the current local domain in order to make the formula true. The notions of components and existential-safeness will play a role in the tableau-based decision procedure to be introduced below.

Before going into the specific tableau rules, we first explain the general method. A *tableau* is a tree-like structure generated from a single formula α by repeatedly applying a few rules with some auxiliary information as the root of the tree. Intuitively, a tableau for α is a pseudo model which can be transformed into a real model of α under some simple consistency conditions. We can then decide the satisfiability of a formula by trying to find a proper tableau. As in [17], a tableau T in our setting is a tree structure such that each node is a triple (w, Γ, σ) where w is a symbol or a finite sequence of symbols intended as the *name* of a possible world in the real model, Γ is a finite set of FOML-formulas, and σ is an assignment function for variables. Since we intend to use the set of variables as the domain in the tableau-induced real model, σ is simply a *partial* identity function on Var, i.e., $\sigma(x) = x$ for all $x \in Dom(\sigma) \subseteq \mathsf{Var}$, where the domain of σ, $Dom(\sigma)$, is intended to be the local domain of the real model. The intended meaning of the node (w, Γ, σ) is that all the formulas in Γ are satisfied on w with the assignment σ, thus we also write $(w : \Gamma, \sigma)$ for the triple.

A tableau rule specifies how the node in the premise of the rule is transformed to or connected with one or more new nodes given by the conclusion of the rule. Applying the rules can generate a tree-like structure, a tableau, which is saturated if every leaf node contains only literals. For any formula α, we refer to a saturated tableau of α simply as a tableau of α. Further, a saturated tableau is *open* if in every node $(w : \Gamma, \sigma)$ of the tableau, Γ does not contain both β and $\neg\beta$ for any formula β.[2]

We call a formula *clean* if no variable occurs both bound and free in it and every use of a quantifier quantifies a distinct variable. A finite set of formulas Γ is *clean* if $\bigwedge \Gamma$, the conjunction of all formulas in Γ, is clean. Note that every FOML-formula can be rewritten into an equivalent clean formula. For instance, the formulas $\exists x \Box Px \vee \forall x \Diamond Qx$ and $Px \wedge \Box \exists x Qx$ are not clean, whereas $\exists x \Box Px \vee \forall y \Diamond Qy$ and $Px \wedge \Box \exists y Qy$ are their clean equivalents respectively. Clean formulas help in handling the witnesses for existential formulas in the tableau in a syntactic way.

Consider a finite set of formulas Γ that is clean. Suppose we want to expand Γ to $\Gamma \cup \{\alpha_1, \dots \alpha_k\}$, then even if each of α_i is clean, it is possible that a bound variable of α_i also occurs in some $\varphi \in \Gamma$ or another α_j. To avoid this, first, we rewrite the bound variables in each α_i one by one by using the fresh variables that do not occur in Γ and other previously rewritten α_j.

Such a rewriting can be fixed by always using the first fresh variable in a fixed enumeration of all the variables. When Γ and $\{\alpha_1, \dots \alpha_k\}$ are clear from the context, we denote α_i^* to be such a fixed rewriting of α_i into a clean formula. It is not hard to see that the resulting finite set $\Gamma \cup \{\alpha_1^*, \dots \alpha_k^*\}$ is clean.

3.1 Tableaux for LBF

The tableau rules for the LBF fragment are described in Fig. 1. The $(\wedge)$ and $(\vee)$ rules are standard, where we make a non-deterministic choice of one of the branches for $(\vee)$. The rule (END) says that if we are left with only modules and there are no $\Diamond$ formulas, then the branch does not need to be explored further. The $(\Diamond)$ rule creates one successor world for every $\Diamond$ formula at the current node and includes all the $\Box$ formulas that need to be

[2] Refer [17] for an illustration of a similar tableau construction.

$$\frac{w : \varphi_1 \vee \varphi_2, \Gamma, \sigma}{w : \varphi_1, \Gamma, \sigma \mid\mid w : \varphi_2, \Gamma, \sigma}\ (\vee) \qquad \frac{w : \varphi_1 \wedge \varphi_2, \Gamma, \sigma}{w : \varphi_1, \varphi_2, \Gamma, \sigma}\ (\wedge)$$

$$\frac{w : \exists x\varphi,\ \Gamma,\ \sigma}{w : \varphi,\ \Gamma,\ \sigma'}(\exists)$$

where $\sigma' = \sigma \cup \{(x, x)\}$

$$\frac{w : \forall y\varphi,\ \Gamma,\ \sigma}{w :\ \{\varphi^*[z/y] \mid z \in Dom(\sigma)\},\ \Gamma,\ \sigma}(\forall)$$

where Γ is Existential-safe and every $\varphi^*[z/y]$ is a clean rewriting of $\varphi[z/y]$ with respect to $\Gamma \cup \{\varphi[z/y] \mid z \in Dom(\sigma)\}$

Given $n \geq 1$: and $m, s \geq 0$

$$\frac{\begin{matrix} w : \Diamond\varphi_1, \dots, \Diamond\varphi_n \\ \Box\beta_1, \dots \Box\beta_m \end{matrix}\ l_1, \cdots, l_s,\ \ \sigma}{\langle wv_i : \varphi_i,\ \{\beta_j \mid j \in [1, m]\},\ \sigma\rangle\ \text{ for all } i \in [1, n]}\quad (\Diamond)$$

Given $m \geq 1, s \geq 0$:

$$\frac{w : \Box\beta_1, \dots \Box\beta_m,\ l_1, \dots, l_s, \sigma}{w : l_1, \cdots, l_s, \sigma}\quad (\texttt{END})$$

Figure 1 Tableau rules for LBF, here every l_i is a literal.

satisfied along with the $\Diamond$ formula and σ is inherited in the successor worlds to preserve the increasing domain property. The $(\exists)$ rule picks x itself as the witness to satisfy $\exists x\varphi$ and $(\forall)$ rule expands the set of formulas to include a clean version of $\varphi[z/y]$ for every variable z in the current local domain.

Note that only the $(\Diamond)$ rule can change (the name of) the possible world, thus creating a new successor. It simply extends the name w by new symbols v_i for each successor. Therefore there can be many nodes in the tableau sharing the same world name but such nodes form a path. Given w we use t_w to denote the *last node* sharing the first component w. Given a node $t = (w : \Gamma, \sigma)$ in a tableau, we use $Dom(t)$ to denote the domain of σ.

Also, there is an implicit ordering on how the rules are applied: $(\Diamond)$ rule can be applied at a node (w, Γ, σ) only if all formulas of Γ are modules and hence may be applied only after the $(\wedge, \vee, \forall, \exists)$ rules have been applied as many times as necessary at w. Similarly $(\forall)$ rule can be applied only when Γ is Existential-safe which means that the $(\exists)$ rule cannot be applied anymore at the current node.

▶ **Proposition 8.** *For every tableau T and every node $v = (w, \Gamma, \sigma)$ in T, if v is a leaf then either Γ contains only literals or there is some rule that can be applied at v.*

The proposition is true since the LBF ensures that if Γ is not Existential-safe and the $(\wedge, \vee, \Diamond, \texttt{END})$ rules cannot be applied, then it has to be the case that there is some $\exists x\varphi \in \Gamma$, for which we can apply the $(\exists)$ rule.

▶ **Theorem 9.** *For any clean LBF formula θ, let σ_r be an identity mapping over FV$(\theta) \cup \{z\}$ where z does not occur in θ. There is an open tableau T with root $(r : \{\theta\}, \sigma_r)$ iff θ is satisfiable in an increasing domain model.*

Proof. First, we claim that the rules preserve the cleanliness of the formulas. To see this, we verify that for every rule, if Γ in the antecedent of the rule is clean, then the Γ' obtained after the application of the rules is also clean. This is obvious for $(\wedge), (\vee), (\Diamond)$ and $(\texttt{END})$ rules. The $(\exists)$ rule preserves cleanliness because it frees variable x which is not bound by any other quantifier in the antecedent. The $(\forall)$ rule preserves cleanliness by rewriting.

($\Rightarrow$): Let T be an open tableau rooted at $(r : \{\theta\}, \sigma_r)$. Define a model $\mathcal{M} = (\mathcal{W}, \mathcal{D}, \delta, \mathcal{R}, \rho)$ as follows:

- $\mathcal{W} = \{w \mid (w : \Gamma, \sigma) \text{ is a node in } T\}$
- $\mathcal{D} = \mathsf{Var}$
- $\mathcal{R} = \{(w, v) \mid v \text{ is of the form } wv' \text{ for some } v'\}$
- For every $w \in \mathcal{W}$, define $\delta(w) = Dom(t_w)$ where t_w is the last node of w in T
- For every $w \in \mathcal{W}$ and $p \in \mathcal{P}$, define $\rho(w, P) = \{\overline{x} \mid P\overline{x} \in \Gamma \text{ where } t_w = (w, \Gamma, \sigma)\}$

Clearly, $\mathcal{M}$ is an increasing domain model, and since $z \in Dom(\sigma_r)$, there is no empty local domain. As T is an open tableau, ρ is well-defined.

Claim. For every node $(w : \Gamma, \sigma)$ in T and for every LBF formula φ, if $\varphi \in \Gamma$ then $\mathcal{M}, w, \sigma \models \varphi$.

The claim is proved using a standard argument by induction on the height of the nodes of T from the leaves to the root (details in [13]). Thus, from the claim it follows that $\mathcal{M}, r, \sigma_r \models \theta$ since the label of the root of T is $(r : \{\theta\}, \sigma_r)$.

($\Leftarrow$) From Proposition 8 it follows that we can always apply some rule until every leaf node $(w : \Gamma, \sigma)$ is such that Γ contains only literals. Thus every (partial) tableau can be extended to a saturated tableau. To prove that such a tableau is open, it suffices to show that all rules preserve satisfiability (details in [13]). ◀

Note that the depth of the tableau is linear in the size of the formula. However, as we have to rewrite formulas using new variables when applying ($\forall$) rule, the size of the domain is exponential in the size of the formula. Hence, the tableau procedure can be implemented in ExpSpace.

▶ **Corollary 10.** *LBF is decidable in ExpSpace.*

4 The (Un)decidability Border

Note that the fragment LBF cannot express formulas of the form $\forall x \exists y \Box\ \alpha$ and also $\forall x \Box \forall y \Box \forall z \alpha$. There are many combinations of the bundled fragments that can express thesse formulas (like $\exists\Box + \Box\forall$ and $\forall\Box + \exists\Box + \Box\exists$ fragments). In fact, we can prove that a bundled fragment is undecidable if we can assert both $\forall x \exists y \Box \alpha$ and $\forall x \Box \forall y \Box \forall z\ \alpha$ in the fragment.

To prove this, we can use tiling encoding where $\forall x \exists y \Box \alpha$ can be used to assert that every "grid point" x has a horizontal/vertical successor y. In this case, it is important that both quantifiers are applicable over the same local domain and $\Box \alpha$ in $\forall x \exists y \Box \alpha$ ensures that the witness y acts uniformly across all the descendants. The second formula $\forall x \Box \forall y \Box \forall z\ \alpha$ is used to verify the "diagonal property" of the grid. In the companion technical report of this paper [13] we prove these results formally.

Also, note that there are fragments like $\exists\Box + \Box\exists$ where $\forall x \exists y \Box \alpha$ is expressible but not $\forall x \Box \forall y \Box \forall z\ \alpha$. In these cases, we can prove that such fragments do not have a finite model property. This is also proved in the companion technical report [13] where we show that this fragment gives a formula that can induce a linear order on the local domain of some world in the model and assert that this linear order does not have a maximal element.

This leaves us with the fragments that cannot express $\forall x \exists y \Box \alpha$ formulas and LBF is one such fragment which we proved to be decidable. The fragments $\forall\Box + \exists\Box$ and $\Box\forall + \Box\exists$ also fall in this category and since they are subfragments of LBF, decidability follows. So we *only* need to consider the fragment $\forall\Box + \exists\Box + \Box\forall$ to complete the terrain (cf. Table. 1).

5 The $\forall\Box + \Box\forall + \Box\exists$ Fragment

In this fragment, we are allowed to express $\forall x\Box\alpha$, $\Box\forall x\alpha$ and $\Box\exists x\alpha$ and their duals. Note that this fragment is not closed under subformulas. For instance, $\varphi := \forall x\left(\exists y\Diamond\alpha \vee \forall z\Box\beta\right)$ is a subformula of $\varphi' := \Diamond\forall x\left(\exists y\Diamond\alpha \vee \forall z\Box\beta\right)$. But φ' is in the fragment and φ is not in the fragment. We say that φ is a subformula of $\forall\Box + \Box\forall + \Box\exists$ if there is some formula $\varphi' \in \forall\Box + \Box\forall + \Box\exists$ such that φ is a subformula of φ'.

Note that even though we cannot express formulas of the form $\forall x\exists y\Box\varphi$ in the fragment, $\forall x\exists y\Diamond\varphi$ is still allowed. For instance, the formula $\Diamond\forall x\left(\exists y\Diamond\alpha\right)$ is in the fragment. Thus, we can have $\forall x\exists y\Diamond\varphi$ but not $\forall x\exists y\Box\varphi$. Intuitively this means that the different witnesses y for each x can work on *different* successor worlds. The fragment cannot enforce the interaction between x and y at all successors. This property can be used to prove that we can reuse the witnesses by creating new successor subtrees as required.

To get the decidability for $\forall\Box + \Box\forall + \Box\exists$ fragment, the main idea is to prove that the formulas of the form $\forall x\exists y\Diamond\varphi$ can be satisfied by picking some boundedly many witnesses y that will work for *all* x. This is the same as proving that if $\forall x\exists y\Diamond\varphi$ is satisfiable then $\exists y_1,\dots\exists y_l\forall x\big(\bigvee\Diamond\varphi[y/y_i]\big)$ is satisfiable (where l is bounded). We illustrate the proof idea with an example.

▶ **Example 11.** Consider the formula $\alpha := \forall x\Big(\Box\neg Pxx \wedge \exists y\Diamond Pxy\Big)$ which is a subformula of the fragment. Let $\mathcal{T}, r \models \alpha$ where $\mathcal{T}$ is a tree model rooted at r. Now we will modify $\mathcal{T}$ to obtain $\mathcal{M}$ which is also a tree model rooted at r such that $\mathcal{M}, r \models \exists y_1\exists y_2\forall x\left(\Box\neg Pxx \wedge \left(\,\Diamond Pxy_1 \vee \Diamond Pxy_2\,\right)\right)$.

The model $\mathcal{M}$ is obtained by extending $\mathcal{T}$ in the following way. Let $\delta^{\mathcal{T}}(r) = \mathcal{D}_r$. To obtain $\mathcal{M}$, first we extend the local domain of r by adding a fresh element a. The idea is that for every $d \in \mathcal{D}_r$ (when assigned to x) we will ensure that the new element a can be picked as the y-witness. To achieve this, we do the following: For every $d \in \mathcal{D}_r$ let $d' \in \mathcal{D}_r$ and $(r, s^d) \in \mathcal{R}^{\mathcal{T}}$ such that $\mathcal{T}, s^d \models Pdd'$. Let $\mathcal{T}^d$ be the subtree of $\mathcal{T}$ rooted at s^d. We will create a new copy of $\mathcal{T}^d$ and call its root u^d. Now, in the new subtree rooted at u^d, we make the new element a "behave" like d' and we add an edge from r to u^d. So, in particular, $\mathcal{M}$ will have $(r, u^d) \in \mathcal{R}^{\mathcal{M}}$ such that $\mathcal{M}, u^d \models Pda$. Since we do this construction for every $d \in \mathcal{D}_r$ we obtain that for all $d \in \mathcal{D}_r$ we have $\mathcal{M}, r \models \Diamond Pda$.

Now note that while evaluating α at $(\mathcal{M}, r)$ the $\forall x$ quantification will now also apply to a (since a is added to the local domain at r in $\mathcal{M}$). But then, we cannot use a itself as the witness for a since we also need to ensure that $\mathcal{M}, r \models \forall x\Box\neg Pxx$. Hence we will add another element b that acts as a witness for a. Further, b also needs a witness. But now we can choose a to be the witness for b since that does not violate the formula $\forall x\Box\neg Pxx$.

So to complete the construction, we pick some arbitrary $d \in \mathcal{D}_r$ for which we have some $d' \in \mathcal{D}_r$ and $(r, s^d) \in \mathcal{R}^{\mathcal{T}}$ such that $\mathcal{T}, s^d \models Pdd'$. We create two copies of $\mathcal{T}^d$ (subtree rooted at s^d) and call their roots as v^d and w^d respectively. In the subtree rooted at v^d we ensure that a and b "behave" like d, d' respectively and in the subtree rooted at w^d we ensure that a and b "behave" like d', d respectively. In particular, we have $\mathcal{M}, v^d \models Pab$ and $\mathcal{M}, w^d \models Pba$. Finally we add edges from r to v^d and from r to w^d in $\mathcal{M}$.

Thus, we have: $\delta^{\mathcal{M}}(r) = \delta^{\mathcal{T}}(r) \cup \{a, b\}$ and $\mathcal{M}, r \models \exists y_1\exists y_2\forall x\left(\Box\neg Pxx \wedge \left(\,\Diamond Pxy_1 \vee \Diamond Pxy_2\,\right)\right)$. With the above construction, this assertion can be verified by assigning y_1 and y_2 to a and b respectively.

Note that in principle, it is possible for an $\exists$ quantified formula to occur in the scope of a $\forall$ quantifier as a boolean combination with other $\exists$ quantified formulas and modules. Moreover, these additional formulas can assert some "type" information that may force us to pick additional witnesses. For example, if the formula is

$$\forall x\Big[\Big(\Box(\neg Pxx \wedge Rx) \vee \Box(\neg Pxx \wedge \neg Rx)\Big) \wedge \exists y \Diamond\big(Rx \to (Pxy \wedge \neg Ry) \ \wedge\ \neg Rx \to (Pxy \wedge Ry)\big)\Big]$$

then we need two initial y-witnesses a_1, a_2 where one is used for witness whose "type" is $\Box(\neg Pyy \wedge Ry)$ and other for witness whose "type" is $\Box(\neg Pyy \wedge \neg Ry)$ and we also need the corresponding additional witnesses b_1, b_2. In general, the formula can force us to pick witnesses of a particular "1-type" which means we might need exponentially many witnesses.

Thus, we need to replace one $\exists$ inside the scope of $\forall$ by $2l$ many $\exists$ quantifiers outside the scope of $\forall$ where l is bounded exponentially in the size of the given formulas. We now prove this formally.

For any formula φ if $\alpha \in \mathsf{C}(\varphi)$ we denote this by $\varphi[\alpha]$. This means that α does not occur inside the scope of any modality in φ. Further, for every $\alpha \in \mathsf{C}(\varphi)$ and a formula β, we denote $\varphi[\beta/\alpha]$ obtained by rewriting φ where every occurrence of α in φ is replaced by β. In particular, we are interested in the case where α is of the form $\exists y \Diamond \psi$. Thus we always consider $\varphi[\exists y \Diamond \psi]$.

For every $l \geq 0$ if $\overline{y} = y_1, y'_1 \dots y_l, y'_l$ are fresh variables, we denote $\overline{y}\Diamond\psi$ to be the formula $\bigvee\limits_{i \leq l} \big(\Diamond\psi[y_i/y] \vee \Diamond\psi[y'_i/y]\big)$ which is a big disjunction where each disjunct replaces y in ψ with one of y_i or y'_i. Further, we denote $\varphi[\overline{y}\Diamond\psi/\exists y\Diamond\psi]$ as simply $\varphi[\overline{y}\Diamond\psi]$.

For instance, for the formula $\varphi := \Big(Px \ \vee\ \exists y \Diamond Qxy\Big)$ where $\psi := \exists y \Diamond Qxy$, for $l = 2$ and $\overline{y} = y_1, y'_1, y_2, y'_2$ being fresh variables, $\varphi[\overline{y}\Diamond\psi]$ is given by: $\Big(Px \ \vee \big(\Diamond Qxy_1 \vee \Diamond Qxy'_1 \vee \Diamond Qxy_2 \vee \Diamond Qxy'_2\big)\Big)$.

The size of a formula, denoted by $|\varphi|$, is the number of symbols occurring in φ and for a finite set of formulas Γ, let $|\Gamma| = \sum\limits_{\varphi \in \Gamma} |\varphi|$.

▶ **Lemma 12.** *Let Γ' be a clean finite set of formulas such that every $\alpha \in \Gamma$ is a subformula of $\forall\Box + \Box\forall + \Box\exists$ where $\Gamma' = \Gamma \cup \{\forall x \varphi[\exists y \Diamond\psi]\}$. If $\bigwedge \Gamma \wedge \forall x \varphi[\exists y \Diamond \psi]$ is satisfiable then there exists $l \leq 2^{|\Gamma'|}$ such that $\bigwedge \Gamma \ \wedge\ \exists y_1 \exists y'_1 \exists y_2 \exists y'_2 \dots \exists y_l \exists y'_l\ \forall x\ \varphi[\overline{y}\Diamond\psi]$ is satisfiable, where $\overline{y} = y_1, y'_1, \dots y_l, y'_l$ are fresh variables.*

Note that the $\exists y$ quantifier is pulled outside the scope of the $\forall x$ quantifier and replaced with a bounded number of witnesses $y_1, y'_1, y'_2, y'_2 \dots y_l, y'_l$. Consequently $\Diamond\ \psi$ is replaced with a disjunction each replacing y with one of y_i or y'_i for every $i \leq l$.

To prove the lemma first we formally define the tree editing operation described in the example. Given a tree model $\mathcal{T}$ rooted at r, let $d \notin \mathcal{D}^{\mathcal{T}}$. To add the new domain element d to a local domain of r, we also need to specify the "type" of the new element d at r and its descendants. Towards this, we pick some domain element c that is already present in $\delta(r)$ and assign the type of d to the type of c at every world.

▶ **Definition 13.** *Given a tree model $\mathcal{T} = (\mathcal{W}, \mathcal{D}, \mathcal{R}, \delta, \rho)$ rooted at r, let $d \notin \mathcal{D}$ and $c \in \delta(r)$. Define the operation of "adding d to $\delta(r)$ by mimicking c", denoted by $\mathcal{T}_{d \mapsto c} = (\mathcal{W}, \mathcal{D}', \mathcal{R}, \delta', \rho')$ where:*

- $\mathcal{D}' = \mathcal{D} \cup \{d\}$
- *for all $w \in \mathcal{W}$ we have $\delta'(w) = \delta(w) \cup \{d\}$*

- *For every $w \in \mathcal{W}$ and predicate P define* $\rho'(w,P) = \{\overline{e'} \mid$ *there is some* $\overline{e} \in \rho(w,P)$ *and* $\overline{e'}$ *is obtained from* $\overline{e}$ *by replacing zero or more occurrences of* c *in* $\overline{e}$ *by* $d\}$.

Suppose that we want to extend the domain with $\overline{d} = d_1 \cdots d_n$ *which are fresh. Let* $\omega : \overline{d} \mapsto \mathcal{D}'$ *where* $\mathcal{D}' \subseteq \delta^T(r)$ *and we want each* d_i *to mimic* $\omega(d_i)$*. Then we denote* $\mathcal{T}_\omega$ *to be the tree obtained by* $\Big((\mathcal{T}_{d_1 \mapsto \omega(d_1)})_{d_2 \mapsto \omega(d_2)\ldots}\Big)_{d_n \mapsto \omega(d_n)}$.

▶ **Proposition 14.** *Let* $\mathcal{T} = (\mathcal{W}, \mathcal{D}, \mathcal{R}, \delta, \rho)$ *be a tree model rooted at* r *with* $\mathcal{T}_{d \mapsto c}$ *being an extended tree where* $d \notin \mathcal{D}$ *and* $c \in \delta(r)$*. Then for all interpretations* σ *and for all* FOML *formulas* φ *and for all* $w \in \mathcal{W}$ *we have:*
$\mathcal{T}, w, \sigma_{[x \mapsto c]} \models \varphi$ *iff* $\mathcal{T}_{d \mapsto c}, w, \sigma_{[x \mapsto c]} \models \varphi$ *iff* $\mathcal{T}_{d \mapsto c}, w, \sigma_{[x \mapsto d]} \models \varphi$.

The proposition holds since there is no equality in the syntax and at every world in the extended model, the new element d "behaves" like c and the old elements "behave" like themselves (details in [13]). Now we are ready to prove Lemma 12.

Proof of Lemma 12. Let $\mathcal{T}$ be a tree model rooted at r such that $\mathcal{T}, r, \sigma \models \bigwedge \Gamma \wedge \forall x \varphi[\exists y \Diamond \psi]$. For every domain element $a \in \delta^T(r)$ define:

$$\Pi(r,a) = \bigcup_{\forall x' \alpha \in \Gamma'} \{\lambda \mid \lambda \in \mathsf{C}(\alpha) \text{ and } T, r, \sigma_{[x' \mapsto a]} \models \lambda\}$$

Note that $\Pi(r,a)$ formalizes the notion of "type" of a at the world r. This includes the information of all subformulas that are true at the current world, when a universal variable is instantiated with a. Also, since the size of $|\mathsf{C}(\alpha)|$ is at most the size of Γ', the set $\Pi(r) = \{\Pi(r,a) \mid a \in \delta^T(r)\}$ has size $|\Pi(r)| = l$ where $l \leq 2^{|\Gamma'|}$. Enumerate $\Pi(r) = \{\Lambda_1, \ldots \Lambda_l\}$ and for every $i \leq l$ pick $a_i \in \delta^T(r)$ such that $\Pi(r, a_i) = \Lambda_i$. Now let $\overline{d} = d_1, d_1', d_2, d_2', \ldots d_l, d_l'$ be fresh domain elements and let $\omega : \overline{d} \mapsto \{a_1, a_2, \ldots a_l\}$ where for all $i \leq l$ we have $\omega(d_i) = \omega(d_i') = a_i$. We define the required model $\mathcal{M} = (\mathcal{W}', \mathcal{D}', \mathcal{R}', \delta', \rho')$ as follows:

Let $\mathcal{M}_0 = \mathcal{T}_\omega$ be the new tree model rooted at r obtained by adding $d_1, d_1', \ldots, d_l, d_l'$ to $\delta^T(r)$ where each d_i and d_i' mimics a_i. Now $\mathcal{M}$ is obtained by extending $\mathcal{M}_0$ as follows:

For every $c \in \delta(r)$ such that $\mathcal{T}, r, [x \mapsto c] \models \exists y \Diamond \psi$ we pick $c' \in \delta^T(r)$ and $r \to s^c$ be such that $\mathcal{T}, s^c, [xy \mapsto cc'] \models \psi$. Let $\mathcal{T}^c$ be the sub-tree of $\mathcal{T}$ rooted at s^c and $\Pi(r,c') = \Lambda_j$. Create a new subtree $\mathcal{T}_0^c = \mathcal{T}_{\omega'}^c$ where for all $h \neq j$ we have $\omega'(d_h) = \omega'(d_h') = a_h$ and $\omega'(d_j) = \omega'(d_j') = c'$. Let u^c be the root of $\mathcal{T}_0^c$. Add an edge from r to u^c in $\mathcal{M}$.
Further, for every $i \leq l$ if $\mathcal{T}, r, \sigma_{[x \mapsto a_i]} \models \exists y \Diamond \psi$ then let $b \in \delta^T(r)$ and $r \to s^i \in \mathcal{R}^T$ be such that $\mathcal{T}, s^i, \sigma_{[xy \mapsto a_i b]} \models \psi$. Let $\Pi(r,b) = \Lambda_j$. Then create $\mathcal{T}_1^i = \mathcal{T}_{\omega_1}^i$ and $\mathcal{T}_2^i = \mathcal{T}_{\omega_2}^i$ where ω_1 and ω_2 are defined as follows:

- For all $h \neq j$, $\omega_1(d_h) = \omega_1(d_h') = \omega_2(d_h) = \omega_2(d_h') = a_h$
- $\omega_1(d_j) = a_j$ and $\omega_1(d_j') = b$
- $\omega_2(d_j) = b$ and $\omega_2(d_j') = a_j$

Let v^i and w^i be the root of $\mathcal{T}_1^i$ and $\mathcal{T}_2^i$ respectively. Add the edges from r to v^i and from r to w^i in $\mathcal{M}$. The two copies of subtrees are intended to provide witnesses for $\exists y \Diamond \psi$ for d_j and d_j' respectively. We need the two copies to ensure that ω_1 and ω_2 are well defined in the case when $i = j$ and $a_j \neq b$.

We now explain the idea behind the construction. Note that $\mathcal{T}_0^c$ rooted at u^c is created for every $c \in \delta^T(r)$ such that $\mathcal{T}, r, \sigma_{[x \mapsto c]} \models \exists y \Diamond \psi$. If c' is the picked witness for c with $(r, s^c) \in \mathcal{R}^T$ such that $\mathcal{T}, s^c, \sigma_{[xy \mapsto cc']} \models \psi$ and $\Pi(r,c') = \Lambda_j$ then by construction, $\mathcal{T}_0^c$ is rooted u^c where d_j mimics c' in the subtree rooted at u^c. All these together indicate that we can use d_j and the subtree rooted at u^c in $\mathcal{M}$ to verify that $\mathcal{M}, u^c, \sigma_{[xy \mapsto cd_j]} \models \psi$. Also note that for all $h \neq j$ the fresh elements d_h and d'_h mimic a_h in the subtree $\mathcal{T}_0^c$ (i.e, we have not added any extra "types").

Further, we want the type of d_i and d'_i at r in $\mathcal{M}$ to mimic the type of a_i at r in $\mathcal{T}$. All type information is taken care of in $\mathcal{M}_0$ where both d_i and d'_i mimic a_i except the formula $\exists y \Diamond \psi$. So if $\mathcal{T}, r, \sigma_{[x \mapsto a_i]} \models \exists y \Diamond \psi$ then we need a witness to verify $\mathcal{M}, r, \sigma_{[x \mapsto d_i]} \models \exists y \Diamond \psi$ and $\mathcal{M}, r, \sigma_{[x \mapsto d'_i]} \models \exists y \Diamond \psi$. If the witness for y for a_i is b and $\Pi(r, b) = \Lambda_j$ then we want d'_j to be the witness for d_i and d_j to be the witness for d'_i.

Consequently if s^i is the world such that $a \to s^i \in \mathcal{R}^T$ and $\mathcal{T}, s^i, [xy \mapsto a_i b] \models \psi$ then we create two new copies of subtree $\mathcal{T}^i$ rooted at s^i and call it $\mathcal{T}_1^i$ and $\mathcal{T}_2^i$. By construction, in particular, the new element d_i mimics a_i and d'_j mimics b in $\mathcal{T}_1^i$. Similarly d'_i mimics a_i and d_j mimics b in $\mathcal{T}_2^i$. Thus, we can pick d'_j to be the witness for d_i (and consider $\mathcal{T}_1^i$) and pick d_j to be the witness for d'_i (and consider $\mathcal{T}_2^i$).

Also, it is important to note that for every $r \to v \in \mathcal{R}^M$, if d_i mimics c and d'_i mimics c' at v then we will always have $\Pi(r, c) = \Pi(r, c') = \Lambda_i = \Pi(r, a_i)$. Now it can be verified that $\mathcal{M}, r, \sigma \models \bigwedge \Gamma \ \wedge \ \exists y_1 \exists y'_1 \ldots \exists y_l \exists y'_l \ \forall x \varphi[\overline{y} \Diamond \psi]$.

The details are provided in [13]. ◀

▶ **Corollary 15.** *Let Γ' be a clean finite set of formulas such that every $\alpha \in \Gamma$ is a subformula of $\forall\Box + \Box\forall + \Box\exists$ where $\Gamma' = \Gamma \cup \{\forall x \varphi[\exists y \Diamond \psi]\}$. If $\bigwedge \Gamma \wedge \forall x \varphi[\exists y \Diamond \psi]$ is satisfiable then $\bigwedge \Gamma \wedge \exists y_1 \exists y'_1 \exists y_2 \exists y'_2 \ldots \exists y_l \exists y'_l \ \forall x \ \varphi[\overline{y} \Diamond \psi]$ is satisfiable, where $l = 2^{|\Gamma'|}$ and $\overline{y} = y_1, y'_1, \ldots y_l, y'_l$ are fresh variables.*

To see why the corollary is true, by Lemma 12 we get some $l \leq 2^{|\Gamma'|}$, and we can pad sufficiently many dummy variables to get a strict equality. This gives us a useful tableau rule which we call $(\forall\exists\Diamond)$ rule for $\forall\Box + \Box\forall + \Box\exists$ fragment, described in Fig. 2. The full tableau rules for $\forall\Box + \Box\forall + \Box\exists$ is given by the tableau rules of LBF (Fig. 1) along with the $(\forall\exists\Diamond)$-rule.

$$\frac{w : \forall x \ \varphi[\exists y \Diamond \psi], \ \Gamma, \ \sigma}{w : \forall x \ \varphi[\overline{y} \Diamond \psi], \ \Gamma, \ \sigma'} (\forall\exists\Diamond)$$

where $l = 2^{|\Gamma| + |\varphi|}$ and $\overline{y} = y_1, y'_1, \ldots y_l, y'_l$
are fresh variables and $\sigma' = \sigma \cup \{(y_i, y_i) \ , (y'_i, y'_i) \mid i \leq l\}$

Figure 2 (The $\forall\exists\Diamond$) rule for $\forall\Box + \Box\forall + \Box\exists$ fragment.

▶ **Theorem 16.** *For any clean $\forall\Box + \Box\forall + \Box\exists$ formula θ, let σ_r be an identity mapping over $FV(\theta) \cup \{z\}$ where z does not occur in θ. There is an open tableau with $(r : \{\theta\}, \sigma_r)$ as the root iff θ is satisfiable in an increasing domain model.*

The proof follows along the lines of Theorem 9. The only interesting part of the proof is to show that the $(\forall\exists\Diamond)$ rule preserves satisfiability and this is by Lemma 12(details in [13]).

Note that at if we start with a formula of length n then the application of $(\forall\exists\Diamond)$ rule will blow up the formula to size 2^n. So we have a tableau procedure that can be implemented as an algorithm in ExpSpace.

▶ **Corollary 17.** *The fragment $\forall\Box + \Box\forall + \Box\exists$ is decidable in ExpSpace.*

6 Conclusion

In this paper, we have studied the decidability of bundled fragments of FOML, where we have no restrictions on the use of variables or arity of relations. Specifically, we proved the decidability of the loosely bundled fragment LBF and the $\forall\Box + \Box\forall + \Box\exists$ fragment. The decidability of these fragments hinges on the observation that $\forall x \exists y \Box \alpha$ is not expressible. A NEXPTIME lower bound follows for $\forall\Box + \exists\Box$ and $\Box\forall + \Box\exists$ (via encoding the corresponding version of the tiling problem [13]), which implies the same lower bound for LBF and $\forall\Box + \Box\forall + \Box\exists$. There is a significant gap between the upper and lower complexity bounds and we need sharper technical tools for investigating lower bounds for bundled fragments.

Note that the quantifier prefix in LBF is of the form $\exists^*\forall^*$ and hence any extension of this quantifier prefix or extending LBF with negation closure will result in a fragment that will be able to express $\forall x \exists y \Box\ \alpha$ (and hence will not have the finite model property, [13]). In this sense, LBF is the largest fragment in which $\forall x \exists y \Box \alpha$ is not (syntactically) expressible. For the fragment $\forall\Box + \Box\forall + \Box\exists$ we introduced a technique to pull out $\exists$ quantifiers outside the immediate scope of $\forall$ and obtain a finite model property. This technique may be useful in studying other fragments of first-order modal logic.

The results in the paper, along with those in [13], provide a trichotomy classification of combinations of bundled operators over increasing domain models (Table 1). The fragments in which we can express both $\forall x \exists y \Box \alpha$ and $\forall x \Box \forall y \Box \forall z \beta$ are undecidable, fragments in which we can express the former but not the latter lack finite model property (but decidability is open) and fragments where we cannot express the former are decidable. Similar trichotomy can also be proved for satisfiability over constant domain models [13].

We have considered only the "pure" fragments, without constants, function symbols, or equality. The addition of constants is by itself simple, but equality complicates things considerably. Since equality is extensively used in specifications, mapping fragments with equality is an important direction. The study of bundles over models with various frame conditions is also relevant for applications. Unfortunately, while it is clear that equivalence frames lead to undecidability [17], even with transitive frames the situation is unclear. Obtaining good decidable fragments over linear frames is an important challenge.

In the context of verification of infinite-state systems, we are often more interested in the MODEL CHECKING problem than in satisfiability. If the domain is finite, the problem is no different from model checking of first-order modal logic. However, we are usually interested in the specification being checked against a finitely specified (potentially infinite) model, e.g., when the domain elements form a *regular* infinite set. This is a direction to be pursued in the context of bundled fragments.

We have presented tableau-based decision procedures that are easily implementable, but inference systems for reasoning in these logics require further study.

References

1 Hajnal Andréka, István Németi, 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.

2 Francesco Belardinelli and Alessio Lomuscio. Quantified epistemic logics for reasoning about knowledge in multi-agent systems. *Artif. Intell.*, 173(9-10):982–1013, 2009. doi:10.1016/j.artint.2009.02.003.

3 Francesco Belardinelli and Alessio Lomuscio. Interactions between knowledge and time in a first-order logic for multi-agent systems: Completeness results. *J. Artif. Intell. Res.*, 45:1–45, 2012. doi:10.1613/jair.3547.

4 Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem.* Springer Science & Business Media, 2001.

5 Torben Braüner and Silvio Ghilardi. First-order modal logic. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 549–620. Elsevier, 2007.

6 Clare Dixon, Michael Fisher, Boris Konev, and Alexei Lisitsa. Practical first-order temporal reasoning. In *Proceedings of TIME 2008*, pages 156–163, 2008. doi:10.1109/TIME.2008.15.

7 Harald Ganzinger, Christoph Meyer, and Margus Veanes. The two-variable guarded fragment with transitive relations. In *Proceedings of LICS '99*, pages 24–34, 1999. doi:10.1109/LICS.1999.782582.

8 Ian Hodkinson, Frank Wolter, and Michael Zakharyaschev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Log.*, 106(1-3):85–134, 2000. doi:10.1016/S0168-0072(00)00018-X.

9 Ian Hodkinson, Frank Wolter, and Michael Zakharyaschev. Monodic fragments of first-order temporal logics: 2000-2001 A.D. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of LPAR '01*, volume 2250, pages 1–23. Springer, 2001. doi:10.1007/3-540-45653-8_1.

10 Ian Hodkinson, Frank Wolter, and Michael Zakharyaschev. Decidable and undecidable fragments of first-order branching temporal logics. In *Proceedings LICS 2002*, pages 393–402, 2002. doi:10.1109/LICS.2002.1029847.

11 G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic.* Routledge, 1996.

12 Mo Liu. On the decision problems of some bundled fragments of first-order modal logic. Master's thesis, Peking University, 2019. URL: https://arxiv.org/abs/2201.02336.

13 Mo Liu, Anantha Padmanabha, Ramaswamy Ramanujam, and Yanjing Wang. Are bundles good deals for FOML? *CoRR*, abs/2202.01581, 2022. arXiv:2202.01581.

14 Anantha Padmanabha, R Ramanujam, and Yanjing Wang. Bundled Fragments of First-Order Modal Logic: (Un)Decidability. In *Proceedings of FSTTCS 2018*, volume 122, pages 43:1–43:20, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.43.

15 Mikhail N. Rybakov and Dmitry Shkatov. Undecidability of first-order modal and intuitionistic logics with two variables and one monadic predicate letter. *Stud Logica*, 107(4):695–717, 2019. doi:10.1007/s11225-018-9815-7.

16 Moshe Y Vardi. Why is modal logic so robustly decidable? Technical report, Rice University, 1997.

17 Yanjing Wang. A new modal framework for epistemic logic. In *Proceedings of TARK 2017*, pages 515–534, 2017. doi:10.4204/EPTCS.251.38.

18 Yanjing Wang. Beyond Knowing That: A New Generation of Epistemic Logics. In *Outstanding Contributions to Logic*, volume 12, pages 499–533. Springer Nature, 2018. doi:10.1007/978-3-319-62864-6_21.

19 Frank Wolter and Michael Zakharyaschev. Decidable fragments of first-order modal logics. *J. Symb. Log.*, 66(3):1415–1438, 2001. URL: http://www.jstor.org/stable/2695115.

Membership Problems in Finite Groups

Markus Lohrey
Universität Siegen, Germany

Andreas Rosowski
Universität Siegen, Germany

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

Abstract

We show that the subset sum problem, the knapsack problem and the rational subset membership problem for permutation groups are NP-complete. Concerning the knapsack problem we obtain NP-completeness for every fixed $n \geq 3$, where n is the number of permutations in the knapsack equation. In other words: membership in products of three cyclic permutation groups is NP-complete. This sharpens a result of Luks [36], which states NP-completeness of the membership problem for products of three abelian permutation groups. We also consider the context-free membership problem in permutation groups and prove that it is PSPACE-complete but NP-complete for a restricted class of context-free grammars where acyclic derivation trees must have constant Horton-Strahler number. Our upper bounds hold for black box groups. The results for context-free membership problems in permutation groups yield new complexity bounds for various intersection non-emptiness problems for DFAs and a single context-free grammar.

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

Keywords and phrases algorithms for finite groups, intersection non-emptiness problems, knapsack problems in groups

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.71

Related Version *Full Version*: https://arxiv.org/abs/2206.11756

1 Introduction

Membership problems in groups. The general problem that we study in this paper is the following: Fix a class $\mathcal{C}$ of formal languages. We assume that members of $\mathcal{C}$ have a finite description; typical choices are the class of regular or context-free languages, or a singleton class $\mathcal{C} = \{L\}$. We are given a language $L \in \mathcal{C}$ with $L \subseteq \Sigma^*$, a group G together with a morphism $h : \Sigma^* \to G$ from the free monoid Σ^* to the group G, and a word $w \in \Sigma^*$. The question that we want to answer is whether $w \in h^{-1}(h(L))$, i.e., whether the group element $h(w)$ belongs to $h(L)$. One can study this problem under several settings, and each of these settings has a different motivation. First of all, one can consider the case where G is a fixed finitely generated group that is finitely generated by Σ. One could call this problem the $\mathcal{C}$-membership problem for the group G. The best studied case is the *rational subset membership problem*, where $\mathcal{C}$ is the class of regular languages. It generalizes the subgroup membership problem for G, a classical decision problem in group theory. Other special cases of the rational subset membership problem that have been studied in the past are the submonoid membership problem, the knapsack problem and the subset sum problem, see e.g. [31, 37]. It is a simple observation that for the rational subset membership problem, the word w (which is tested for membership in $h^{-1}(h(L))$ can be assumed to be the empty word, see [28, Theorem 3.1].

In this paper, we study another setting of the above generic problem, where G is a finite group that is part of the input (and L still comes from a languages class $\mathcal{C}$). For the rest of the introduction we restrict to the case where G is a finite symmetric group S_m (the set of

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 71; pp. 71:1–71:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

all permutations on $\{1, \ldots, m\}$) that is represented in the input by the integer m in unary representation, i.e., by the word $\$^m$.[1] Our applications only make use of this case, but we remark that our upper complexity bounds can be proven in the more general black box setting [6] (in particular, one could replace symmetric groups by matrix groups over a finite field and still obtain the same complexity bounds). Note that $|S_m| = m!$, hence the order of the group is exponential in the input length.

Membership problems for permutation groups. One of the best studied membership problems for permutation groups is the *subgroup membership problem*: the input is a unary encoded number m and a list of permutations $a, a_1, \ldots, a_n \in S_m$, and it is asked whether a belongs to the subgroup of S_m generated by $a_1, \ldots, a_n$. The famous Schreier-Sims algorithm solves this problem in polynomial time [39], and the problem is known to be in NC [5].

Several generalizations of the subgroup membership problem have been studied. Luks defined the k-membership problem ($k \geq 1$) as follows: The input is a unary encoded number m, a permutation $a \in S_m$ and a list of k permutation groups $G_1, G_2, \ldots, G_k \leq S_m$ (every G_i is given by a list of generators). The question is whether a belongs to the product $G_1 \cdot G_2 \cdots G_k$. It is a famous open problem whether 2-membership can be solved in polynomial time. This problem is equivalent to several other important algorithmic problems in permutation groups: computing the intersection of permutation groups, computing set stabilizers or centralizers, checking equality of double cosets, see [36] for details. On the other hand, Luks has shown in [36] that k-membership is NP-complete for every $k \geq 3$. In fact, NP-hardness of 3-membership holds for the special case where $G_1 = G_3$ and G_1 and G_2 are both abelian.

Note that the k-membership problem is a special case of rational subset membership for symmetric groups. Let us define this problem again for the setting of symmetric groups (here, 1 denotes the identity permutation and we identify a word over the alphabet S_m with the permutation to which it evaluates):

▶ **Problem 1.1** (rational subset membership problem for symmetric groups).
Input: a unary encoded number m and a nondeterministic finite automaton (NFA) $\mathcal{A}$ over the alphabet S_m.
Question: Does $1 \in L(\mathcal{A})$ hold?

An obvious generalization of the rational subset membership problem for symmetric groups is the *context-free subset membership problem for symmetric groups*; it is obtained by replacing the NFA $\mathcal{A}$ in Problem 1.1 by a context-free grammar $\mathcal{G}$.

Two restrictions of the rational subset membership problem that have been intensively studied for infinite groups in recent years are the *knapsack problem* and *subset sum problem*, see e.g. [4, 8, 9, 21, 22, 29, 32, 34, 37]. For symmetric groups, these problems are defined as follows (note that the number $n + 1$ of permutations is part of the input):

▶ **Problem 1.2** (subset sum problem for symmetric groups).
Input: a unary encoded number m and permutations $a, a_1, \ldots, a_n \in S_m$.
Question: Are there $i_1, \ldots, i_n \in \{0, 1\}$ such that $a = a_1^{i_1} \cdots a_n^{i_n}$?

Note that the subset sum problem is the membership problem for *cubes*, which are subsets of the form $\{a_1^{i_1} \cdots a_n^{i_n} \mid i_1, \ldots, i_n \in \{0, 1\}\}$ [6].

[1] We could also consider the case where G is a subgroup of S_m that is given by a list of generators (i.e., G is a permutation group), but this makes no difference for our problems.

▶ **Problem 1.3** (knapsack problem for symmetric groups).
Input: a unary encoded number m and permutations $a, a_1, \dots, a_n \in S_m$.
Question: Are there $i_1, \dots, i_n \in \mathbb{N}$ such that $a = a_1^{i_1} \cdots a_n^{i_n}$?

We will also consider the following restrictions of these problems.

▶ **Problem 1.4** (abelian subset sum problem for symmetric groups).
Input: a unary encoded number m and pairwise commuting permutations $a, a_1, \dots, a_n \in S_m$.
Question: Are there $i_1, \dots, i_n \in \{0,1\}$ such that $a = a_1^{i_1} \cdots a_n^{i_n}$?

The following problem is the special case of Luks' k-membership problem for cyclic groups. Note that k is a fixed constant here.

▶ **Problem 1.5** (k-knapsack problem for symmetric groups).
Input: a unary encoded number m and $k+1$ permutations $a, a_1, \dots, a_k \in S_m$.
Question: Are there $i_1, \dots, i_k \in \mathbb{N}$ such that $a = a_1^{i_1} \cdots a_k^{i_k}$?

Main results. Our main result for rational subset membership in symmetric groups is:

▶ **Theorem 1.6.** *Problems 1.1–1.4 and Problem 1.5 for $k \geq 3$ are* NP*-complete.*

In contrast, we will show that the 2-knapsack problem can be solved in polynomial time (Theorem 5.7). The NP upper bound for the rational subset membership problem will be shown for black-box groups.

▶ Remark 1.7. The abelian variant of the knapsack problem, i.e., Problem 1.3 with the additional restriction that the permutations $s_1, \dots, s_n$ pairwise commute is of course the abelian subgroup membership problem and hence belongs to NC.

▶ Remark 1.8. Analogously to the k-knapsack problem one might consider the k-subset sum problem, where the number n in Problem 1.2 is fixed to k and not part of the input. This problem can be solved in time $2^k \cdot m^{\mathcal{O}(1)}$ (check all 2^k assignments for exponents $i_1, \dots, i_k$) and hence in polynomial time for every fixed k.

Finally, for the context-free subset membership problem for symmetric groups we show:

▶ **Theorem 1.9.** *The context-free membership problem for symmetric groups is* PSPACE*-complete.*

If we restrict the class of context-free grammars in Theorem 1.9 we can improve the complexity to NP: A derivation tree of a context-free grammar is called *acyclic* if no nonterminal appears twice on a path from the root to a leaf. Hence, the height of an acyclic derivation tree is bounded by the number of nonterminals of the grammar. The *Horton-Strahler number* $\mathrm{hs}(t)$ of a binary tree t (introduced by Horton and Strahler in the context of hydrology [25, 40]; see [19] for a good survey emphasizing the importance of Horton-Strahler numbers in computer science) is recursively defined as follows: If t consists of a single node then $\mathrm{hs}(t) = 0$. Otherwise, assume that t_1 and t_2 are the subtrees rooted in the two children of the node. If $\mathrm{hs}(t_1) = \mathrm{hs}(t_2)$ then $\mathrm{hs}(t) = 1 + \mathrm{hs}(t_1)$, and if $\mathrm{hs}(t_1) \neq \mathrm{hs}(t_2)$ then $\mathrm{hs}(t) = \max\{\mathrm{hs}(t_1), \mathrm{hs}(t_2)\}$. For $k \geq 1$ let $\mathrm{CFG}(k)$ be the set of all context-free grammars in Chomsky normal form (hence, derivation trees are binary trees if we ignore the leaves labelled with terminal symbols) such that every acyclic derivation tree has Horton-Strahler number at most k.

▶ **Theorem 1.10.** *For every $k \geq 1$, the context-free membership problem for symmetric groups restricted to context-free grammars from $\mathrm{CFG}(k)$ is* NP*-complete.*

Table 1 Complexity of various intersection non-emptiness problems.

	no CFG	one CFG(k)	one CFG
DFAs	PSPACE-c. [30]	EXPTIME-c. for k large enough	EXPTIME-c. [41]
group DFAs	NP-c. [11]	NP-c. for all $k \geq 1$	PSPACE-c.

Theorem 1.10 generalizes the statement for rational subsets in Theorem 1.6: Every regular grammar (when brought into Chomsky normal form) belongs to CFG(1). Linear context-free grammars belong to CFG(1) as well. Note that Theorem 1.10 is a promise problem in the sense that coNP is the best upper bound for testing whether a given context-free grammar belongs to CFG(k) that we are aware of; see [33, Theorem A.2]. The upper bounds in Theorems 1.6, 1.9, and 1.10 will actually be shown for black box groups.

Application to intersection non-emptiness problems. We can apply Theorems 1.9 and 1.10 to intersection non-emptiness problems. The *intersection non-emptiness problem for deterministic finite automata* (DFAs) is the following problem:

▶ **Problem 1.11** (intersection non-emptiness problem for DFAs)**.**
Input: DFAs $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$
Question: Is $\bigcap_{1 \leq i \leq n} L(\mathcal{A}_i)$ *non-empty?*

Kozen [30] has shown that this problem is PSPACE-complete. When restricted to group DFAs (see Section 2) the intersection non-emptiness problem was shown to be NP-complete by Blondin et al. [11]. Based on Cook's characterization of EXPTIME by polynomially space bounded AuxPDAs [13], Swernofsky and Wehar [41] showed that the intersection non-emptiness problem is EXPTIME-complete for a list of general DFAs and a single context-free grammar; see also [24, p. 275] and see [18] for a related EXPTIME-complete problem. Using Theorems 1.9 and 1.10 we can easily show the following new results:

▶ **Theorem 1.12.** *The following problem is* NP*-complete for every* $k \geq 1$*:*

Input: A list of group DFAs $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ *and a context-free grammar* $\mathcal{G} \in \mathrm{CFG}(k)$*.*
Question: Is $L(\mathcal{G}) \cap \bigcap_{1 \leq i \leq n} L(\mathcal{A}_i)$ *non-empty?*

▶ **Theorem 1.13.** *The following problem is* PSPACE*-complete:*

Input: A list of group DFAs $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ *and a context-free grammar* $\mathcal{G}$*.*
Question: Is $L(\mathcal{G}) \cap \bigcap_{1 \leq i \leq n} L(\mathcal{A}_i)$ *non-empty?*

Table 1 gives an overview on the complexity of intersection non-emptiness problems. For the intersection non-emptiness problem for arbitrary DFAs and one grammar from CFG(k) one has to notice that the EXPTIME-hardness proof from [41] works for a fixed context-free grammar. Moreover, every fixed context-free grammar belongs to CFG(k) for some $k \geq 1$.

Related work. Computationally problems for permutation groups have a long history (see e.g. the text book [38]), and have applications, e.g. for graph isomorphism testing [3, 35]. A problem that is similar to subset sum is the *minimum generator sequence problem* (MGS) [20]: The input consists of unary encoded numbers m, ℓ and a list of permutations $a, a_1, \ldots, a_n \in S_m$. The question is, whether a can be written as a product $b_1 b_2 \cdots b_k$ with $k \leq \ell$ and $b_1, \ldots, b_k \in \{a_1, \ldots, a_n\}$. The problem MGS was shown to be NP-complete in [20]. For the case, where the number ℓ is given in binary representation, the problem is PSPACE-complete [27]. This yields in fact the PSPACE-hardness in Theorem 1.9.

Intersection nonemptiness problems have been studied intensively in recent years, see e.g. [1, 14]. The papers [10, 26] prove PSPACE-hardness of the intersection nonemptiness problem for inverse automata (DFAs, where the transition monoid is an inverse monoid).

There are several algorithms for context-free (and other) grammars that exploit that an input grammar is *index-m*, meaning that *all* derivation trees have Horton-Strahler number $\leq m$, where m is part of the input in unary notation [17, 12, 16, 15, 7, 2] (see [19] for a survey). This assumption, called *finite-index*, is incomparable to our CFG(k)-assumption: In CFG(k), the k is fixed but one only considers acyclic derivation trees.

2 Preliminaries

Groups. Let G be a finite group and let G^* be the free monoid of all finite words over the alphabet G. There is a canonical morphism $\phi_G : G^* \to G$ that is the identity mapping on $G \subseteq G^*$. Throughout this paper we suppress applications of ϕ_G and identify words over the alphabet G with the corresponding group elements. For a subset $S \subseteq G$ we denote with $\langle S \rangle$ the subgroup generated by S. The following folklore lemma is a straightforward consequence of Lagrange's theorem (if A and B are subgroups of G with $A < B$, then $|B| \geq 2 \cdot |A|$).

▶ **Lemma 2.1.** *Let G be a finite group and $S \subseteq G$ a generating set for G. Then, there exists a subset $S' \subseteq S$ such that $\langle S' \rangle = G$ and $|S'| \leq \log_2 |G|$.*

Assume that $G = \langle S \rangle$. A *straight-line program* over the generating set S is a sequence of definitions $\mathcal{S} = (x_i := r_i)_{1 \leq i \leq n}$ where the x_i are variables and every right-hand side r_i is either from S or of the form $x_j x_k$ with $1 \leq j, k < i$. Every variable x_i evaluates to a group element $g_i \in G$ in the obvious way: if $r_i \in S$ then $g_i = r_i$ and if $r_i = x_j x_k$ then $g_i = g_j g_k$. Then, $\mathcal{S}$ is said to *produce* g_n. The size of $\mathcal{S}$ is n. The following result is known as the reachability theorem [6, Theorem 3.1].

▶ **Theorem 2.2** (reachability theorem). *Let G be a finite group, $S \subseteq G$ a generating set for G, and $g \in G$. Then there exists a straight-line program over S of size at most $(1 + \log_2 |G|)^2$ that produces the element g.*

For a set Q let S_Q be the *symmetric group* on Q, i.e., the set of all permutations on Q with composition of permutations as the group operation. If $Q = \{1, \ldots, m\}$ we also write S_m for S_Q. Let $a \in S_Q$ be a permutation and let $q \in Q$. We also write q^a for $a(q)$. We multiply permutations from left to right, i.e., for $a, b \in S_Q$, ab is the permutation with $q^{ab} = (q^a)^b$ for all $q \in Q$. A *permutation group* is a subgroup of some S_Q.

Quite often, the permutation groups we consider are actually direct products $\prod_{1 \leq i \leq d} S_{m_i}$ for small numbers m_i. Clearly, we have $\prod_{1 \leq i \leq d} S_{m_i} \leq S_m$ for $m = \sum_{1 \leq i \leq d} m_i$ and an embedding of $\prod_{1 \leq i \leq d} S_{m_i}$ into S_m can be computed in polynomial time.

Horton-Strahler number. Recall the definition of the Horton-Strahler number $\mathrm{hs}(t)$ of binary trees t from the introduction. We need the following simple fact; see [33, Lemma 2.3]. Here, the *height* of a tree is the maximal number of edges on a path from the root to a leaf.

▶ **Lemma 2.3.** *Let t be a binary tree of height d and let $s = \mathrm{hs}(t)$. Then, t has at most d^s many leaves and therefore at most $2 \cdot d^s$ many nodes.*

Formal languages. We assume that the reader is familiar with basic definitions from automata theory. Our definitions of deterministic finite automata (DFA), nondeterministic finite automata (NFA), and context-free grammars are the standard ones.

Consider a DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition mapping and $F \subseteq Q$ is the set of final states. The *transformation monoid* of $\mathcal{A}$ is the submonoid of Q^Q (the set of all mappings on Q and composition of functions as the monoid operation) generated by all mappings $q \mapsto \delta(q, a)$ for $a \in \Sigma$. A *group DFA* is a DFA whose transformation monoid is a group.

Context-free grammars will be always in Chomsky normal form. When we speak of a *derivation tree* of a context-free grammar, we always assume that the root of the tree is labelled with the start nonterminal and every leaf is labelled with a terminal symbol. When we talk about the Horton-Strahler number of such a tree, we remove all terminal-labelled leaves so that the resulting tree is a binary tree (due to the Chomsky normal form). In a *partial derivation tree*, we also allow leaves labelled with nonterminals (but we still assume that the root is labelled with the start nonterminal).

3 Black box groups

More details on black box groups can be found in [6, 38]. Roughly speaking, in the black box setting group elements are encoded by bit strings of a certain length b and there exist oracles for multiplying two group elements, computing the inverse of a group element, checking whether a given group element is the identity, and checking whether a given bit string of length b is a valid encoding of a group element (the latter operation is not allowed in [6]). As usual, each execution of an oracle operation counts one time unit, but the parameter b enters the input length additively.

Formally, a *black box* is a tuple $B = (b, c, \mathsf{valid}, \mathsf{inv}, \mathsf{prod}, \mathsf{id}, G, f)$ such that G is a finite group (the *group in the box*), $b, c \in \mathbb{N}$, and the following properties hold:

- $f : \{0,1\}^b \to G \uplus \{*\}$ satisfies $G \subseteq f(\{0,1\}^b)$. Here, $f^{-1}(g)$ is the set of *names* of $g \in G$.
- $\mathsf{valid} : \{0,1\}^b \to \{\mathsf{yes}, \mathsf{no}\}$ is such that $\forall x \in \{0,1\}^b : f(x) \in G \iff \mathsf{valid}(x) = \mathsf{yes}$.
- $\mathsf{inv} : \{0,1\}^b \to \{0,1\}^b$ is such that $\forall x \in f^{-1}(G) : f(\mathsf{inv}(x)) = f(x)^{-1}$.
- $\mathsf{prod} : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}^b$ is such that $\forall x, y \in f^{-1}(G) : f(\mathsf{prod}(x, y)) = f(x)f(y)$.
- $\mathsf{id} : \{0,1\}^b \times \{0,1\}^c \to \{\mathsf{yes}, \mathsf{no}\}$ is such that for all $x \in f^{-1}(G)$: $f(x) = 1$ iff there exists $y \in \{0,1\}^c$ with $\mathsf{id}(x, y) = \mathsf{yes}$ (such a y is called a *witness* for $f(x) = 1$).

We call b the *code length* of the black box.

A *black box Turing machine* is a deterministic or nondeterministic oracle Turing machine T that is equipped with oracles for the four operations valid, inv, prod, and id. The input for T consists of two unary encoded numbers b and c and some additional problem specific input. In order to determine the behavior of T, it must be coupled with a black box $B = (b, c, \mathsf{valid}, \mathsf{inv}, \mathsf{prod}, \mathsf{id}, G, f)$ (where b and c must match the first part of the input of T). Then, given bit strings $x, y \in \{0,1\}^b$ on a special tape of T, it can compute instantaneously in a single computation step the bit string $\mathsf{prod}(x, y)$. Analogous instantaneous computations can be done for the operations valid, inv, and id. We denote with T_B the machine T coupled with the black box B. Note that the black box $B = (b, c, \mathsf{valid}, \mathsf{inv}, \mathsf{prod}, \mathsf{id}, G, f)$ is not part of the input of T, only the unary encoded numbers b and c are part of the input.

Assume that $\mathcal{P}$ is an algorithmic problem for finite groups. The input for $\mathcal{P}$ is a finite group G and some additional data X (e.g. a context-free grammar with terminal alphabet G in the next section). We do not specify exactly, how G is represented. The additional input X may contain elements of G. We will say that $\mathcal{P}$ belongs to NP for black box groups if there is a nondeterministic black box Turing machine T, whose input is of the form b, c, X with unary encoded numbers b and c, such that for every black box $B = (b, c, \mathsf{valid}, \mathsf{inv}, \mathsf{prod}, \mathsf{id}, G, f)$ the following holds: T_B accepts the input b, c, X (where X denotes the additional input for $\mathcal{P}$ and group elements in X are represented by bit strings from $f^{-1}(G)$) if and only if (G, X) belongs to $\mathcal{P}$. The running time of T_B is polynomial in $b + c + |X|$. Note that since G may have order 2^b, the order of G may be exponential in the input length. We will use the analogous definition for other complexity classes, in particular for PSPACE.

In the rest of the paper we handle black box groups in a slightly more casual way. We identify bits strings from $x \in f^{-1}(G)$ with the corresponding group elements. So, we will never talk about bit strings $x \in f^{-1}(G)$, but instead directly deal with elements of G. The reader should notice that we cannot directly verify whether a given element $g \in G$ is the identity. This is only possible in a nondeterministic way by guessing a witness $y \in \{0,1\}^c$. The same applies to the verification of an identity $g = h$, which is equivalent to $gh^{-1} = 1$. This allows to cover also quotient groups by the black box setting; see [6].

We need the following well-known fact from [6]:

▶ **Lemma 3.1.** *The subgroup membership problem for black box groups (given group elements $g, g_1, \dots, g_n$, does $g \in \langle g_1, \dots, g_n \rangle$ hold?) belongs to* NP.

This is a consequence of the reachability theorem: Let b be the code length of the black box. Hence there are at most 2^b group elements. By the reachability theorem (Theorem 2.2) it suffices to guess a straight-line program over $\{g_1, \dots, g_n\}$ of size at most $(1 + \log_2 2^b)^2 = (b+1)^2$, evaluate it using the oracle for prod (let g' be the result of the evaluation) and check whether $g'g^{-1} = 1$. The later can be done nondeterministically using the oracle for id.

4 Context-free membership in black box groups

In this section, we sketch the proofs for the following two results.

▶ **Theorem 4.1.** *The context-free membership problem for black box groups is in* PSPACE.

▶ **Theorem 4.2.** *For every $k \geq 1$, the context-free membership problem for black box groups restricted to context-free grammars from* $\mathrm{CFG}(k)$ *is in* NP.

Recall the definition of the class $\mathrm{CFG}(k)$ from the introduction. Let us first derive some corollaries. Theorem 4.2 directly implies Theorem 1.10. Applied to regular grammars (which are in $\mathrm{CFG}(1)$ after bringing them to Chomsky normal form), it yields:

▶ **Corollary 4.3.** *The rational subset membership problem for black box groups is in* NP. *In particular, the rational subset membership problem for symmetric groups is in* NP.

Also Theorem 1.9 can be easily obtained now: The upper bound follows directly from Theorem 4.1. The lower bound can be obtained from a result of Jerrum [27]. In the introduction we mentioned that Jerrum proved the PSPACE-completeness of the MGS problem for the case where the number ℓ is given in binary notation. Given permutations $a_1, \dots, a_n \in S_m$ and a binary encoded number ℓ one can easily construct a context-free grammar for $\{1, a_1, \dots, a_n\}^\ell \subseteq S_m$. Hence, the PSPACE-hard MGS problem with ℓ given in binary notation reduces to the context-free membership problem for symmetric groups. This reduction shows that Theorem 1.9 still holds if the input grammar is index-m (see the end of the introduction) with m being part of the input in unary notation.

The rest of the section is devoted to the upper bounds in Theorems 4.1 and 4.2. Detailed proofs of all lemmas can be found in the long version [33].

Outline. A straightforward algorithm for the context-free membership problem constructs an exponential-sized pushdown automaton and check the latter for emptiness. This would yield an EXPTIME upper bound. Hence, the difficulty is to achieve this in PSPACE. To this end, we exploit the fact that every subgroup of a black box group can be stored in polynomial space. Since context-free subsets of a finite group G are not necessarily subgroups, we need to find a way to encode information about derivations, as subgroups. We do this as follows. Recall that $\phi_G : G^* \to G$ is the canonical morphism from Section 2. For a context-free grammar $\mathcal{G}$ with terminal alphabet G and a nonterminal A we define

$$G_A = \{(\phi_G(u), \phi_G(v)) \mid u, v \in G^*,\ A \Rightarrow^*_{\mathcal{G}} uAv\}.$$

This is a subgroup of $G \times \hat{G}$, where $\hat{G}$ is the *dual group* of G: It has the same underlying set as G and if $g \cdot h$ is the product in G then the multiplication $\circ$ in $\hat{G}$ is defined by $g \circ h = h \cdot g$. A black box for G easily yields a black box for $G \times \hat{G}$. Thus, G_A and its subgroups have polynomial-size generating sets. When given generating sets for all G_A, one can check membership in $L(\mathcal{G})$ in PSPACE. Therefore, the idea is to compute G_A by saturating underapproximating subgroups H_A of G_A: Initially, H_A is the trivial subgroup of $G \times \hat{G}$ for each A. At each step, the groups H_A yield underapproximations of the sets $L(A)$ of all $w \in G^*$ derivable from a nonterminal A (as usual, we identify $L(A)$ with $\phi_G(L(A))$). The underapproximation of $L(A)$, in turn, is used to enlarge the subgroups H_A, yielding a better underapproximation of the G_A. Finally, we obtain the entire groups G_A.

The operations Δ and Γ. We fix a finite group G that is only accessed via a black box. We now define the operations Δ and Γ, which turn underapproximations of G_A into underapproximations of $L(A)$ (or vice versa, resp.). Let $\mathcal{G} = (N, G, P, S)$ be a context-free grammar in Chomsky normal form that is part of the input, whose terminal alphabet is the finite group G. When we speak of the input size in the following, we refer to $|\mathcal{G}| + b + c$, where b and c are the two unary encoded numbers from the black box for G and the size $|\mathcal{G}|$ is defined as the number of productions of the grammar.

Recall from the introduction that a derivation tree T is acyclic if a nonterminal does not occur twice on a path from the root to a leaf of T. The height of an acyclic derivation tree is at most $|N|$. We now define two operations Δ and Γ. The operation Δ maps a tuple $s = (H_A)_{A \in N}$ of subgroups $H_A \le G \times \hat{G}$ to a tuple $\Delta(s) = (L_A)_{A \in N}$ of subsets $L_A \subseteq G$ (not necessarily subgroups), whereas Γ maps a tuple $t = (L_A)_{A \in N}$ of subsets $L_A \subseteq G$ to a tuple $\Gamma(t) = (H_A)_{A \in N}$ of subgroups $H_A \le G \times \hat{G}$.

We start with Δ. Let $s = (H_A)_{A \in N}$ be a tuple of subgroups $H_A \le G \times \hat{G}$. The tuple $\Delta(s) = (L_A)_{A \in N}$ of subsets $L_A \subseteq G$ is obtained as follows: Let T be an acyclic derivation tree with root r labelled by $A \in N$. We assign inductively a set $L_v \subseteq G$ to every inner node v: Let B the label of v. If v has only one child it must be a leaf since our grammar is in Chomsky normal form. Let $g \in G$ be the label of this leaf. Then we set $L_v = \{h_1 g h_2 \mid (h_1, h_2) \in H_B\}$. If v has two children v_1, v_2 (where v_1 is the left child and v_2 the right child), then the sets $L_{v_1} \subseteq G$ and $L_{v_2} \subseteq G$ are already determined and we set

$$L_v = \{h_1 g_1 g_2 h_2 \mid (h_1, h_2) \in H_B, g_1 \in L_{v_1}, g_2 \in L_{v_2}\}.$$

We set $L(T) = L_r$ and finally define L_A as the union of all sets $L(T)$ where T is an acyclic derivation whose root is labelled with A.

The second operation Γ is defined as follows: Let $t = (L_A)_{A \in N}$ be a tuple of subsets $L_A \subseteq G$. Then we define the tuple $\Gamma(t) = (H_A)_{A \in N}$ with $H_A \le G \times \hat{G}$ as follows: Fix a nonterminal $A \in N$. Consider a sequence $p = (A_i \to A_{i,0} A_{i,1})_{1 \le i \le m}$ of productions

$(A_i \to A_{i,0}A_{i,1}) \in P$ and a sequence $d = (d_i)_{1 \le i \le m}$ of directions $d_i \in \{0,1\}$ such that $A_{i+1} = A_{i,d_i}$ for all $1 \le i \le m$, $A_1 = A = A_{m,d_m}$. Basically, p and d define a path from A back to A. For every $1 \le i \le m$ we define the sets

$$M_i = \begin{cases} L_{A_{i,0}} \times \{1\} \text{ if } d_i = 1 \\ \{1\} \times L_{A_{i,1}} \text{ if } d_i = 0 \end{cases}$$

We view M_i as a subset of $G \times \hat{G}$ and define $M(p,d) = M_1 \cdot \ldots \cdot M_m$, in which $\cdot$ refers to the product in $G \times \hat{G}$. If p and d are the empty sequences ($m = 0$) then $M(p,d) = \{(1,1)\}$. Finally we define H_A as the set of all $M(p,d)$, where $p = (A_i \to A_{i,0}A_{i,1})_{1 \le i \le m}$ and $d = (d_i)_{1 \le i \le m}$ are as above (including the empty sequences). This set H_A is a subgroup of $G \times \hat{G}$. To see this, it suffices to argue that H_A is a monoid. The latter follows from the fact that two pairs of sequences (p,d) and (p',d') of the above form can be composed to a single pair (pp',dd').

In the following, we will speak of NP algorithms with oracles. Here, we mean nondeterministic polynomial-time Turing machines with oracles. However, the oracle can only be queried positively: There is an instruction that succeeds if the oracle answers "yes", but cannot be executed if the oracle would answer "no". This implies that if there is an NP (resp. PSPACE algorithm) for the oracle queries, there exists an NP (resp. PSPACE) algorithm for the entire problem. Likewise, we will use the notion of oracle PSPACE algorithms.

▶ **Lemma 4.4.** *For tuples $t = (L_A)_{A \in N}$, there is an* NP *algorithm for membership to the entries of $\Gamma(t)$, with access to an oracle for the entries of t.*

The main idea for the proof is to represent the entries of $\Gamma(t)$ (which are subgroups of $G \times \hat{G}$) by NFAs over the alphabet $G \times \hat{G}$ (the NFA will not be given explicitly to us) and to guess generating sets for the subgroups in $\Gamma(t)$ using the so-called spanning tree approach. Finally, one checks membership in the entries of $\Gamma(t)$ using Lemma 3.1.

▶ **Lemma 4.5.** *Assume that the input grammar $\mathcal{G}$ is restricted to the class* CFG(k) *for some fixed k. For tuples $s = (H_A)_{A \in N}$ of subgroups $H_A \le G \times \hat{G}$, there exists an* NP *algorithm for membership to entries in $\Delta(s)$, with access to an oracle for membership to each entry of s.*

To prove this lemma, we guess an acyclic derivation tree of $\mathcal{G}$; by Lemma 2.3 its size is bounded by $2|N|^k$. Moreover, for every internal B-labelled node v (with $B \in N$) we guess a pair $(h_{v,1}, h_{v,2}) \in G \times \hat{G}$ and verify in NP that $(h_{v,1}, h_{v,2}) \in H_A$. Then we evaluate the derivation tree in the right way (following the definition of $\Delta(s)$) and check that we obtain the the input group element that we want to test for membership.

If the input grammar $\mathcal{G}$ is not restricted to a class CFG(k) for some k then we can still prove the following PSPACE-version of Lemma 4.5.

▶ **Lemma 4.6.** *For tuples $s = (H_A)_{A \in N}$ of subgroups $H_A \le G \times \hat{G}$, there is a* PSPACE *algorithm for membership to $\Delta(s)$, using an oracle for membership to the entries of s.*

For the proof we can no longer guess an acyclic derivation tree, since it may have exponential size. Instead, we guess the derivation tree in an incremental way and store on a stack of size at most $|N|$ (the maximal height of an acyclic derivation tree) the information that is needed in order to evaluate the derivation tree.

The following lemma is a straightforward consequence of Lemma 2.1:

▶ **Lemma 4.7.** *For tuples $s = (H_A)_{A \in N}$ subgroups $H_A \le G \times \hat{G}$, there exists an* NP *algorithm, with access to an oracle for membership to each entry of s, with the following properties:*

- *On every computation path the machine outputs a tuple $(S_A)_{A \in N}$ of subsets $S_A \subseteq H_A$.*
- *There is at least one computation path on which the machine outputs a tuple $(S_A)_{A \in N}$ such that every S_A generates H_A.*

▶ **Lemma 4.8.** $\Delta((G_A)_{A\in N}) = (L(A))_{A\in N}$.

Let $s_0 = (H_A)_{A\in N}$ with $H_A = \{(1,1)\}$ for all $A \in N$ be the tuple of trivial subgroups of $G \times \hat{G}$. For two tuples $s_1 = (H_{A,1})_{A\in N}$ and $s_2 = (H_{A,2})_{A\in N}$ of subgroups of $G \times \hat{G}$ we write $s_1 \leq s_2$ if $H_{A,1} \leq H_{A,2}$ for every $A \in N$. Since Γ and Δ are monotone w.r.t. component-wise inclusion, we have $(\Gamma\Delta)^i(s_0) \leq (\Gamma\Delta)^{i+1}(s_0)$ for all i. We can thus define $\lim_{i\to\infty}(\Gamma\Delta)^i(s_0)$.

▶ **Lemma 4.9.** *$(G_A)_{A\in N} = \lim\limits_{i\to\infty}(\Gamma\Delta)^i(s_0) = (\Gamma\Delta)^j(s_0)$ where $j = 2 \cdot |N| \cdot \lfloor \log_2 |G| \rfloor$.*

Let us briefly sketch the proof of this lemma. From the definition of Γ and Δ we easily obtain $(\Gamma\Delta)^i(s_0) \leq (G_A)_{A\in N}$ by induction on $i \geq 0$. To show $(G_A)_{A\in N} \leq \lim_{i\to\infty}(\Gamma\Delta)^i(s_0)$, we define $(H_A)_{A\in N} = \lim_{i\to\infty}(\Gamma\Delta)^i(s_0)$ and take a pair $(g,h) \in G_A$. Hence, there exists a derivation $A \Rightarrow_G^* uAv$ such that $g = \phi_G(u)$ and $h = \phi_G(v)$. One can prove $(g,h) \in H_A$ by an induction on the length of this derivation. This shows the identity $(G_A)_{A\in N} = \lim\limits_{i\to\infty}(\Gamma\Delta)^i(s_0)$.

For the second identity in Lemma 4.8 note that since all G_A are finite groups there is a smallest number $j \geq 0$ such that $(\Gamma\Delta)^j(s_0) = (\Gamma\Delta)^{j+1}(s_0)$. We then have $(\Gamma\Delta)^j(s_0) = \lim_{i\to\infty}(\Gamma\Delta)^i(s_0)$. It remains to show that $j \leq 2 \cdot |N| \cdot \log_2 |G|$. In each component of the $|N|$-tuples $(\Gamma\Delta)^i(s_0)$ $(0 \leq i \leq j)$ we have a chain of subgroups of $G \times \hat{G}$. By Lagrange's theorem, any chain $\{(1,1)\} = H_0 < H_1 < \cdots < H_{k-1} < H_k \leq G \times \hat{G}$ satisfies $k \leq 2 \cdot \log_2 |G|$. This shows that $j \leq 2 \cdot |N| \cdot \log_2 |G|$.

Proofs of Theorem 4.1 and 4.2. We start with Theorem 4.2. By Lemmas 4.5 and 4.8 it suffices to decide membership to G_A in NP. For this, we construct a nondeterministic polynomial time machine that computes on every computation path a subset $S_A \subseteq G_A$ for every $A \in N$ such that on at least one computation path it computes a generating set for groups G_A for all $A \in N$. Then, by Lemma 3.1, membership for each $\langle S_A \rangle$ is in NP.

We compute S_A by initializing $S_A = \{(1,1)\}$ for every $A \in N$ and then performing $2 \cdot |N| \cdot \log_2 |G|$ iterations of the following procedure: Suppose we have already produced the subsets $(S_A)_{A\in N}$. Membership in $\langle S_A \rangle$ can be decided in NP by Lemma 3.1. Hence, by Lemmas 4.4 and 4.5 one can decide membership in every entry of the tuple $\Gamma(\Delta((\langle S_A \rangle)_{A\in N}))$ in NP. Finally, by Lemma 4.7 we can produce nondeterministically in polynomial time a subset $S'_A \subseteq G \times \hat{G}$ for every $A \in N$ such that for every computation path we have $(\langle S'_A \rangle)_{A\in N} \leq \Gamma(\Delta((\langle S_A \rangle)_{A\in N}))$ and for at least one computation path the machine produces subsets S'_A with $(\langle S'_A \rangle)_{A\in N} = \Gamma(\Delta((\langle S_A \rangle)_{A\in N}))$. With the sets S'_A we go into the next iteration. By Lemma 4.9 there will be at least one computation path on which after $2 \cdot |N| \cdot \log_2 |G|$ iterations we get a generating set for the entire groups G_A. This concludes Theorem 4.2. The proof of Theorem 4.1 only differs by using Lemma 4.6 instead of Lemma 4.5. ◀

5 Subset sum and knapsack in symmetric groups

In this section, we want to contrast the general upper bounds from the previous sections with lower bounds for symmetric groups and restricted versions of the rational subset membership problem. We start with the subset sum problem. The following result refers to the abelian group $\mathbb{Z}_3^m$, for which we use the additive notation.

▶ **Theorem 5.1.** *The following problem is* NP*-hard:*
Input: unary encoded number m and a list of group elements $g, g_1, \ldots, g_n \in \mathbb{Z}_3^m$.
Question: Are there $i_1, \ldots, i_n \in \{0,1\}$ such that $g = \sum_{1\leq k\leq n} i_k \cdot g_k$?

The proof uses a straightforward reduction from the *exact 3-hitting set problem* (X3HS):

▶ **Problem 5.2** (X3HS).
Input: a finite set A and a set $\mathcal{B} \subseteq 2^A$ of subsets of A, all of size 3.
Question: Is there a subset $A' \subseteq A$ such that $|A' \cap C| = 1$ for all $C \in \mathcal{B}$?

X3HS is the same problem as positive 1-in-3-SAT, which is NP-complete [23, Problem LO4]. The reduction of X3HS to the problem from Theorem 5.1 can be found in [33, Theorem 5.1]. Since $\mathbb{Z}_3^m \leq S_{3m}$ we obtain the following corollary:

▶ **Corollary 5.3.** *The abelian subset sum problem for symmetric groups is* NP*-hard.*

Let us remark that the subset sum problem for $\mathbb{Z}_2^m$ (with m part of the input) is equivalent to the subgroup membership problem for $\mathbb{Z}_2^m$ (since every element of $\mathbb{Z}_2^m$ has order two) and therefore can be solved in polynomial time.

We now come to the knapsack problem in permutation groups. NP-hardness of the general version of knapsack can be easily deduced from a result of Luks: Recall from the introduction that Luks [36] proved NP-completeness of 3-membership for the special case of membership in a product GHG where G and H are abelian subgroups of S_m. Let $g_1, g_2, \ldots, g_k$ be the given generators of G and let $h_1, h_2, \ldots, h_l$ be the given generators of H. Then $s \in GHG$ is equivalent to the solvability of the following knapsack equation:

$$s = g_1^{x_1} g_2^{x_2} \cdots g_k^{x_k} h_1^{y_1} h_2^{y_2} \cdots h_l^{y_l} g_1^{z_1} g_2^{z_2} \cdots g_k^{z_k}.$$

We next want to prove that already 3-knapsack is NP-hard. In other words: the k-membership problem is NP-hard for every $k \geq 3$ even if the groups are cyclic.

Let $p > 0$ be an integer. For the rest of the section we write $[p]$ for the cycle $(1, 2, \ldots, p)$ mapping p to 1 and i to $i + 1$ for $1 \leq i \leq p - 1$. The proofs for the following two lemmas are available in the full version [33].

▶ **Lemma 5.4.** *Let $p, q \in \mathbb{N}$ such that q is odd and $p > q > 0$ holds. Then the products $[p][q]$ and $[q][p]$ are cycles of length p.*

▶ **Lemma 5.5.** *Let $p, q \in \mathbb{N}$ be primes such that $2 < q < p$ holds. Then*

$$[p]^{-x_2}[q]^{x_1}([p][q])^{x_2} = [q] = [q]^{x_1}[p]^{-x_2}([p][q])^{x_2} \tag{1}$$

if and only if ($x_1 \equiv 1 \bmod q$ and $x_2 \equiv 0 \bmod p$) or ($x_1 \equiv 0 \bmod q$ and $x_2 \equiv 1 \bmod p$).

We now come to our main result for the knapsack problem:

▶ **Theorem 5.6.** *The problem 3-knapsack for symmetric groups is* NP*-hard.*

Proof. To prove the theorem we give a log-space reduction from the NP-complete problem X3HS (Problem 5.2) to 3-knapsack. Let A be a finite set and $\mathcal{B} \subseteq 2^A$ such that every $C \in \mathcal{B}$ has size 3. W.l.o.g.let $A = \{1, \ldots, m\}$ and let $\mathcal{B} = \{C_1, C_2, \ldots, C_d\}$ where $C_i = \{\alpha(i,1), \alpha(i,2), \alpha(i,3)\}$ for a mapping $\alpha : \{1, \ldots, d\} \times \{1, 2, 3\} \to \{1, \ldots, m\}$.

Let $p_1, \ldots, p_m, r_1, \ldots, r_m, q_1, \ldots, q_d$ be the first $2m + d$ odd primes such that $p_j > r_j > 2$ and $p_j > q_i > 2$ for $1 \leq i \leq d$ and $1 \leq j \leq m$ hold. Moreover let $P = \max_{1 \leq j \leq m} p_j$. Intuitively, the primes p_j and r_j ($1 \leq j \leq m$) belong to $j \in A$ and the prime q_i ($1 \leq i \leq d$) belongs to the set C_i. We will work in the group

$$G = \prod_{j=1}^{m} \mathcal{V}_j \times \prod_{i=1}^{d} \mathcal{C}_i,$$

where $\mathcal{V}_j \leq S_{4p_j+r_j}$ and $\mathcal{C}_i \leq S_{q_i+3P}$. More precisely we have

$$\mathcal{V}_j = S_{p_j} \times S_{p_j} \times \mathbb{Z}_{p_j} \times \mathbb{Z}_{p_j} \times \mathbb{Z}_{r_j} \text{ and } \mathcal{C}_i = \mathbb{Z}_{q_i} \times S_P \times S_P \times S_P.$$

In the following, we denote the identity element of a symmetric group S_m with id in order to not confuse it with the generator of a cyclic group $\mathbb{Z}_m$.

We now define four group elements $g, g_1, g_2, g_3 \in G$. We write $g = (v_1, \ldots, v_m, c_1, \ldots c_d)$ and $g_k = (v_{k,1}, \ldots, v_{k,m}, c_{k,1}, \ldots, c_{k,d})$ with $v_j, v_{k,j} \in \mathcal{V}_j$ and $c_i, c_{k,i} \in \mathcal{C}_i$. These elements are defined as follows:

$$\begin{array}{rclllll}
v_j & = & ([r_j], & [r_j], & 0, & 0, & 0) \\
v_{1,j} & = & ([r_j], & [p_j]^{-1}, & 1, & 1, & 1) \\
v_{2,j} & = & ([p_j]^{-1}, & [r_j], & -1, & 0, & -1) \\
v_{3,j} & = & ([p_j][r_j], & [p_j][r_j], & 0, & -1, & 0)
\end{array}
\qquad
\begin{array}{rcllll}
c_i & = & (1, & \mathsf{id}, & \mathsf{id}, & \mathsf{id}) \\
c_{1,i} & = & (1, & [q_i]^{-1}, & [p_{\alpha(i,2)}]^{-1}, & [q_i][p_{\alpha(i,3)}]) \\
c_{2,i} & = & (1, & [q_i][p_{\alpha(i,1)}], & [q_i]^{-1}, & [p_{\alpha(i,3)}]^{-1}) \\
c_{3,i} & = & (1, & [p_{\alpha(i,1)}]^{-1}, & [q_i][p_{\alpha(i,2)}], & [q_i]^{-1})
\end{array}$$

We claim that there is a subset $A' \subseteq A$ such that $|A' \cap C_i| = 1$ for every $1 \leq i \leq d$ if and only if there are $z_1, z_2, z_3 \in \mathbb{Z}$ with $g = g_1^{z_1} g_2^{z_2} g_3^{z_3}$ in the group G. Due to the direct product decomposition of G and the above definition of g, g_1, g_2, g_3, the statement $g = g_1^{z_1} g_2^{z_2} g_3^{z_3}$ is equivalent to the conjunctions of the following statements (read the above definitions of the $v_j, v_{k,j}, c_i, c_{k,i}$ columnwise) for all $1 \leq j \leq m$ and $1 \leq i \leq d$:

(a) $[r_j] = [r_j]^{z_1}[p_j]^{-z_2}([p_j][r_j])^{z_3}$
(b) $[r_j] = [p_j]^{-z_1}[r_j]^{z_2}([p_j][r_j])^{z_3}$
(c) $z_1 \equiv z_2 \bmod p_j$
(d) $z_1 \equiv z_3 \bmod p_j$
(e) $z_1 \equiv z_2 \bmod r_j$
(f) $1 \equiv z_1 + z_2 + z_3 \bmod q_i$
(g) $\mathsf{id} = [q_i]^{-z_1}([q_i][p_{\alpha(i,1)}])^{z_2}[p_{\alpha(i,1)}]^{-z_3}$
(h) $\mathsf{id} = [p_{\alpha(i,2)}]^{-z_1}[q_i]^{-z_2}([q_i][p_{\alpha(i,2)}])^{z_3}$
(i) $\mathsf{id} = ([q_i][p_{\alpha(i,3)}])^{z_1}[p_{\alpha(i,3)}]^{-z_2}[q_i]^{-z_3}$

Recall that by Lemma 5.4, $[p_j][r_j]$ and $[q_i][p_j]$ are cycles of length p_j. Due to the congruences in (c), (d), and (e), the conjunction of (a)–(i) is equivalent to the conjunction of (j)–(p):

(j) $z_1 \equiv z_2 \equiv z_3 \bmod p_j$
(k) $z_1 \equiv z_2 \bmod r_j$
(l) $[p_j]^{-z_1}[r_j]^{z_2}([p_j][r_j])^{z_1} = [r_j] = [r_j]^{z_2}[p_j]^{-z_1}([p_j][r_j])^{z_1}$
(m) $1 \equiv z_1 + z_2 + z_3 \bmod q_i$
(n) $\mathsf{id} = [q_i]^{-z_1}([q_i][p_{\alpha(i,1)}])^{z_1}[p_{\alpha(i,1)}]^{-z_1}$
(o) $\mathsf{id} = [p_{\alpha(i,2)}]^{-z_1}[q_i]^{-z_2}([q_i][p_{\alpha(i,2)}])^{z_1}$
(p) $\mathsf{id} = ([q_i][p_{\alpha(i,3)}])^{z_1}[p_{\alpha(i,3)}]^{-z_1}[q_i]^{-z_3}$

By Lemma 5.5, the conjunction of (j)–(p) is equivalent to the conjunction of (q)–(u):

(q) ($z_1 \equiv z_2 \equiv z_3 \equiv 0 \bmod p_j$ and $z_1 \equiv z_2 \equiv 1 \bmod r_j$) or
($z_1 \equiv z_2 \equiv z_3 \equiv 1 \bmod p_j$ and $z_1 \equiv z_2 \equiv 0 \bmod r_j$)
(r) $1 \equiv z_1 + z_2 + z_3 \bmod q_i$
(s) $\mathsf{id} = [q_i]^{-z_1}([q_i][p_{\alpha(i,1)}])^{z_1}[p_{\alpha(i,1)}]^{-z_1}$
(t) $\mathsf{id} = [p_{\alpha(i,2)}]^{-z_1}[q_i]^{-z_2}([q_i][p_{\alpha(i,2)}])^{z_1}$
(u) $\mathsf{id} = ([q_i][p_{\alpha(i,3)}])^{z_1}[p_{\alpha(i,3)}]^{-z_1}[q_i]^{-z_3}$

Let us now assume that $A' \subseteq A$ is such that $|A' \cap C_i| = 1$ for every $1 \leq i \leq d$. Let $\sigma : \{1, \ldots, m\} \to \{0, 1\}$ such that $\sigma(j) = 1$ iff $j \in S'$. Thus, $\alpha(i,1) + \alpha(i,2) + \alpha(i,3) = 1$ for all $1 \leq i \leq d$. By the Chinese remainder theorem, we can set $z_1, z_2, z_3 \in \mathbb{Z}$ such that

- $z_1 \equiv z_2 \equiv z_3 \equiv \sigma(j) \bmod p_j$ and $z_1 \equiv z_2 \equiv 1 - \sigma(j) \bmod r_j$ for $1 \leq j \leq m$,
- $z_k \equiv \sigma(\alpha(i,k)) \bmod q_i$ for $1 \leq i \leq d$ and $1 \leq k \leq 3$.

Then (q) and (r) hold. For (s), one has to consider two cases: if $\sigma(\alpha(i,1)) = 0$, then $z_1 \equiv 0 \bmod q_i$ and $z_1 \equiv 0 \bmod p_{\alpha(i,1)}$. Hence, the right-hand side of (s) evaluates to

$$[q_i]^{-0}([q_i][p_{\alpha(i,1)}])^0[p_{\alpha(i,1)}]^{-0} = \mathsf{id}.$$

On the other hand, if $\sigma(\alpha(i,1)) = 1$, then $z_1 \equiv 1 \bmod q_i$ and $z_1 \equiv 1 \bmod p_{\alpha(i,1)}$ and the right-hand side of (s) evaluates again to

$$[q_i]^{-1}[q_i][p_{\alpha(i,1)}][p_{\alpha(i,1)}]^{-1} = \mathsf{id}.$$

In the same way, one can show that also (t) and (u) hold.

For the other direction, assume that $z_1, z_2, z_3 \in \mathbb{Z}$ are such that (q)–(u) hold. We define $A' \subseteq \{1, \dots, m\}$ such that for every $1 \le j \le m$:

- $j \notin S'$ if $z_1 \equiv z_2 \equiv z_3 \equiv 0 \bmod p_j$ and $z_1 \equiv z_2 \equiv 1 \bmod r_j$, and
- $j \in S'$ if $z_1 \equiv z_2 \equiv z_3 \equiv 1 \bmod p_j$ and $z_1 \equiv z_2 \equiv 0 \bmod r_j$.

Consider a set $C_i = \{\alpha(i,1), \alpha(i,2), \alpha(i,3)\}$. From the equations (s), (t), and (u) we get for every $1 \le i \le d$ and $1 \le k \le 3$:

- if $z_1 \equiv 0 \bmod p_{\alpha(i,k)}$ then $z_k \equiv 0 \bmod q_i$
- if $z_1 \equiv 1 \bmod p_{\alpha(i,k)}$ then $z_k \equiv 1 \bmod q_i$

Together with $1 \equiv z_1 + z_2 + z_3 \bmod q_i$ and $q_i \ge 3$, this implies that there must be exactly one $k \in \{1,2,3\}$ such that $z_1 \equiv 1 \bmod p_{\alpha(i,k)}$. Hence, for every $1 \le i \le d$ there is exactly one $k \in \{1,2,3\}$ such that $\alpha(i,k) \in A'$. ◀

Theorem 1.6 is an immediate consequence of Corollaries 4.3 and 5.3 and Theorem 5.6.

Theorem 5.6 leads to the question what the exact complexity of the 2-knapsack problem for symmetric groups is. Recall that the complexity of Luks' 2-membership problem is a famous open problem in the algorithmic theory of permutation groups. The restriction of the 2-membership problem to cyclic groups is easier. The proof of the following theorem uses a reduction to the membership problem for commutative subgroups of matrix groups [8] and then applies [4, Theorem 1.4]; see [33, Theorem 5.8].

▶ **Theorem 5.7.** *The* 2*-knapsack problem for symmetric groups belongs to* P.

6 Application to intersection problems

It remains to show Theorems 1.12 and 1.13. We obtain the upper bound in Theorem 1.12 from Theorem 1.10: Let $\mathcal{G}$ be a grammar from $\mathsf{CFG}(k)$ and let $\mathcal{A}_i = (Q_i, \Sigma, q_{i,0}, \delta_i, F_i)$ be be a group DFA for $1 \le i \le n$. W.l.o.g. assume that the Q_i are pairwise disjoint and let $Q = \bigcup_{1 \le i \le n} Q_i$. To every $a \in \Sigma$ we can associate a permutation $\pi_a \in S_Q$ by setting $\pi_a(q) = \delta_i(q,a)$ if $q \in Q_i$. Let $\mathcal{G}' \in \mathsf{CFG}(k)$ be the context-free grammar over the terminal alphabet S_Q obtained by replacing in $\mathcal{G}$ every occurence of $a \in \Sigma$ by π_a. Then, we have $L(\mathcal{G}) \cap \bigcap_{1 \le i \le n} L(\mathcal{A}_i) \neq \emptyset$ if and only if there exists a permutation $\pi \in L(\mathcal{G}')$ such that $\pi(q_{i,0}) \in F_i$ for every $1 \le i \le n$. We can nondeterministically guess such a permutation and check $\pi \in L(\mathcal{G}')$ in NP using Theorem 1.10. This proves the upper bound from Theorem 1.12.

The upper bound from Theorem 1.13 can be obtained in the same way from Theorem 1.9. For the lower bounds in Theorems 1.12 and 1.13, a simple reduction from the lower bounds in Theorems 1.10 and 1.9 can be employed; see [33, Section 6] for details.

References

1 Emmanuel Arrighi, Henning Fernau, Stefan Hoffmann, Markus Holzer, Ismaël Jecker, Mateus de Oliveira Oliveira, and Petra Wolf. On the complexity of intersection non-emptiness for star-free language classes. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*, volume 213 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.34.

2 Mohamed Faouzi Atig and Pierre Ganty. Approximating petri net reachability along context-free traces. In *Proceedings of the 31st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.152.

3 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

4 László Babai, Robert Beals, Jin-Yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1996*, pages 498–507. ACM/SIAM, 1996.

5 László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, STOC 1987*, pages 409–420. ACM, 1987. doi:10.1145/28395.28439.

6 László Babai and Endre Szemerédi. On the complexity of matrix group problems I. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science, FOCS 1984*, pages 229–240. IEEE Computer Society, 1984. doi:10.1109/SFCS.1984.715919.

7 Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In *Proceedings of the 9th International Conference on Language and Automata Theory and Applications, LATA 2015*, volume 8977 of *Lecture Notes in Computer Science*, pages 473–485. Springer, 2015. doi:10.1007/978-3-319-15579-1_37.

8 Paul Bell, Vesa Halava, Tero Harju, Juhani Karhumäki, and Igor Potapov. Matrix equations and Hilbert's tenth problem. *International Journal of Algebra and Computation*, 18(8):1231–1241, 2008. doi:10.1142/S0218196708004925.

9 Pascal Bergsträßer, Moses Ganardi, and Georg Zetzsche. A characterization of wreath products where knapsack is decidable. In *Proceeding of the 38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021*, volume 187 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.11.

10 Jean-Camille Birget, Stuart Margolis, John Meakin, and Pascal Weil. PSPACE-complete problems for subgroups of free groups and inverse finite automata. *Theoretical Computer Science*, 242(1-2):247–281, 2000. doi:10.1016/S0304-3975(98)00225-4.

11 Michael Blondin, Andreas Krebs, and Pierre McKenzie. The complexity of intersecting finite automata having few final states. *Computational Complexity*, 25(4):775–814, 2016. doi:10.1007/s00037-014-0089-9.

12 Michal Chytil and Burkhard Monien. Caterpillars and context-free languages. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1990*, volume 415 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 1990. doi:10.1007/3-540-52282-4_33.

13 Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the Association for Computing Machinery*, 18(1):4–18, 1971. doi:10.1145/321623.321625.

14 Mateus de Oliveira Oliveira and Michael Wehar. On the fine grained complexity of finite automata non-emptiness of intersection. In *Proceedings of the 24th International Conference Developments in Language Theory, DLT 2020*, volume 12086 of *Lecture Notes in Computer Science*, pages 69–82. Springer, 2020. doi:10.1007/978-3-030-48516-0_6.

15 Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Computing least fixed points of probabilistic systems of polynomials. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010*, volume 5 of *LIPIcs*, pages 359–370. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.STACS.2010.2468.

16 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Information Processing Letters*, 111(12):614–619, 2011. doi:10.1016/j.ipl.2011.03.019.

17 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Proceedings of the 25th International Conference on Computer Aided Verification, CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer, 2013. doi:10.1007/978-3-642-39799-8_8.

18 Javier Esparza, Antonín Kucera, and Stefan Schwoon. Model checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003. doi:10.1016/S0890-5401(03)00139-1.

19 Javier Esparza, Michael Luttenberger, and Maximilian Schlund. A brief history of Strahler numbers. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA 2014*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014. doi:10.1007/978-3-319-04921-2_1.

20 Shimon Even and Oded Goldreich. The minimum-length generator sequence problem is NP-hard. *Journal of Algorithms*, 2(3):311–313, 1981. doi:10.1016/0196-6774(81)90029-8.

21 Michael Figelius, Moses Ganardi, Markus Lohrey, and Georg Zetzsche. The complexity of knapsack problems in wreath products. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 126:1–126:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.126.

22 Elizaveta Frenkel, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 2015. doi:10.1016/j.jsc.2015.05.006.

23 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.

24 Alexander Heußner, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre. Reachability analysis of communicating pushdown systems. In *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010. doi:10.1007/978-3-642-12032-9_19.

25 Robert E. Horton. Erosional development of streams and their drainage basins: hydro-physical approach to quantitative morphology. *Geological Society of America Bulletin*, 56(3):275–370, 1945. doi:10.1130/0016-7606(1945)56[275:EDOSAT]2.0.CO;2.

26 Trevor Jack. On the complexity of properties of partial bijection semigroups, 2021. doi:10.48550/ARXIV.2101.00324.

27 Mark Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985. doi:10.1016/0304-3975(85)90047-7.

28 Mark Kambites, Pedro V. Silva, and Benjamin Steinberg. On the rational subset problem for groups. *Journal of Algebra*, 309(2):622–639, 2007. doi:10.1016/j.jalgebra.2006.05.020.

29 Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 138–153. American Mathematical Society, 2016. doi:10.1090/conm/677.

30 Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science, FOCS 1977*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.

31 Markus Lohrey. *The rational subset membership problem for groups: a survey*, pages 368–389. London Mathematical Society Lecture Note Series. Cambridge University Press, 2015. doi:10.1017/CBO9781316227343.024.

32 Markus Lohrey. Knapsack in hyperbolic groups. *Journal of Algebra*, 545(1):390–415, 2020. doi:10.1016/j.jalgebra.2019.04.008.

33 Markus Lohrey, Andreas Rosowski, and Georg Zetzsche. Membership problems in finite groups, 2022. doi:10.48550/ARXIV.2206.11756.

34 Markus Lohrey and Georg Zetzsche. Knapsack in graph groups. *Theory of Computing Systems*, 62(1):192–246, 2018. doi:10.1007/s00224-017-9808-3.

35 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

36 Eugene M. Luks. Permutation groups and polynomial-time computation. In *Groups And Computation, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 7-10, 1991*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 139–175. DIMACS/AMS, 1991. doi:10.1090/dimacs/011/11.

37 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015. doi:10.1090/S0025-5718-2014-02880-9.

38 Ákos Seress. *Permutation Group Algorithms.* Cambridge Tracts in Mathematics. Cambridge University Press, 2003. doi:10.1017/CBO9780511546549.

39 Charles C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon, 1970. doi:10.1016/B978-0-08-012975-4.50020-5.

40 Arthur N. Strahler. Hypsometric (area-altitude) analysis of erosional topology. *Geological Society of America Bulletin*, 63(11):1117–1142, 1952. doi:10.1130/0016-7606(1952)63[1117:HAAOET]2.0.CO;2.

41 Joseph Swernofsky and Michael Wehar. On the complexity of intersecting regular, context-free, and tree languages. In *Proceedings of the 42nd International Colloquium Automata, Languages, and Programming, Part II, ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2015. doi:10.1007/978-3-662-47666-6_33.

Streaming Word Problems

Markus Lohrey
Universität Siegen, Germany

Lukas Lück
Universität Siegen, Germany

Abstract

We study deterministic and randomized streaming algorithms for word problems of finitely generated groups. For finitely generated linear groups, metabelian groups and free solvable groups we show the existence of randomized streaming algorithms with logarithmic space complexity for their word problems. We also show that the class of finitely generated groups with a logspace randomized streaming algorithm for the word problem is closed under several group theoretical constructions: finite extensions, direct products, free products and wreath products by free abelian groups. We contrast these results with several lower bound. An example of a finitely presented group, where the word problem has only a linear space randomized streaming algorithm, is Thompson's group F.

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

Keywords and phrases word problems for groups, streaming algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.72

Related Version *Full Version*: `https://arxiv.org/abs/2202.04060`

Funding *Markus Lohrey*: Funded by DFG project LO 748/12-1.

1 Introduction

The word problem for a finitely generated group G is the following computational problem: Fix a finite set of generators Σ for G (which means that every element of G can be written as a finite product of elements from Σ. The input for the word problem is a finite word $a_1 a_2 \cdots a_n$ over the alphabet Σ and the question is whether this word evaluates to the group identity of G. The word problem was introduced by Dehn in 1911 [10]. It is arguably the most important computational problem in group theory and has been studied by group theorists as well as computer scientists; see [29] for a good survey. In recent years, complexity theoretic investigations of word problems moved into the focus. For many important classes of groups it turned out that the word problem belongs to low-level complexity classes. The first result in this direction was proved by Lipton and Zalcstein [23] (if the field F has characteristic zero) and Simon [36] (if the field F has prime characteristic): if G is a finitely generated linear group over an arbitrary field F (i.e., a finitely generated group of invertible matrices over F), then the word problem for G can be solved in deterministic logarithmic space. Related results can be found in [20, 41].

The word problem of a group G with a finite generating set Σ can be identified with a formal language $\mathsf{WP}(G, \Sigma)$ consisting of all words over the alphabet Σ that evaluate to the group identity of G. Language theoretic aspects of the word problem have been studied intensively in the past. For instance, Anissimov and Seifert [2] showed that $\mathsf{WP}(G, \Sigma)$ is regular if and only if G is finite, and Muller and Schupp showed that $\mathsf{WP}(G, \Sigma)$ is context-free [31] if and only if G is virtually free,[1] see [18] for an overview.

[1] If C is a property or class of groups, then a group is virtually C if it is a finite extension of a C-group.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 72; pp. 72:1–72:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we initiate the study of streaming algorithms for word problems. These are algorithms that do not have random access on the whole input. Instead, the k-th input symbol is only available at time k [1]. Typically, streaming algorithms are randomized and have a bounded error probability. Usually, one is interested in the space used by a streaming symbol, but also update times have been investigated. Clearly, every regular language has a streaming algorithm with constant space. Randomized streaming algorithms for context-free languages have been studied in [4, 7, 12, 26].

Let us now explain the main results of this paper. For a finitely generated group G with generating set Σ, the *deterministic (resp., randomized) streaming space complexity* of $\mathsf{WP}(G, \Sigma)$ is the space complexity of the best deterministic (resp., randomized) streaming algorithm for $\mathsf{WP}(G, \Sigma)$. The concrete choice of the generating set has only a minor influence on the deterministic (resp., randomized) streaming space complexity of $\mathsf{WP}(G, \Sigma)$; see Lemma 2 for a precise statement. In statements where the influence of the generating set on the streaming space complexity is blurred by the Landau notation, we speak of the deterministic/randomized streaming space complexity of the word problem of G or simply the deterministic/randomized streaming space complexity of G.

The deterministic streaming space complexity of $\mathsf{WP}(G, \Sigma)$ is directly linked to the growth function $\gamma_{G,\Sigma}(n)$ of the group G. The latter is the number of different group elements of G that can be represented by words over the generating set Σ of length at most n (also here the generating set Σ only has a minor influence). The deterministic streaming space complexity of the word problem for G turns out to be $\log_2 \gamma_{G,\Sigma}(n/2)$ up to a small additive constant (Theorem 3). The growth of finitely generated groups is a well investigated topic in geometric group theory. A famous theorem of Gromov says that a finitely generated group has polynomial growth if and only if it is virtually nilpotent; see [9, 28] for a discussion. Theorem 3 reduces all questions about the deterministic streaming space complexity of word problems to questions about growth functions. Due to this, we mainly study randomized streaming algorithms for word problems in this paper.

In the randomized setting, the growth of G still yields a lower bound: The randomized streaming space complexity of the word problem of G is lower bounded by $\Omega(\log\log \gamma_{G,\Sigma}(n/2))$ (Theorem 4). A large class of groups, where this lower bound can be exactly matched by an upper bound are finitely generated linear groups. Recall that Lipton and Zalcstein [23] and Simon [36] showed that the word problem of a finitely generated linear group can be solved in logarithmic space. Their algorithm can be turned into a randomized streaming algorithm with logarithmic space complexity. In order to plug these streaming algorithms into closure results for randomized streaming space complexity (that are discussed below) we need an additional property that we call ϵ-injectivity. Roughly speaking, a randomized streaming algorithm for a finitely generated group G with generating set Σ is *ϵ-injective* if for all words $u, v \in \Sigma^*$ of length at most n we have that: (i) if u and v evaluate to the same element of G then with probability at least $1 - \epsilon$, u and v lead to the same memory state of the streaming algorithm, and (ii) if u and v evaluate to different elements of G then with probability at least $1 - \epsilon$, u and v lead to different memory states of the streaming algorithm; see Section 5. We then show that for every finitely generated linear group G there is a randomized ϵ-injective streaming algorithm with space complexity $\mathcal{O}(\log n)$ (Theorem 8). If G is moreover virtually nilpotent, then the space complexity can be further reduced to $\mathcal{O}(\log\log n)$. In fact, using a known gap theorem for the growth of linear groups [30, 42], it turns out that the randomized streaming space complexity of the word problem for a finitely generated linear group G is either $\Theta(\log\log n)$ (if G is virtually nilpotent) or $\Theta(\log n)$ (if G is not virtually nilpotent), see Theorem 11.

For non-linear groups the situation turns out to be more difficult. We show that the randomized streaming space complexity of word problems is preserved by certain group constructions including finite extensions (Theorem 10), direct products (Lemma 12), free products (Theorem 20) and wreath products by free abelian groups (Theorem 17). For the latter two constructions we also get an additional additive term $\Theta(\log n)$ in the space bounds. For a wreath product $A \wr G$ with A free abelian (resp., a free product $G * H$) it is also important that we start with an ϵ-injective randomized streaming algorithm for G (and H). Using these transfer results we obtain also non-linear groups with a logarithmic randomized streaming space complexity, e.g., metabelian groups (Corollary 13) and free solvable groups (Corollary 18).

In the last section of the paper, we prove lower bounds for the randomized streaming space complexity of word problems. For wreath products of the form $G \wr S$ such that G is non-abelian and S is infinite, we can show that the randomized streaming space complexity is $\Theta(n)$ by a reduction from the randomized communication complexity of disjointness (Theorem 21). A concrete finitely presented group with randomized streaming space complexity $\Theta(n)$ is Thompson's group F (Corollary 22). Thompson's groups F (introduced by Richard Thompson in 1965) belongs due to its unusual properties to the most intensively studied infinite groups; see e.g. [8]. From a computational perspective it is interesting to note that F is co-context-free (i.e., the set of all non-trivial words over any set of generators is a context-free language) [22]. This implies that the word problem for Thompson's group is in LogCFL. To the best of our knowledge no better upper complexity bound is known. Finally, we consider the famous Grigorchuk group G [14], which was the first example of a group with intermediate word growth as well as the first example of a group that is amenable but not elementary amenable. We show that the deterministic streaming space complexity of G is $\mathcal{O}(n^{0.768})$, whereas the randomized streaming space complexity of G is $\Omega(n^{0.5})$ (Theorem 23).

Related results. In this paper, we are only interested in streaming algorithms for a fixed infinite group. Implicitly, streaming algorithms for finite groups are studied in [13]. Obviously, every finite group has a deterministic streaming space complexity $\mathcal{O}(\log |G|)$.[2] In [13], it is shown that for the group $G = \mathsf{SL}(2, \mathbb{F}_p)$ this upper bound is matched by a lower bound, which even holds for the randomized streaming space complexity. More precisely, Gowers and Viola study the communication cost of the following problem: Alice receives a sequence of elements $a_1, \ldots, a_n \in G$, Bob receives a sequence of elements $b_1, \ldots, b_n \in G$ and they are promised that the interleaved product $a_1 b_1 \cdots a_n b_n$ is either 1 or some fixed element $g \in G \setminus \{1\}$ and their job is to determine which of these two cases holds. For $G = \mathsf{SL}(2, \mathbb{F}_p)$ it is shown that the randomized communication complexity of this problem is $\Theta(\log |G| \cdot n)$ (the upper bound is trivial). From this it follows easily that the randomized streaming space complexity of $\mathsf{SL}(2, \mathbb{F}_p)$ is $\Omega(\log |G|)$.

2 Streaming algorithms

For integers $a < b$ let $[a, b] = \{a, a+1, \ldots, b\}$. Fix a finite alphabet Σ. For a word $w \in \Sigma^*$ let $|w|$ be its length and let $\Sigma^{\leq n} = \{w \in \Sigma^* \colon |w| \leq n\}$ be the set of words of length at most n.

In the following we introduce probabilistic finite automata [33, 34] as a model for randomized streaming algorithms. A *probabilistic finite automaton* (PFA) $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ consists of a finite set of states Q, an alphabet Σ, an *initial state distribution* $\iota \colon Q \to \{r \in$

[2] In our setting, $|G|$ would be a constant, but for the moment let us make the dependence on the finite group G explicit.

$\mathbb{R}\colon 0 \leq r \leq 1\}$, a *transition probability function* $\rho\colon Q \times \Sigma \times Q \to \{r \in \mathbb{R}\colon 0 \leq r \leq 1\}$ and a set of final states $F \subseteq Q$ such that $\sum_{q \in Q} \iota(q) = 1$ and $\sum_{q \in Q} \rho(p, a, q) = 1$ for all $p \in Q$, $a \in \Sigma$. If ι and ρ map into $\{0, 1\}$, then $\mathcal{A}$ is a *deterministic finite automaton* (DFA). If only ρ is required to map into $\{0, 1\}$, then $\mathcal{A}$ is called *semi-probabilitistic.* A *run* on a word $a_1 \cdots a_m \in \Sigma^*$ in $\mathcal{A}$ is a sequence $\pi = (q_0, a_1, q_1, a_2, \ldots, a_m, q_m)$ where $q_0, \ldots, q_m \in Q$. Given such a run π in $\mathcal{A}$ we define $\rho_\iota(\pi) = \iota(q_0) \cdot \prod_{i=1}^{n} \rho(q_{i-1}, a_i, q_i)$. For each $w \in \Sigma^*$ the function ρ_ι is a probability distribution on the set $\mathrm{Runs}(w)$ of all runs of $\mathcal{A}$ on w. A run $\pi = (q_0, a_1, \ldots, a_m, q_m)$ is *correct* with respect to a language $L \subseteq \Sigma^*$ if $q_m \in F \Leftrightarrow a_1 \cdots a_m \in L$ holds. The *error probability* of $\mathcal{A}$ on w for L is

$$\epsilon(\mathcal{A}, w, L) = \sum \{\rho_\iota(\pi) : \pi \in \mathrm{Runs}(w) \text{ is not correct w.r.t. } L\}.$$

If $\mathcal{A}$ is semi-probabilitistic then we can identify ρ with a mapping $\rho : Q \times \Sigma \to Q$, where $\rho(p, a)$ is the unique state q with $\rho(p, a, q) = 1$. This mapping ρ is extended to a mapping $\rho : Q \times \Sigma^* \to Q$ in the usual way: $\rho(p, \varepsilon) = p$ and $\rho(p, aw) = \rho(\rho(p, a), w)$. We then obtain

$$\epsilon(\mathcal{A}, w, L) = 1 - \sum \{\iota(q) : q \in Q, \delta(q, w) \in F \Leftrightarrow w \in L\}.$$

A (non-uniform) *randomized streaming algorithm* is a sequence $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ of PFA $\mathcal{A}_n$ over the same alphabet Σ. If every $\mathcal{A}_n$ is deterministic (resp., semi-probabilitistic), we speak of a *deterministic (resp., semi-randomized)* streaming algorithm.

Let $0 \leq \epsilon \leq 1$ be an error probability. A randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ is ϵ-*correct* for a language $L \subseteq \Sigma^*$ if for every $n \geq 0$ and every word $w \in \Sigma^{\leq n}$ we have $\epsilon(\mathcal{A}_n, w, L) \leq \epsilon$. We say that $\mathcal{R}$ is a *randomized streaming algorithm for* L if it is 1/3-correct for L. The choice of 1/3 for the error probability is not important. Using a standard application of the Chernoff bound one can reduce the error probability from 1/3 to every constant. If $\mathcal{R}$ is deterministic and 0-correct for L then we say that $\mathcal{R}$ is a *deterministic streaming algorithm for* L. The *space complexity* of the randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ is the function $s(\mathcal{R}, n) = \lceil \log_2 |Q_n| \rceil$, where Q_n is the state set of $\mathcal{A}_n$. The motivation for this definition is that states of Q_n can be encoded by bit strings of length at most $\lceil \log_2 |Q_n| \rceil$. The *deterministic/randomized streaming space complexity of the language* L is the smallest possible function $s(\mathcal{R}, n)$, where $\mathcal{R}$ is a deterministic/randomized streaming algorithm for L. By a result of Rabin [34, Theorem 3], the deterministic streaming space complexity of a language L is bounded by $2^{\mathcal{O}(S(n))}$, where $S(n)$ is the randomized streaming space complexity of L.

The deterministic streaming space complexity of a language L is directly linked to the *automaticity of* L. The automaticity of $L \subseteq \Sigma^*$ is the the function $A_L(n)$ that maps n to the number of states of a smallest DFA $\mathcal{A}_n$ such that for all words $w \in \Sigma^{\leq n}$ we have: $w \in L$ if and only if w is accepted by $\mathcal{A}_n$. Hence, the deterministic streaming space complexity of L is exactly $\lceil \log_2 A_L(n) \rceil$. The automaticity of languages was studied in [35]. Interesting in our context is the following result of Karp [19]: if L is a non-regular language then $A_L(n) \geq (n+3)/2$ for infinitely many n. Hence, for every non-regular language the deterministic streaming space complexity of L is at least $\log_2(n) - c$ for a constant c and infinitely many n. Another related measure is the online space complexity from [11], which is defined by the growth function of an infinite automaton for a language L.

Note that our concept of streaming algorithms is non-uniform in the sense that for every input length n we have a separate streaming algorithm $\mathcal{A}_n$. This makes lower bounds stronger. On the other hand, the streaming algorithms that we construct for concrete groups will be uniform in the sense that the streaming algorithms $\mathcal{A}_n$ follow a common pattern. The following result uses non-uniformity in a crucial way; its proof (see [25, Theorem 3.1]) is similar to Newman's theorem on public versus private coins in communication complexity, see e.g. [21].

▶ **Theorem 1.** *Let $\mathcal{R}$ be a randomized streaming algorithm such that $s(\mathcal{R}, n) \geq \Omega(\log n)$ and $\mathcal{R}$ is ϵ-correct for a language L. Then there exists a semi-randomized streaming algorithm $\mathcal{S}$ such that $s(\mathcal{S}, n) = \Theta(s(\mathcal{R}, n))$ and $\mathcal{S}$ is 2ϵ-correct for the language L.*

3 Groups and word problems

For a group G and a subset $\Sigma \subseteq G$, we denote with $\langle \Sigma \rangle$ the subgroup of G generated by Σ. It is the set of all products of elements from $\Sigma \cup \Sigma^{-1}$. We only consider *finitely generated (f.g.) groups* G, for which there is a finite set $\Sigma \subseteq G$ such that $G = \langle \Sigma \rangle$; such a set Σ is called a *finite generating set* for G. If $\Sigma = \Sigma^{-1}$ then we say that Σ is a *finite symmetric generating set* for G. In the following we assume that all finite generating sets are symmetric. Every word $w \in \Sigma^*$ evaluates to a group element $\pi_G(w)$ in the natural way; here $\pi_G : \Sigma^* \to G$ is the canonical morphism from the free monoid Σ^* to G. Instead of $\pi_G(u) = \pi_G(v)$ we also write $u \equiv_G v$.

Let $C(G, \Sigma)$ be the Cayley graph of G with respect to the finite symmetric generating set Σ. It is the edge-labelled graph whose vertex set is G and that has an a-labelled edge from $\pi_G(u)$ to $\pi_G(ua)$ for all $u \in \Sigma^*$ and $a \in \Sigma$. Let $\mathsf{WP}(G, \Sigma) = \{w \in \Sigma^* \mid \pi_G(w) = 1\}$ be the *word problem for* G with respect to the generating set Σ.

We are interested in streaming algorithms for words problems $\mathsf{WP}(G, \Sigma)$. The following lemma is easy to prove; see [25, Lemma 4.1].

▶ **Lemma 2.** *Let Σ_1 and Σ_2 be finite symmetric generating sets for the group G and let $s_i(n)$ be the deterministic/randomized streaming space complexity of* $\mathsf{WP}(G, \Sigma_i)$*. Then there exists a constant c that depends on G, Σ_1 and Σ_2 such that $s_1(n) \leq s_2(c \cdot n)$.*

By Lemma 2, the dependence of the streaming space complexity from the generating set is often blurred by the use of the $\mathcal{O}$-notation. In such situations we will speak of the deterministic/randomized streaming space complexity for the group G (instead of the deterministic/randomized streaming space complexity of the language $\mathsf{WP}(G, \Sigma)$).

4 Streaming algorithms for word problems and growth

Let G be a finitely generated group and let Σ be a finite symmetric generating set for G. For $n \in \mathbb{N}$ let $B_{G,\Sigma}(n) = \pi_G(\Sigma^{\leq n}) \subseteq G$ be the ball of radius n in the Cayley-graph of G with center 1. The growth function $\gamma_{G,\Sigma} : \mathbb{N} \to \mathbb{N}$ is the function with $\gamma_{G,\Sigma}(n) = |B_{G,\Sigma}(n)|$. The deterministic streaming space complexity of G is completely determined by the growth of G:

▶ **Theorem 3.** *Let G be a finitely generated infinite group and let Σ be a finite symmetric generating set for G. Define the function $S(n)$ by $S(n) = \gamma_{G,\Sigma}(\lfloor n/2 \rfloor) + (n \bmod 2)$. Then, the deterministic streaming space complexity of* $\mathsf{WP}(G, \Sigma)$ *is $\lceil \log_2 S(n) \rceil$.*

Proof. We start with the upper bound in case n is even. In the following we identify the ball $B_{G,\Sigma}(n/2)$ with its induced subgraph of the Cayley graph $C(G, \Sigma)$. We define a deterministic finite automaton $\mathcal{A}_n$ by taking the edge-labelled graph $B_{G,\Sigma}(n/2)$ with the initial and unique final state 1. It can be viewed as a partial DFA in the sense that for every $g \in B_{G,\Sigma}(n/2)$ and every $a \in \Sigma$, g has at most one outgoing edge labelled with a (that leads to $g \cdot a$ if $g \cdot a \in B_{G,\Sigma}(n/2)$). In order to add the missing transitions we choose an element $g_f \in B_{G,\Sigma}(n/2) \setminus B_{G,\Sigma}(n/2 - 1)$ (here, we set $B_{G,\Sigma}(-1) = \emptyset$). Such an element exists because G is infinite. If $g \in B_{G,\Sigma}(n/2)$ has not outgoing a-labelled edge in $B_{G,\Sigma}(n/2)$ then we add an a-labelled edge from g to g_f. We call those edges *spurious*. The resulting DFA is $\mathcal{A}_n$.

We claim that for every word $w \in \Sigma^{\leq n}$, w is accepted by $\mathcal{A}_n$ if and only if $w \in \mathsf{WP}(G, \Sigma)$. This is clear, if no spurious edge is traversed while reading w into $\mathcal{A}_n$. In this case, after reading w, we end up in state $\pi_G(w)$. Now assume that a spurious edge is traversed while reading w into $\mathcal{A}_n$ and let x be the shortest prefix of w such that a spurious edge is traversed while reading the last symbol of x. Let us write $w = xy$. We must have $|x| > n/2$ and $\pi_G(x) \notin B_{G,\Sigma}(n/2)$. Moreover, $|y| < n - n/2 = n/2$. Since $\pi_G(x) \notin B_{G,\Sigma}(n/2)$, we have $w = xy \notin \mathsf{WP}(G, \Sigma)$. Moreover, w is rejected by $\mathcal{A}_n$, because x leads in $\mathcal{A}_n$ from the initial state 1 to state g_f and there is no path of length at most $n/2 - 1$ from g_f back to the final state 1.

For the case that n is odd, we take the ball $B_{G,\Sigma}(\lfloor n/2 \rfloor)$. Instead of adding spurious edges we add a failure state f. If $g \in B_{G,\Sigma}(\lfloor n/2 \rfloor)$ has no outgoing a-labelled edge in $B_{G,\Sigma}(\lfloor n/2 \rfloor)$, then we add an a-labelled edge from g to f. Moreover, for every $a \in \Sigma$ we add an a-labelled loop at state f. As for the case n even, one can show that the resulting DFA accepts a word $w \in \Sigma^{\leq n}$ if and only if $w \in \mathsf{WP}(G, \Sigma)$.

For the lower bound, let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a smallest DFA such that for every word $w \in \Sigma^{\leq n}$, w is accepted by $\mathcal{A}$ if and only if $w \in \mathsf{WP}(G, \Sigma)$. We have to show that $|Q| \geq S(n)$. Let us consider two words $u, v \in \Sigma^*$ of length at most $\lfloor n/2 \rfloor$ such that $u \not\equiv_G v$ and $\delta(q_0, u) = \delta(q_0, v)$. We then have $uv^{-1} \notin \mathsf{WP}(G, \Sigma)$ and $vv^{-1} \in \mathsf{WP}(G, \Sigma)$. On the other hand, we have $\delta(q_0, uv^{-1}) = \delta(q_0, vv^{-1})$, which is a contradiction (note that $|uv^{-1}|, |vv^{-1}| \leq n$). Hence, if $\delta(q_0, u) = \delta(q_0, v)$ for two words $u, v \in \Sigma^*$ of length at most $\lfloor n/2 \rfloor$, then $u \equiv_G v$.

Let $Q' = \{\delta(q_0, w) \mid w \in \Sigma^*, |w| \leq \lfloor n/2 \rfloor\} \subseteq Q$. The previous paragraph shows that $|Q'| \geq \gamma_{G,\Sigma}(\lfloor n/2 \rfloor)$. If n is even then $\lfloor n/2 \rfloor = n/2$ and we are done. So, let us assume that n is odd.

If $|Q| > \gamma_{G,\Sigma}(\lfloor n/2 \rfloor)$ then we are again done. So, let us assume that $Q = Q'$ and $|Q| = \gamma_{G,\Sigma}(\lfloor n/2 \rfloor)$. Then, to every state $q \in Q$ we can assign a unique group element $g_q \in B_{G,\Sigma}(\lfloor n/2 \rfloor)$ such that for every word $w \in \Sigma^*$ with $|w| \leq \lfloor n/2 \rfloor$ we have $\delta(q_0, w) = q$ if and only if $\pi_G(w) = g_q$. The mapping $q \mapsto g_q$ is a bijection between Q and $B_{G,\Sigma}(\lfloor n/2 \rfloor)$.

Let us now take a state $q \in Q$ and a generator $a \in \Sigma$ such that $g_q \cdot a \notin B_{G,\Sigma}(\lfloor n/2 \rfloor)$. Such a state and generator must exist since G is infinite. Let $u, v \in \Sigma^*$ be words of length at most $\lfloor n/2 \rfloor$ such that $\delta(q_0, u) = q$ and $\delta(q_0, v) = \delta(q, a) = \delta(q_0, ua)$. We obtain $\delta(q_0, vv^{-1}) = \delta(q_0, uav^{-1})$. But $vv^{-1} \in \mathsf{WP}(G, \Sigma)$ and $uav^{-1} \notin \mathsf{WP}(G, \Sigma)$ since $\pi_G(uav^{-1}) = g_q \cdot a \cdot \pi_G(v^{-1})$ and $g_q \cdot a \notin B_{G,\Sigma}(\lfloor n/2 \rfloor)$, $\pi_G(v^{-1}) \in B_{G,\Sigma}(\lfloor n/2 \rfloor)$. This is a contradiction since vv^{-1} and uav^{-1} both have length at most n. ◀

The growth of f.g. groups is well-studied and Theorem 3 basically closes the chapter on deterministic streaming algorithms for word problems. Hence, in the rest of the paper we focus on randomized streaming algorithms. Here, we can still prove a lower bound (that will turn out to be sharp in some cases but not always) using the randomized one-way communication complexity of the equality problem; see [25, Theorem 5.2] for details.

▶ **Theorem 4.** *Let G be a finitely generated group and let Σ be a finite symmetric generating set for G. The randomized streaming space complexity of* $\mathsf{WP}(G, \Sigma)$ *is* $\Omega(\log\log \gamma_{G,\Sigma}(\lfloor n/2 \rfloor))$.

▶ Remark 5. Since every f.g. infinite group has growth at least n, Theorem 4 has the following consequence: If G is a f.g. infinite group, then the randomized streaming space complexity of G is $\Omega(\log\log n)$.

▶ Remark 6. Later in this paper, we will make use of the following two famous results on the growth of groups, see also [9, 28]:

- Gromov's theorem [16]: A f.g. group has polynomial growth iff it is virtually nilpotent.

- Wolf-Milnor theorem [30, 42]; see also [9, p. 202]: A f.g. solvable group G is either virtually nilpotent (and hence has polynomial growth) or there is a constant $c > 1$ such that G has growth c^n (i.e., G has exponential growth). It is well known that the same dichotomy also holds for f.g. linear groups. This is a consequence of Tits alternative [37]: A f.g. linear group G is either virtually solvable or contains a free group of rank at least two (in which case G has exponential growth).

The dichotomy theorem of Milnor and Wolf does not generalize to all f.g. groups. Grigorchuk [14] constructed a f.g.group whose growth is lower bounded by $\exp(n^{0.515})$ [6] and upper bounded by $\exp(n^{0.768})$ [5]. The streaming space complexity of this remarkable group will be studied in Theorem 23.

5 Injective randomized streaming algorithms

For a semi-probabilistic finite automaton $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ and some boolean condition $\mathcal{E}(q)$ that depends on the state $q \in Q$, we define the probability

$$\mathsf{Prob}_{q \in Q}[\mathcal{E}(q)] = \sum_{q \in Q, \mathcal{E}(q)=1} \iota(q).$$

Let G be a f.g. group G with the finite generating set Σ. A randomized streaming algorithm $(\mathcal{A}_n)_{n \geq 0}$ with $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n, F_n)$ is called ϵ*-injective* for G (with respect to Σ) if the following properties hold for all $n \geq 0$ and all words $u, v \in \Sigma^{\leq n}$:

- $\mathcal{A}_n$ is semi-randomized.
- If $u \equiv_G v$ then $\mathsf{Prob}_{q \in Q_n}[\rho_n(q, u) = \rho_n(q, v)] \geq 1 - \epsilon$.
- If $u \not\equiv_G v$ then $\mathsf{Prob}_{q \in Q_n}[\rho_n(q, u) \neq \rho_n(q, v)] \geq 1 - \epsilon$.

Note that the set F_n of final states of $\mathcal{A}_n$ is not important and we will just write $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n)$ in the following if we talk about an ϵ-injective randomized streaming algorithm $(\mathcal{A}_n)_{n \geq 0}$. The easy proof of the following lemma can be found in [25, Lemma 6.1].

▶ **Lemma 7.** *If $\mathcal{R}$ is an ϵ-injective randomized streaming algorithm for G w.r.t. Σ, then* $\mathsf{WP}(G, \Sigma)$ *has an ϵ-correct semi-randomized streaming algorithm with space complexity $2 \cdot s(\mathcal{R}, n)$.*

Due to Lemma 7, our goal in the rest of the paper will the construction of space efficient ϵ-injective randomized streaming algorithms for groups. We will need ϵ-injectivity for wreath products and free products; see Sections 7.2 and 7.3.

6 Randomized streaming algorithms for linear groups

For every f.g. linear group, the word problem can be solved in logarithmic space. This was shown by Lipton and Zalcstein [23] (if the underlying field has characteristic zero) and Simon [36] (if the underlying field has prime characteristic). The idea is to carry out all computations modulo sufficiently many small prime numbers. This idea can be easily turned into a randomized streaming algorithm by randomly choosing a small prime number. With some care, one can turn their idea into an $\epsilon(n)$-injective randomized streaming algorithm with $\epsilon(n) = 1/n^c$ for a constant c and space complexity $\mathcal{O}(\log n)$; see [25, Theorem 7.1] for details.

▶ **Theorem 8.** *For every f.g. linear group G and every $c > 0$ there exists an $\epsilon(n)$-injective randomized streaming algorithm with $\epsilon(n) = 1/n^c$ and space complexity $\mathcal{O}(\log n)$.*

Every nilpotent group is linear. For nilpotent groups we can improve the algorithm from the proof of Theorem 8, at least if we sacrifice the inverse polynomial error probability. The proof of the following theorem (see [25, Theorem 7.2]) uses the fact that every f.g. nilpotent group is a finite extension of a nilpotent group that can be embedded in the group $\mathsf{UT}_d(\mathbb{Z})$ of d-dimensional unitriangular matrices over $\mathbb{Z}$. The entries in a product of n such matrices $A_1, \ldots, A_n$ can be bounded by $\mathcal{O}(n^{d-1})$, provided all entries in the A_i are of size $\mathcal{O}(1)$ (in absolute value). This allows to compute modulo a random prime number with $\mathcal{O}(\log\log n)$ bits.

▶ **Theorem 9.** *For every f.g. nilpotent group G and every constant $c > 0$ there exists an $\epsilon(n)$-injective randomized streaming algorithm with $\epsilon(n) = 1/\log^c n$ and space complexity $\mathcal{O}(\log\log n)$.*

Note that if G is infinite, the upper bound from Theorem 9 is sharp up to constant factors even if we allow a constant error probability; see Remark 5.

7 Closure properties for streaming space complexity

In this section, we will show that many group theoretical constructions preserve randomized streaming space complexity.

7.1 Easy cases: finite extensions and direct products

For many algorithmic problems in group theory, the complexity is preserved by finite extensions. This is also true for the streaming space complexity of the word problem; see also [25, Theorem 8.1]:

▶ **Theorem 10.** *Assume that H is a f.g. group and G is a subgroup of H of finite index (hence, also G must be finitely generated). Assume that $\mathcal{R}$ is an ϵ-injective randomized streaming algorithm for G. Then H has an ϵ-injective randomized streaming algorithm with space complexity $s(\mathcal{R}, c \cdot n) + \mathcal{O}(1)$ for some constant c.*

Recall that Gromov proved that a finitely generated group has polynomial growth if and only if it is virtually nilpotent.

▶ **Corollary 11.** *Let G be an infinite finitely generated linear group.*

- *If G is virtually nilpotent then the randomized streaming space complexity of G is $\Theta(\log\log n)$.*
- *If G is not virtually nilpotent then the randomized streaming space complexity of G is $\Theta(\log n)$.*

Proof. The upper bounds follow from Theorems 8 and 9. Since G is infinite, the randomized streaming space complexity of the word problem of G is $\Omega(\log\log n)$ (see Remark 5), which yields the lower bound for the virtually nilpotent case. If G is not virtually nilpotent, then G has growth c^n for some constant $c > 1$ (see Remark 6), which yields the lower bound $\Theta(\log n)$ by Theorem 4. ◀

It is conjectured that for every f.g. group G that is not virtually nilpotent the growth is lower bounded by $\exp(n^{0.5})$. This is known as the *gap conjecture* [15]. It would imply that for every f.g. group that is not virtually nilpotent the randomized streaming space complexity is lower bounded by $\Omega(\log n)$.

Also direct products preserve the streaming space complexity of the word problem (simply run the streaming algorithms for the two factor groups in parallel):

▶ **Lemma 12.** *Let G and H be f.g. groups for which there exist $\epsilon(n)$-injective randomized streaming algorithms $\mathcal{R}$ and $\mathcal{S}$, respectively. Then there exists a $2\epsilon(n)$-injective randomized streaming algorithm for $G \times H$ with space complexity $s(\mathcal{R}, n) + s(\mathcal{S}, n)$.*

Recall that a group G is metabelian if it has an abelian normal subgroup $A \leq G$ such that the quotient G/A is abelian as well. Every f.g. metabelian group can be embedded into a direct product of linear groups (over fields of different characteristics) [40]. Hence, with Lemma 12 and Theorem 8 we obtain:

▶ **Corollary 13.** *For every f.g. metabelian group and every $c > 0$ there exists an $\epsilon(n)$-injective randomized streaming algorithm with $\epsilon(n) = 1/n^c$ and space complexity $\mathcal{O}(\log n)$.*

7.2 Randomized streaming algorithms for wreath products

Let G and H be groups. Consider the direct sum $K = \bigoplus_{g \in G} H_g$, where H_g is a copy of H. We view K as the set $H^{(G)}$ of all mappings $f \colon G \to H$ such that $\mathsf{supp}(f) := \{g \in G \mid f(g) \neq 1\}$ is finite, together with pointwise multiplication in H as the group operation. The set $\mathsf{supp}(f) \subseteq G$ is called the *support* of f. The group G has a natural left action on $H^{(G)}$ given by $gf(a) = f(g^{-1}a)$, where $f \in H^{(G)}$ and $g, a \in G$. The corresponding semidirect product $H^{(G)} \rtimes G$ is the (restricted) *wreath product* $H \wr G$. In other words:

- Elements of $H \wr G$ are pairs (f, g), where $g \in G$ and $f \in H^{(G)}$.
- The multiplication in $H \wr G$ is defined as follows: Let $(f_1, g_1), (f_2, g_2) \in H \wr G$. Then $(f_1, g_1)(f_2, g_2) = (f, g_1 g_2)$, where $f(a) = f_1(a) f_2(g_1^{-1} a)$ for all $a \in G$.

Intuitively, the mapping $a \mapsto f_2(g_1^{-1}a)$ is the mapping f_2 shifted by g_1.

Clearly, G is a subgroup of $H \wr G$. We also regard H as a subgroup of $H \wr G$ by identifying H with the set of all $f \in H^{(G)}$ with $\mathsf{supp}(f) \subseteq \{1\}$. This copy of H together with G generates $H \wr G$. In particular, if $G = \langle \Sigma \rangle$ and $H = \langle \Gamma \rangle$ with $\Sigma \cap \Gamma = \emptyset$ then $H \wr G$ is generated by $\Sigma \cup \Gamma$. In [32] it was shown that the word problem of a wreath product $H \wr G$ is TC^0-reducible to the word problems for G and H.

In this section, we study streaming algorithms for wreath products. The case of a wreath product $H \wr G$ with G finite is easy:

▶ **Proposition 14.** *Let H be a f.g. group for which there exists an ϵ-injective randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ and let G be a finite group of size c. Then, there exists an $(c \cdot \epsilon)$-injective randomized streaming algorithm for $H \wr G$ with space complexity $\mathcal{O}(s(\mathcal{R}, n))$.*

Proof. We run c independent copies of $\mathcal{A}_n$ (for the direct product of c copies of H). In addition we have to store an element of G. ◀

The case of a wreath product $H \wr G$ with G infinite turns out to be more interesting. In Section 8 we will prove a lower bound for the case that H is non-abelian. In this section, we consider the case where H is abelian. Our construction will start with ϵ-injective randomized streaming algorithms for G and uses the following simple fact.

▶ **Lemma 15.** *Let $(\mathcal{A}_n)_{n \geq 0}$ be an ϵ-injective randomized streaming algorithm for the finitely generated group G with respect to the generating set Σ. Let $\mathcal{A}_n = (Q_n, \Sigma, \iota, \rho)$. Consider a set $S \subseteq \Sigma^{\leq n}$. For every state q of $\mathcal{A}_n$ consider the equivalence relation $\equiv_q$ on S with $u \equiv_q v$ if and only if $\rho_n(q, u) = \rho_n(q, v)$. Then*

$$\operatorname*{\mathsf{Prob}}_{q \in Q_n}[\equiv_q \textit{ coincides with } \equiv_G \textit{ on } S] \geq 1 - \epsilon \cdot \binom{|S|}{2}.$$

Proof. For all $u, v \in S$, the probability that either $u \equiv_G v$ and $u \not\equiv_q v$ or $u \not\equiv_G v$ and $u \equiv_q v$ is bounded by ϵ. The lemma follows from the union bound since there are $\binom{|S|}{2}$ many unordered pairs. ◀

We start with the case of a wreath product $\mathbb{Z} \wr G$. Note that the following theorem makes only sense if $\epsilon < 1/2n^2$. On the other hand, such an inverse polynomial error probability can be achieved for linear groups by Theorem 8.

▶ **Theorem 16.** *Let G be a f.g. infinite group and $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ an ϵ-injective randomized streaming algorithm for G. Let d be a fixed constant and $\zeta = 2\epsilon n^2 + \max\{\epsilon, 1/n^d\}$. Then there exists an ζ-injective randomized streaming algorithm $\mathcal{S} = (\mathcal{B}_n)_{n \geq 0}$ for $\mathbb{Z} \wr G$ with space complexity $2 \cdot s(\mathcal{R}, n) + \Theta(\log n)$.*

Proof sketch. A complete proof of the theorem can be found in [25, Theorem 8.9]. Fix a symmetric generating set Σ for G and let a be a generator of $\mathbb{Z}$. Let $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n)$. W.l.o.g. we can assume that $Q_n = [0, |Q_n| - 1]$. Fix an input length n. For an input word $w \in (\Sigma \cup \{a, a^{-1}\})^{\leq n}$ our randomized streaming algorithm for $\mathbb{Z} \wr G$ runs $\mathcal{A}_n$ on the projection $\pi_\Sigma(w)$ of the word w to the subalphabet Σ. Initially, the algorithm guesses a state $q \in Q_n$ according to the initial state distribution ι_n and (independently from q) a prime number $p \in [2, \alpha]$. The number α will be fixed latter. For the moment, let us only mention that p will have at most $s(\mathcal{R}, n) + \Theta(\log n)$ bits. Apart from the current state of $\mathcal{A}_n$ the algorithm also stores a number $z \in [0, p-1]$ which initially is set to zero.

Assume that at some time instant the algorithm reads the letter a^γ, where $\gamma \in \{-1, 1\}$. Let q be the current $\mathcal{A}_n$-state, which is a number in the range $[0, |Q_n| - 1]$. The above prime number p has to be chosen uniformly from the set of all primes of size $\Theta(|Q_n| \cdot n^{d+1})$. Then the algorithm updates $z \in [0, p-1]$ as follows:

$$z := (z + \gamma \cdot (n+1)^q) \bmod p.$$

With our input word w and a state q of $\mathcal{A}_n$ we associate a polynomial $P_{q,w}(x) \in \mathbb{Z}[x]$ as follows: Let R_w be the set of all prefixes of w that end with a letter a^γ and for $s \in R_w$ define $\sigma(s) = \gamma \in \{-1, 1\}$ if s ends with a^γ. Moreover, for every $s \in R_w$ consider the $\mathcal{A}_n$-state $q_s = \rho_n(q, \pi_\Sigma(s)) \in [0, |Q_n| - 1]$. We then define the polynomial

$$P_{q,w}(x) := \sum_{s \in R_w} \sigma(s) \cdot x^{q_s}.$$

Note that this polynomial has degree at most $|Q_n| - 1$ and all its coefficients have absolute value at most n. Moreover, the number $z = z(p, q, w)$ computed by the algorithm on input w for the random choice $p \in [2, \alpha], q \in [0, |Q_n| - 1]$ is

$$z(p, q, w) = P_{q,w}(n+1) \bmod p. \tag{1}$$

This concludes the description of the streaming algorithm. It remains to show for all words $u, v \in (\Sigma \cup \{a, a^{-1}\})^{\leq n}$ the following:

(a) If $u \equiv_{\mathbb{Z} \wr G} v$ then $\underset{p \in [2,\alpha], q \in Q_n}{\mathsf{Prob}}[\rho_n(q, \pi_\Sigma(u)) = \rho_n(q, \pi_\Sigma(v)) \wedge z(p, q, u) = z(p, q, v)] \geq 1 - \zeta$.

(b) If $u \not\equiv_{\mathbb{Z} \wr G} v$ then $\underset{p \in [2,\alpha], q \in Q_n}{\mathsf{Prob}}[\rho_n(q, \pi_\Sigma(u)) \neq \rho_n(q, \pi_\Sigma(v)) \vee z(p, q, u) \neq z(p, q, v)] \geq 1 - \zeta$.

Let $(f_u, g_u) \in \mathbb{Z} \wr G$ (resp., $(f_v, g_v) \in \mathbb{Z} \wr G$) be the group element represented by the word u (resp., v). First assume that $u \equiv_{\mathbb{Z} \wr G} v$, i.e., $f_u = f_v$ and $g_u = g_v$. From $g_u = g_v$ we get

$$\underset{q \in Q_n}{\mathsf{Prob}}[\rho_n(q, \pi_\Sigma(u)) = \rho_n(q, \pi_\Sigma(v))] \geq 1 - \epsilon. \tag{2}$$

Moreover, from $f_u = f_v$ and Lemma 15 one can deduce

$$\underset{p\in[2,\alpha],q\in Q_n}{\mathsf{Prob}}[z(p,q,u) = z(p,q,v)] \geq \underset{q\in Q_n}{\mathsf{Prob}}[P_{q,u}(x) = P_{q,v}(x)] \geq 1 - 2\epsilon n^2. \tag{3}$$

Finally, (2) and (3) easily yield the conclusion of point (a). Now assume that $u \not\equiv_{\mathbb{Z}\wr G} v$. If $g_u \neq g_v$, i.e., then we get

$$\underset{q\in Q_n}{\mathsf{Prob}}[\rho_n(q,\pi_\Sigma(u)) \neq \rho_n(q,\pi_\Sigma(v))] \geq 1 - \epsilon \geq 1 - \zeta.$$

On the other hand, if the mappings f_u and f_v differ, then from Lemma 15 we obtain

$$\underset{q\in Q_n}{\mathsf{Prob}}[P_{q,u}(n+1) \neq P_{q,v}(n+1)] \geq \underset{q\in Q_n}{\mathsf{Prob}}[P_{q,u}(x) \neq P_{q,v}(x)] \geq 1 - 2\epsilon n^2.$$

The first inequality follows from Cauchy's bound. This, together with (1) and some standard bounds on the number of different prime factors of $|P_{q,u}(n+1) - P_{q,v}(n+1)|$ yields $\mathsf{Prob}_{p\in[2,\alpha],q\in Q_n}[z(p,q,u) \neq z(p,q,v)] \geq 1 - \zeta$ and finally the conclusion of point (b). ◀

It is easy to extend Theorem 16 to a wreath product $\mathbb{Z}_p \wr G$ with p prime. For a wreath product $\mathbb{Z}_{p^k} \wr G$ with p a prime and $k \geq 2$ we can only prove the following weaker statement using a polynomial identity testing algorithm for local rings from [3]. Putting it all together we can show the following result; see [25, Section 8.3] for details.

▶ **Theorem 17.** *Let G be a f.g. group for which there exists an ϵ-injective randomized streaming algorithm $\mathcal{R}$. Let A be a finitely generated abelian group. Then for all constants $\epsilon \leq \epsilon' < 1$ and $d \geq 1$ there exists a ζ-injective randomized streaming algorithm $\mathcal{S}$ for $A \wr G$ with the following properties:*

- *$\zeta \leq \mathcal{O}(\epsilon' + \epsilon n^2)$ and $s(\mathcal{S}, n) \leq \mathcal{O}(s(\mathcal{R}, n)^2 + \log n)$*
- *$\zeta \leq \mathcal{O}(1/n^d + \epsilon n^2)$ and $s(\mathcal{S}, n) \leq \mathcal{O}(s(\mathcal{R}, n) + \log n)$ if A is a direct product of copies of $\mathbb{Z}$ and $\mathbb{Z}_p$ with p prime.*

▶ **Corollary 18.** *Every free solvable group has randomized streaming space complexity $\Theta(\log n)$.*

Proof. Magnus' embedding theorem [27] says that every free solvable group can be embedded into an iterated wreath product $\mathbb{Z}^m \wr (\mathbb{Z}^m \wr (\mathbb{Z}^m \wr \cdots))$. Since $\mathbb{Z}^m$ is linear, we can, using Theorem 8, obtain an $\epsilon(n)$-injective randomized streaming algorithm for $\mathbb{Z}^m$ with space complexity $\mathcal{O}(\log n)$ for every inverse polynomial $\epsilon(n)$. We then apply the second statement of Theorem 17 a constant number of times and obtain a randomized streaming algorithm with space complexity $\mathcal{O}(\log n)$. The lower bound follows from Theorem 4 and the Milnor-Wolf theorem (see Remark 6). ◀

In [38] it is shown that the word problem of a free solvable group can be solved with a randomized algorithm running in time $\mathcal{O}(n \cdot \log^k n)$ for some constant k. Our algorithm achieves the same running time (because for every new input symbol, only numbers of bit length $\mathcal{O}(\log n)$ have to be manipulated). In contrast to our algorithm, the algorithm from [38] is non-streaming and does not work in logarithmic space.

7.3 Randomized streaming algorithms for free products

In [39], Waack proved that the word problem of a free product $G * H$ of two f.g. groups G and H can be solved in logspace if the word problems of G and H can be solved in logspace. Here, we show that Waack's reduction can be also used for randomized streaming algorithms.

For a group G and two subgroups $A, B \leq G$, the commutator subgroup $[A, B] \leq G$ is the group generated by all commutators $[a, b] = a^{-1}b^{-1}ab$ with $a \in A$ and $b \in B$. Let us denote with $F(\Sigma)$ the free group generated by the set Σ. A free group F is freely generated by the set $A \subseteq F$ if F is isomorphic to $F(A)$. It is well known that the free group $F_2 = F(\{a, b\})$ of rank 2 contains a copy of the free group $F(\mathbb{N})$ of countable infinite rank. For instance, the mapping $\phi : \mathbb{N} \to F_2$ with $\phi(n) = a^{-n}ba^n$ defines an injective homomorphism $\phi : F(\mathbb{N}) \to F_2$. The following group theoretic lemma underlies Waack's reduction:

▶ **Lemma 19** (c.f. [39, Proposition 4.2]). *Let G and H be groups. Then $[G, H]$ is a normal subgroup of the free product $G * H$ such that $(G * H)/[G, H] \cong G \times H$. Moreover, $[G, H]$ is a free group that is freely generated by the set of commutators $\{[g, h] \mid g \in G \setminus \{1\}, h \in H \setminus \{1\}\}$.*

▶ **Theorem 20.** *Let G and H be a f.g. groups for which there exist ϵ-injective randomized streaming algorithms $\mathcal{R} = (\mathcal{A}_n)_{n\geq 0}$ and $\mathcal{S} = (\mathcal{B}_n)_{n\geq 0}$, respectively. Then, there exists a $(4n^2 + 1)\epsilon$-injective randomized streaming algorithm $(\mathcal{C}_n)_{n\geq 0}$ for $G * H$ with space complexity $s(\mathcal{R}, n) + s(\mathcal{S}, n) + \mathcal{O}(\log n)$.*

The proof of Theorem 20 is a bit technical and can be found in [25, Section 8.2]. In order to test whether a word w is trivial in $G * H$, Waack checks whether the image of w in the quotient $(G * H)/[G, H] \cong G \times H$ is trivial (for which algorithms for G and H can be used). If this holds, then w represents an element of the free group $[G, H]$, which by the above remark can be embedded into F_2. Waack then computes from w the corresponding image in F_2. Basically, we follow the same strategy but only obtain the image in F_2 with high probability using Lemma 15. For F_2 (a linear group) we can then apply Theorem 8.

8 Lower bounds

In this section, we will construct groups with a large randomized streaming space complexity. We will make use of the disjointness problem from communication complexity. The *disjointness problem* is defined as follows: Alice (resp., Bob) has a bit string $u \in \{0, 1\}^n$ (resp., $v \in \{0, 1\}^n$) and their goal is to determine whether there is a position $1 \leq i \leq n$ such that $u[i] = v[i] = 1$ ($u[i]$ and $v[i]$ are the bits at position i). It is well known that the randomized communication complexity for the disjointness problem is $\Theta(n)$, see e.g. [21, Section 4.6].

▶ **Theorem 21.** *Let H be a f.g. non-abelian group and G be a f.g. infinite group. The randomized streaming space complexity of $H \wr G$ is $\Theta(n)$.*

Proof. Let $\mathcal{R} = (\mathcal{A}_n)_{n\geq 0}$ be a randomized streaming algorithm for the word problem of $H \wr G$. We show that we obtain a randomized communication protocol for the disjointness problem with communication cost $3 \cdot s(\mathcal{R}, 12n - 8)$.

Fix $n \geq 1$ and two elements $g, h \in H$ with $[g, h] \neq 1$. We can assume that g and h are generators of H. We also fix a finite generating set for G. Let $s := t_1 t_2 \cdots t_{n-1}$ be a word over the generators of G such that $t_1 t_2 \cdots t_i$ and $t_1 t_2 \cdots t_j$ represent different elements whenever $i \neq j$. Such a word exists since the Cayley graph of G is an infinite locally finite graph and hence contains an infinite ray. For a word $w = a_0 a_1 \cdots a_{n-1} \in \{0, 1\}^n$ and an element $x \in \{g, h, g^{-1}, h^{-1}\}$ define the word $w[x] = x^{a_0} t_1 x^{a_1} t_2 \cdots x^{a_{n-2}} t_{n-1} x^{a_{n-1}} s^{-1}$. It represents the element $(f_{w,x}, 1) \in H \wr G$ with $\mathsf{supp}(f_{w,x}) = \{t_1 \ldots t_i \mid i \in [0, n-1], w[i] = 1\}$ and $f_{w,x}(t) = x$ for all $t \in \mathsf{supp}(f_{w,x})$. Therefore, for two words $u, v \in \{0, 1\}^n$ we have $u[g]v[h]u[g^{-1}]v[h^{-1}] = 1$ in $H \wr G$ if and only if there is a position $i \in [0, n-1]$ with $u[i] = v[i] = 1$. Note that the length of the word $u[g]v[h]u[g^{-1}]v[h^{-1}]$ is $4(3n - 2) = 12n - 8$.

Our randomized communication protocol for the disjointness problem works as follows, where $u \in \{0, 1\}^n$ is the input for Alice and $v \in \{0, 1\}^n$ is the input for Bob.

- Alice reads the word $u[g]$ into $\mathcal{A}_{12n-8}$ and sends the resulting state to Bob.
- Bob continues the run in the state he received from Alice, reads the word $v[h]$ into the automaton and sends the resulting state back to Alice.
- Alice continues the run with the word $u[g^{-1}]$ and sends the resulting state of Bob.
- Bob continues the run with $v[h^{-1}]$ and finally accepts if the resulting state is accepting.

Both Alice and Bob use their random coins in order to make the random decisions in the PFA $\mathcal{A}_{12n-8}$. Clearly, the protocol is correct and its communication cost is $3 \cdot s(\mathcal{R}, 12n-8)$. We hence must have $3 \cdot s(\mathcal{R}, 12n-8) \in \Omega(n)$ which implies $s(\mathcal{R}, m) \in \Omega(m)$. ◀

In 1965, Thompson introduced three finitely presented groups $F < T < V$ acting on the unit-interval, the unit-circle and the Cantor set, respectively. Of these three groups, F received most attention (the reader should not confuse F with a free group). This is mainly due to the still open conjecture that F is not amenable. The group F consists of all homeomorphisms of the unit interval that are piecewise affine, with slopes a power of 2 and dyadic breakpoints. It is a finitely presented group. Important for us is the fact that F contains a copy of $F \wr \mathbb{Z}$ [17, Lemma 20]. Since F is non-abelian, Theorem 21 implies:

▶ **Corollary 22.** *The randomized streaming space complexity of Thompson's group F is $\Theta(n)$.*

Grigorchuk's group was introduced by Grigorchuk in [14]. It is a f.g. group of automorphisms of the infinite binary tree; the generators are usually denoted a, b, c, d and satisfy the identities $a^2 = b^2 = c^2 = d^2 = 1$ and $bc = cb = d, bd = db = c, dc = cd = b$. Grigorchuk's group is a f.g. infinite torsion group and was the first example of a group with intermediate growth and the first example of a group that is amenable but not elementary amenable.

▶ **Theorem 23.** *Let G be the Grigorchuk group. Then the following hold:*

- *The deterministic streaming space complexity of G is $\mathcal{O}(n^{0.768})$.*
- *The randomized streaming space complexity of G is $\Omega(n^{0.5})$.*

Proof sketch. The first statement follows from Theorem 3 and the upper growth bound $\exp(n^{0.768})$ for the Grigorchuk group; see [5]. For the second statement we use a non-abelian subgroup $K \leq G$ such that K contains a copy of $K \times K$, see [9, p. 262]. This allows a similar reduction from the disjointness problem as in the proof of Theorem 21; see [25, Theorem 9.3] for details. ◀

9 Open problems

We conclude with some open problems.

- Can the space bound $\mathcal{O}(s(\mathcal{R}, n)^2 + \log n)$ in Theorem 17 be reduced to $\mathcal{O}(s(\mathcal{R}, n) + \log n)$?
- What is the space complexity of randomized streaming algorithms for hyperbolic groups? The best complexity bound for the word problem for a hyperbolic group is LogCFL, which is contained in $\mathsf{DSPACE}(\log^2 n)$ [24].
- Is there a group that is not residually finite and for which there exists a randomized streaming algorithm with space complexity $o(n)$? An example of group that is not residually finite is the Baumslag-Solitar group $\mathsf{BS}(2, 3)$. The word problem for every Baumslag-Solitar group $\mathsf{BS}(p, q)$ can be solved in logarithmic space [41].

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

2 Anatolij W. Anissimov and Franz D. Seifert. Zur algebraischen Charakteristik der durch kontext-freie Sprachen definierten Gruppen. *Elektronische Informationsverarbeitung und Kybernetik*, 11(10–12):695–702, 1975.

3 Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010. doi:10.1007/s00037-010-0299-8.

4 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013. doi:10.1016/j.tcs.2012.12.028.

5 Laurent Bartholdi. The growth of Grigorchuk's torsion group. *International Mathematics Research Notices*, 20:1049–1054, 1998. doi:10.1155/S1073792898000622.

6 Laurent Bartholdi. Lower bounds on the growth of a group acting on the binary rooted tree. *International Journal of Algebra and Computation*, 11(01):73–88, 2001. doi:10.1142/S0218196701000395.

7 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 119:1–119:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.119.

8 John W. Cannon, William J. Floyd, and Walter R. Parry. Introductory notes on Richard Thompson's groups. *L'Enseignement Mathématique*, 42(3):215–256, 1996.

9 Pierre de la Harpe. *Topics in Geometric Group Theory.* University of Chicago Press, 2000.

10 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144, 1911. In German. doi:10.1007/BF01456932.

11 Nathanaël Fijalkow. The online space complexity of probabilistic languages. In *Proceedings of the International Symposium on Logical Foundations of Computer Science, LFCS 2016*, volume 9537 of *Lecture Notes in Computer Science*, pages 106–116. Springer, 2016. doi:10.1007/978-3-319-27683-0_8.

12 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

13 William Timothy Gowers and Emanuele Viola. Interleaved group products. *SIAM Journal on Computing*, 48(2):554–580, 2019. doi:10.1137/17M1126783.

14 Rostislav I. Grigorchuk. Burnside's problem on periodic groups. *Functional Analysis and Its Applications*, 14:41–43, 1980. doi:10.1007/BF01078416.

15 Rostislav I. Grigorchuk. On the gap conjecture concerning group growth. *Bulletin of Mathematical Sciences*, 4(1):113–128, 2014. doi:10.1007/s13373-012-0029-4.

16 Mikhail Gromov. Groups of polynomial growth and expanding maps. *Publications Mathématiques de L'Institut des Hautes Scientifiques*, 53:53–78, 1981. doi:10.1007/BF02698687.

17 Victor S. Guba and Mark V. Sapir. On subgroups of the R. Thompson group F and other diagram groups. *Matematicheskii Sbornik*, 190(8):3–60, 1999. doi:10.1070/SM1999v190n08ABEH000419.

18 Derek F. Holt, Sarah Rees, and Claas E. Röver. *Groups, Languages and Automata*, volume 88 of *London Mathematical Society Student Texts.* Cambridge University Press, 2017. doi:10.1017/9781316588246.

19 Richard M. Karp. Some bounds on the storage requirements of sequential machines and Turing machines. *Journal of the Association for Computing Machinery*, 14(3):478–489, 1967. doi:10.1145/321406.321410.

20 Daniel König and Markus Lohrey. Evaluation of circuits over nilpotent and polycyclic groups. *Algorithmica*, 80(5):1459–1492, 2018. doi:10.1007/s00453-017-0343-z.

21 Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997. doi:10.1017/CBO9780511574948.

22 Jörg Lehnert and Pascal Schweitzer. The co-word problem for the Higman-Thompson group is context-free. *Bulletin of the London Mathematical Society*, 39(2):235–241, February 2007. doi:10.1112/blms/bdl043.

23 Richard J. Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *Journal of the Association for Computing Machinery*, 24(3):522–526, 1977. doi:10.1145/322017.322031.

24 Markus Lohrey. Decidability and complexity in automatic monoids. *International Journal of Foundations of Computer Science*, 16(4):707–722, 2005. doi:10.1142/S0129054105003248.

25 Markus Lohrey and Lukas Lück. Streaming word problems. *CoRR*, abs/2202.04060, 2022. doi:10.48550/ARXIV.2202.04060.

26 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

27 Wilhelm Magnus. On a theorem of Marshall Hall. *Annals of Mathematics. Second Series*, 40:764–768, 1939. doi:10.2307/1968892.

28 Avinoam Mann. *How Groups Grow.* London Mathematical Society Lecture Note Series. Cambridge University Press, 2011. doi:10.1017/CBO9781139095129.

29 Charles F Miller III. Decision problems for groups – survey and reflections. In G. Baumslag and Charles F Miller III, editors, *Algorithms and classification in combinatorial group theory*, pages 1–59. Springer, 1992. doi:10.1007/978-1-4613-9730-4_1.

30 John Milnor. Growth of finitely generated solvable groups. *Journal of Differential Geometry*, 2(4):447–449, 1968. doi:10.4310/jdg/1214428659.

31 David E. Muller and Paul E. Schupp. Groups, the theory of ends, and context-free languages. *Journal of Computer and System Sciences*, 26:295–310, 1983. doi:10.1016/0022-0000(83)90003-X.

32 Alexei Myasnikov, Vitaly Roman'kov, Alexander Ushakov, and AnatolyVershik. The word and geodesic problems in free solvable groups. *Transactions of the American Mathematical Society*, 362(9):4655–4682, 2010. doi:10.1090/S0002-9947-10-04959-7.

33 Azaria Paz. *Introduction to Probabilistic Automata.* Academic Press, 1971. doi:10.1016/C2013-0-11297-4.

34 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.

35 Jeffrey Shallit and Yuri Breitbart. Automaticity I: properties of a measure of descriptional complexity. *Journal of Computer and System Sciences*, 53(1):10–25, 1996. doi:10.1006/jcss.1996.0046.

36 Hans-Ulrich Simon. Word problems for groups and contextfree recognition. In *Proceedings of Fundamentals of Computation Theory, FCT 1979*, pages 417–422. Akademie-Verlag, 1979.

37 Jacques Tits. Free subgroups in linear groups. *Journal of Algebra*, 20:250–270, 1972. doi:10.1016/0021-8693(72)90058-0.

38 Alexander Ushakov. Algorithmic theory of free solvable groups: Randomized computations. *Journal of Algebra*, 407:178–200, 2014. doi:10.1016/j.jalgebra.2014.02.014.

39 Stephan Waack. The parallel complexity of some constructions in combinatorial group theory. *Journal of Information Processing and Cybernetics, EIK*, 26:265–281, 1990.

40 Bertram A. F. Wehrfritz. On finitely generated soluble linear groups. *Mathematische Zeitschrift*, 170:155–167, 1980. doi:10.1007/BF01214771.

41 Armin Weiß. A logspace solution to the word and conjugacy problem of generalized Baumslag-Solitar groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*. American Mathematical Society, 2016. doi:10.1090/conm/677.

42 Joseph A. Wolf. Growth of finitely generated solvable groups and curvature of Riemannian manifolds. *Journal of Differential Geometry*, 2(4):421–446, 1968. doi:10.4310/jdg/1214428658.

A Universal Skolem Set of Positive Lower Density

Florian Luca
School of Mathematics, University of the Witwatersrand, Johannesburg, South Africa
Research Group in Algebraic Structures & Applications, King Abdulaziz University, Saudi Arabia
Centro de Ciencias Matemáticas UNAM, Morelia, Mexico
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany

James Worrell
Department of Computer Science, University of Oxford, UK

Abstract

The Skolem Problem asks to decide whether a given integer linear recurrence sequence (LRS) has a zero term. Decidability of this problem has been open for many decades, with little progress since the 1980s. Recently, a new approach was initiated via the notion of a Skolem set – a set of positive integers relative to which the Skolem Problem is decidable. More precisely, $\mathcal{S}$ is a Skolem set for a class $\mathcal{L}$ of integer LRS if there is an effective procedure that, given an LRS in $\mathcal{L}$, decides whether the sequence has a zero in $\mathcal{S}$. A recent work exhibited a Skolem set for the class of all LRS that, while infinite, had density zero. In the present work we construct a Skolem set of positive lower density for the class of simple LRS.

2012 ACM Subject Classification Computing methodologies → Symbolic and algebraic algorithms; Computing methodologies → Number theory algorithms

Keywords and phrases Linear Recurrence Sequences, Skolem Problem, Exponential Diophantine Equations, Sieve Methods

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.73

Funding *Joël Ouaknine*: Also affiliated with Keble College, Oxford as `emmy.network` Fellow, and supported by DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).

1 Introduction

An (integer) linear recurrence sequence (LRS) $\langle u_n \rangle_{n=0}^{\infty}$ is a sequence of integers satisfying a recurrence of the form

$$u_{n+k} = a_1 u_{n+k-1} + \cdots + a_k u_n \qquad (n \in \mathbb{N}), \tag{1}$$

where the coefficients $a_1, \ldots, a_k$ are integers. The celebrated theorem of Skolem, Mahler, and Lech [23, 16, 14] describes the set of zero terms of such a recurrence:

▶ **Theorem 1.** *Given an integer linear recurrence sequence $\langle u_n \rangle_{n=0}^{\infty}$, the set $\{n \in \mathbb{N} : u_n = 0\}$ is a union of finitely many arithmetic progressions together with a finite set.*

The statement of Theorem 1 can be refined by considering the notion of *non-degeneracy* of an LRS. An LRS is non-degenerate if in its minimal recurrence the quotient of no two distinct roots of the characteristic polynomial is a root of unity. A given LRS can be effectively decomposed as the merge of finitely many non-degenerate sequences, some of which may be identically zero. The core of the Skolem-Mahler-Lech Theorem is the fact that a non-zero non-degenerate linear recurrence sequence has finitely many zero terms. Unfortunately, all known proofs of this last result are ineffective: it is not known how to compute the finite set

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 73; pp. 73:1–73:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of zeros of a given non-degenerate linear recurrence sequence. It is readily seen that existence of a procedure to do so is equivalent to the existence of a procedure to decide whether an arbitrary given LRS has a zero term. The problem of deciding whether an LRS has a zero term is variously known as Skolem's Problem or the Skolem-Pisot Problem. We refer to [5, Chapter 6] and [24, Chapter X] for expository accounts of the Skolem-Mahler-Lech Theorem and discussion of the ineffectiveness of known proofs.

Decidability of Skolem's Problem is known only for certain special cases, based on the relative order of the absolute values of the characteristic roots. Say that a characteristic root λ is *dominant* if its absolute value is maximal among all the characteristic roots. Decidability is known in case there are at most 3 dominant characteristic roots, and also for recurrences of order at most 4 [18, 26]. However for LRS of order 5 it is not currently known how to decide Skolem's Problem. For a (highly restricted) subclass of LRS, the paper [1] obtains nearly matching complexity lower and upper bounds for the problem.

In computer science, Skolem's Problem lies at the heart of key decision problems in formal power series [21, 4], stochastic model checking [20], control theory [6, 10], and loop termination [19]. The problem is also closely related to membership problems on commutative matrix groups and semigroups, as considered in [7, 13]. We note that in several of the above-mentioned citations, the Skolem Problem is used as a reference to show hardness of other open decision problems.

A recent paper [15] initiated a new approach to the decidability of Skolem's Problem. Rather than place syntactic restrictions on sequences (e.g., on the order of the recurrence or dominance pattern of the characteristic roots), the idea is to restrict the domain in which to search for zeros. To this end, [15] introduced the following definition.

▶ **Definition 2.** *An infinite set $\mathcal{S} \subseteq \mathbb{N}$ is a* Skolem set *for a class $\mathcal{L}$ of LRS if there is an effective procedure that, given an LRS $\langle u_n \rangle_{n=0}^{\infty}$ in $\mathcal{L}$, determines whether there exists $n \in \mathcal{S}$ with $u_n = 0$.*

The main technical contribution of [15] was to exhibit a Skolem set for the class of all LRS. Such sets are called *Universal Skolem sets.* Specifically, define $f : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ by $f(n) = \lfloor \sqrt{\log n} \rfloor$, and inductively define the sequence $\langle s_n \rangle_{n=0}^{\infty}$ by $s_0 = 1$ and $s_n = n! + s_{f(n)}$ for $n > 0$. Then $\{s_n : n \in \mathbb{N}\}$ is a Universal Skolem set. This construction yields a very sparse set, which has density zero. This leads to the question of whether one can construct Skolem sets of positive lower density. Decidability of the Skolem Problem is, by definition, equivalent to the assertion that $\mathbb{N}$ is a Universal Skolem set (in fact, decidability follows already from the existence of Universal Skolem set that contains an infinite arithmetic progression). Hence seeking Skolem sets of increasingly higher density is a natural direction in which to make progress on the Skolem Problem.

The main result of the present paper exhibits a set $\mathcal{S}$ of positive lower density, i.e., having

$$\liminf_{n\to\infty} \frac{|\mathcal{S} \cap \{1,\dots,n\}|}{n} > 0\,,$$

such that $\mathcal{S}$ is a Skolem set for the class of simple LRS. For short we call $\mathcal{S}$ a *Simple Universal Skolem set.* Recall that a simple LRS is one for which the characteristic roots of its minimal-order recurrence are simple. The Skolem Problem for simple LRS is open already for LRS of order 5.

The construction of $\mathcal{S}$ is fundamentally different from the example in [15]. Roughly speaking, our set consists of positive integers n that admit many representations of the form $n = Pq + a$, with P, q prime and q, a logarithmic in n. The formal definition of the set is given in Section 3. In Section 4 we show that one can decide, given a simple LRS

$\langle u_n\rangle_{n=0}^{\infty}$, whether there exists $n \in \mathcal{S}$ with $u_n = 0$. The crucial ingredients here are results of Schlickewei and Schmidt [22] that give explicit upper bounds on the number of solutions of certain exponential Diophantine equations. Such results have previously been used to give effective upper bounds on the *number* of zeros of a given non-degenerate LRS (see [8, Theorem 2.7]), thereby strengthening the statement of the Skolem-Mahler-Lech Theorem (which, recall, asserts mere finiteness of the number of zeros). However such bounds do not obviously yield a solution to the Skolem Problem itself, which would instead require effective bounds on the *magnitude* of the zeros of an LRS. The essential novelty of our approach is, via the notion of representation, to leverage bounds on the number of solutions of exponential Diophantine equations to obtain bounds on the magnitude of the zeros of a simple LRS that lie in $\mathcal{S}$. In Section 5, we show that the set $\mathcal{S}$ has positive lower density. Here, classical number-theoretic techniques for upper bounding the number of pairs of primes in certain linear relations (such as twin primes) play the main role.

As discussed in the Conclusion, an extended version of this paper will show that for the set $\mathcal{S}$ introduced in this paper, the Skolem Problem is decidable relative to $\mathcal{S}$ for the class of all LRS, not just the simple ones. In the Conclusion we also briefly discuss the prospects of refining our analysis of the density of $\mathcal{S}$.

2 Background

In this section we briefly recall some relevant notation and definitions concerning number fields. We also recall a result from [9, 25] on unit equations that is derived from the Subspace Theorem and that will play a crucial role in our construction of a Simple Universal Skolem set.

Throughout we use the Vinogradov notation $f \ll g$ for $f \in O(g)$.

Recall that a *number field* $\mathbb{K}$ is a subfield of $\mathbb{C}$ that is finite dimensional as a vector space over $\mathbb{Q}$. The subring of algebraic integers in $\mathbb{K}$ is denoted $\mathcal{O}_{\mathbb{K}}$. For such a field $\mathbb{K}$, we denote by $\mathrm{Gal}(\mathbb{K}/\mathbb{Q})$ the group of automorphisms of $\mathbb{K}$. Given $\alpha \in \mathbb{K}$, the norm of α is defined by

$$N_{\mathbb{K}/\mathbb{Q}}(\alpha) = \prod_{\sigma \in \mathrm{Gal}(\mathbb{K}/\mathbb{Q})} \sigma(\alpha)\,.$$

The *norm* $N_{\mathbb{K}/\mathbb{Q}}(\alpha)$ is rational for all $\alpha \in \mathbb{K}$; moreover $N_{\mathbb{K}/\mathbb{Q}}(\alpha)$ is an integer if $\alpha \in \mathcal{O}_{\mathbb{K}}$. Clearly we have $|N(\alpha)| < M^{d_{\mathbb{K}}}$, where $d_{\mathbb{K}}$ is the degree of $\mathbb{K}$ and

$$M := \max_{\sigma \in \mathrm{Gal}(\mathbb{K}/\mathbb{Q})} |\sigma(\alpha)|$$

is the *house* of α.

We say that $\alpha, \beta \in \mathbb{K}$ are *multiplicatively dependent* if there exist integers k, ℓ, not both zero, such that $\alpha^k = \beta^\ell$. Observe that if $\alpha \in \mathbb{K}$ is not a root of unity then given $\sigma \in \mathrm{Gal}(\mathbb{K}/\mathbb{Q})$, every multiplicative relations $\alpha^k = \sigma(\alpha)^\ell$ is such that $k = \pm\ell$. Indeed, repeatedly applying σ to this relation we deduce that $\alpha^{k^d} = (\sigma^d(\alpha))^{\ell^d}$ for all $d \geq 1$. In particular, choosing d to be the order of σ we get that $\alpha^{k^d} = \alpha^{\ell^d}$ and hence $k = \pm\ell$.

We recall that every ideal in $\mathcal{O}_{\mathbb{K}}$ can be written uniquely as the product of prime ideals. Given a rational prime $p \in \mathbb{Z}$, we say that a prime ideal $\mathfrak{p}$ lies above p if $\mathfrak{p}$ is a factor of $p\mathcal{O}_{\mathbb{K}}$. In this case we have that $p \mid N_{\mathbb{K}/\mathbb{Q}}(\alpha)$ for all $\alpha \in \mathfrak{p}$.

We will need a result of Schlickewei and Schmidt [22] that gives upper bounds on the number of integer solutions of certain exponential Diophantine equations. The result (which we have specialised to our setting) is as follows:

▶ **Theorem 3** ([22, Theorem 1]). *For $i = 1, \dots, \ell$, let α_i, β_i, C_i be non-zero and lie in a number field of degree d over $\mathbb{Q}$. Suppose that the system of equations*

$$\alpha_i^{z_1}\beta_i^{z_2} = \alpha_j^{z_1}\beta_j^{z_2} \quad i, j \in \{1, \dots, \ell\}$$

has no non-zero solution in integers z_1, z_2. Then the number of solutions of the equation

$$\sum_{i=1}^{\ell} C_i\alpha_i^{x_1}\beta_i^{x_2} = 0 \tag{2}$$

in integers x_1, x_2 for which no proper sub-sum of the left-hand side vanishes is at most $2^{35\ell^2}d^{6\ell^2}$.

3 The Definition of the Set $\mathcal{S}$

In this section we give the definition of our Simple Universal Skolem set $\mathcal{S}$.

For a positive real number $x > 0$, denote by $\log x$ the natural logarithm of x. For a positive integer $k \geq 1$, we inductively define the iterated logarithm function $\log_k x$ as follows: $\log_1 x := \log x$, and for $k \geq 2$ we set $\log_k x := \max\{1, \log_{k-1}(\log x)\}$. Thus, for x sufficiently large, $\log_k x$ is the k-fold iterate of log applied to x. We omit the subscript when $k = 1$.

Fix a positive integer parameter X. We define disjoint intervals

$$A(X) := \left[\log_2 X, \sqrt{\log X}\right] \quad \text{and} \quad B(X) := \left[\frac{\log X}{\sqrt{\log_3 X}}, \frac{2\log X}{\sqrt{\log_3 X}}\right].$$

We further define a *representation* of an integer $n \in [X, 2X]$ to be a triple (q, P, a) such that $q \in A(X)$, $a \in B(X)$, P and q are prime, and $n = Pq + a$. We say that two representations (q, P, a) and (q', P', a') of the same number *overlap* if either $a + q = a' + q'$ or $a - q = a' - q'$. It is clear that two overlapping representations $(q, P, a) \neq (q', P', a')$ must have both $a \neq a'$ and $q \neq q'$.

We denote by $r(n)$ the number of representations of n. Finally we put

$$\mathcal{S}(X) := \{n \in [X, 2X] \ : \ r(n) > \log_4 X \text{ and no two representations of } n \text{ overlap}\}$$

and we define

$$\mathcal{S} := \bigcup_{X \geq 1} \mathcal{S}(X).$$

This completes the definition of the set $\mathcal{S}$. The rest of the paper is devoted to showing that $\mathcal{S}$ is a Simple Universal Skolem set and that it has positive lower density.

4 Solving the Simple Skolem Problem Relative to $\mathcal{S}$

The following result is the first half of the argument that the set $\mathcal{S}$, defined in Section 3, is a Simple Universal Skolem set. We use the bounds on the solutions of exponential Diophantine equations stated in Section 2 to show that the Skolem Problem for simple LRS is decidable relative to $\mathcal{S}$.

▶ **Proposition 4.** *Given a non-degenerate simple linear recurrence sequence $\langle u_n \rangle_{n=0}^{\infty}$, there is an effectively computable upper bound on the set $\{n \in \mathcal{S} : u(n) = 0\}$.*

Proof. Fix a non-degenerate simple linear recurrence sequence $\langle u_n\rangle_{n=0}^\infty$ with distinct characteristic roots $\alpha_1,\ldots,\alpha_\ell$. Recall that non-degeneracy is the condition that α_i/α_j is not a root of unity for all $i\neq j$. It is well-known that u_n admits an exponential-sum representation

$$u_n=\sum_{i=1}^{\ell} C_i\alpha_i^n\,,$$

where the constants C_i lie in the number field $\mathbb{K}:=\mathbb{Q}(\alpha_1,\ldots,\alpha_\ell)$. Multiplying the sequence $\langle u_n\rangle_{n=0}^\infty$ by a suitable integer, we may assume without loss of generality that the C_i lie in the ring of integers $\mathcal{O}_\mathbb{K}$.

Suppose that $n\in\mathcal{S}$ is such that $u_n=0$. By the definition of $\mathcal{S}$ we have that $n\in\mathcal{S}(X)$ for some positive integer X. We show that there is an upper bound on X that is effectively computable from the description of the sequence $\langle u_n\rangle_{n=0}^\infty$. Since $n\leq 2X$ this gives the desired effective upper bound on n.

Fix $n\in\mathcal{S}(X)$ such that $u_n=0$ and consider a representation (q,P,a) of n. Then $n=qP+a\geq X$, and so

$$P\geq\frac{X-a}{q}\geq\frac{X-\log X}{\sqrt{\log X}}>\sqrt{X} \tag{3}$$

for X sufficiently large.

Let $\mathfrak{p}$ be a prime ideal of $\mathcal{O}_\mathbb{K}$ lying above P and let $\sigma\in\mathrm{Gal}(\mathbb{K}/\mathbb{Q})$ be a Frobenius automorphism corresponding to $\mathfrak{p}$, that is, such that $\sigma(\alpha)\equiv\alpha^P \bmod \mathfrak{p}$ for all $\alpha\in\mathcal{O}_\mathbb{K}$. From $u_n=0$ and $n=qP+a$ we have

$$\sum_{i=1}^{\ell} C_i\alpha_i^{qP+a}=0\,.$$

Since $\sigma(\alpha_i)\equiv\alpha_i^P \bmod \mathfrak{p}$ for all $i\in\{1,\ldots,\ell\}$, we can write

$$\sum_{i=1}^{\ell} C_i\sigma(\alpha_i)^q\alpha_i^a\equiv 0 \bmod \mathfrak{p}\,. \tag{4}$$

Taking norms, we see that P divides

$$N:=N_{\mathbb{K}/\mathbb{Q}}\left(\sum_{i=1}^{\ell} C_i\sigma(\alpha_i)^q\alpha_i^a\right)\,.$$

Moreover, we have the inequality

$$|N|\leq M^{(1+q+a)d_\mathbb{K}}\,, \tag{5}$$

where $M=\max\{|\alpha_i|,|\beta_i|,|C_i|:1\leq i\leq\ell\}$ and $d_\mathbb{K}$ is the degree of $\mathbb{K}$ over $\mathbb{Q}$.

From the fact that $q\in A(X)$ and $a\in B(X)$, we see that for X sufficiently large we have $a,q<\frac{\log X}{8d_\mathbb{K}\log M}$. In this case, Equation (5) yields $N\ll X^{1/4}$. Hence for X large enough, using Equation (3), we have $N<X^{1/2}<P$. But N is an integer that is divisible by P, so it must be zero. We conclude that, for sufficiently large X, the left-hand side of (4) is zero; that is,

$$\sum_{i=1}^{\ell} C_i\sigma(\alpha_i)^q\alpha_i^a=0\,. \tag{6}$$

Carrying over the terminology for representations, let us say that two different solutions q, a and q', a' of Equation (6) *overlap* if either $a+q = a'+q'$ or $a-q = a'-q'$. Our goal is to give an explicit upper bound on the size of any collection of pairwise non-overlapping solutions of (6). For this, it is enough to give an upper bound under the additional assumption that no vanishing proper sub-sum of the left-hand side of (6) vanishes. Multiplying the latter quantity by the number (at most 2^ℓ) of possible sub-sums gives the desired upper bound on the total number of non-overlapping solutions.

Consider an instance of Equation (6) for which no proper sub-sum vanishes. Let $\mathcal{G}$ be the additive group of vectors $(z_1, z_2) \in \mathbb{Z}^2$ such that

$$\sigma(\alpha_i)^{z_1}\alpha_i^{z_2} = \sigma(\alpha_j)^{z_1}\alpha_j^{z_2}$$

for all $1 \leq i < j \leq \ell$. Note that there exist effective upper bounds on the magnitude of the entries a basis of $\mathcal{G}$ [17].

For $(z_1, z_2) \in \mathcal{G}$ we have $\sigma(\alpha_i/\alpha_j)^{z_1} = (\alpha_i/\alpha_j)^{-z_2}$. As shown in Section 2, since α_i/α_j is not a root of unity we must have either $z_1 = z_2$ or $z_1 = -z_2$. We thus have the following three possibilities for the form of the group $\mathcal{G}$ – either $\mathcal{G} = \{\mathbf{0}\}$, $\mathcal{G} = \{(z,z) : z \in m\mathbb{Z}\}$, or $\mathcal{G} = \{(z,-z) : z \in m\mathbb{Z}\}$, where $m \in \mathbb{Z}$. We consider three cases according to these three eventualities.

Case (i): $\mathcal{G} = \{\mathbf{0}\}$. Applying Theorem 3, the total number of solutions (overlapping or not) of Equation (6) in this case is at most $2^{35\ell^2}(d_{\mathbb{K}})^{6\ell^2}$.

Case (ii): $\mathcal{G} = \{(z,z) : z \in m\mathbb{Z}\}$ for some $m \in \mathbb{Z}$. In this case we have that $\sigma(\alpha_i)^m\alpha_i^m$ takes the same value for all $i \in \{1, \ldots, \ell\}$. Dividing Equation (6) by the $\lfloor q/m \rfloor$-th power of this common value we get

$$\sum_{i=1}^{\ell} \widetilde{C}_i \alpha_i^{a-q} = 0\,,$$

where each constant $\widetilde{C}_i$ is uniquely determined by C_i and the residue of q modulo m.

In other words, for every solution (q, a) of Equation (6), we have that $q - a$ is a zero of one of a finite number (at most m^ℓ) of non-degenerate LRS, each of which takes values in $\mathbb{K}$ and has order at most k. Applying Theorem 3, the number of different values of $q - a$ over all solutions (q, a) is at most $(2k)^{35k^2}(d_{\mathbb{K}})^{6k^2}m^\ell$. But the latter quantity is then a bound on the cardinality of a set of pairwise non-overlapping solutions of Equation (6).

Case (iii): $\mathcal{G} = \{(z,-z) : z \in m\mathbb{Z}\}$ for some $m \in \mathbb{Z}$. The argumentation is almost exactly as in Case (ii). We have that $\sigma(\alpha_i)^m\alpha_i^{-m}$ takes the same value for all $i \in \{1, \ldots, \ell\}$. Dividing Equation (6) by the $\lfloor q/m \rfloor$-th power of this common value we get

$$\sum_{i=1}^{\ell} \widetilde{C}_i \alpha_i^{a+q} = 0$$

where each constant $\widetilde{C}_i$ is uniquely determined by C_i and the residue of q modulo m. The argument now follows exactly as in Case (ii). In particular, we get the same upper bound on the number of solutions under Case (iii) as under Case (ii).

We can now summarise and wrap up. If $n \in \mathcal{S}(X)$ is a zero of u_n then for every representation $n = qP + a$ it holds that (q, a) is a solution of Equation (6). Moreover, since $n \in \mathcal{S}(X)$, no two representations of n are overlapping. Since we have an effective upper bound on the cardinality of a set of pairwise non-overlapping solutions of (6), we get an effective upper bound (that does not depend on n) for the number of representations of n. Finally, since by the definition of $\mathcal{S}(X)$ the number of representations is at least $\log_4 X$, we obtain the desired upper bound on X. ◀

5 The Set $\mathcal{S}$ has Positive Lower Density

Our goal in this section is to show that the set $\mathcal{S}$ has positive lower density. The key tool is the following result, (see [11, Chapter 2.6, Theorem 2.3]), derived using Selburg's upper-bound sieve, that bounds from above the number of times that two linear forms simultaneously take prime values.

▶ **Theorem 5.** *Let a_1, a_2, b_1, b_2 be integers such that*

$$E := |a_1 a_2 (a_1 b_2 - a_2 b_1)|$$

is non-zero. Then

$$|\{t \leq X : a_1 t + b_1, a_2 t + b_2 \textit{ both prime}\}| \ll \frac{X}{(\log X)^2} \frac{E}{\varphi(E)}\,,$$

where the implied constant is independent of a_1, a_2, b_1, b_2, and φ denotes Euler's totient function.

To help illustrate this result, observe that in case $a_1 = a_2 = 1$, $b_1 = 0$, and $b_2 = 2$, Theorem 5 gives an upper bound on the number of twin primes less than X.

We will also need the following straightforward proposition. (Note that here and in the rest of this section, variables p and q always range over prime numbers.)

▶ **Proposition 6.** $\sum_{q \in A(X)} \frac{1}{q} \sim \log_3 X$

Proof. We use the fact (see [3, Theorem 13.6]) that there exists an absolute constant c such that

$$\sum_{p \leq t} \frac{1}{p} = \log\log t + c + O\left(\frac{1}{\log t}\right)$$

for all $t \geq 2$. Using this fact, we have

$$\begin{aligned}
\sum_{q \in A(X)} \frac{1}{q} &= \log\log\sqrt{\log X} - \log\log\log_2 X + o(1) \\
&= \log_3 X - \log_4 X + o(1) \\
&\sim \log_3 X
\end{aligned}$$

for large enough X. ◀

We further note the following useful fact concerning the Euler function (see [3, Exercise 2.10(xii)]):

$$\sum_{k=1}^{n} \frac{k}{\varphi(k)} \ll n\,. \tag{7}$$

The rest of this section is devoted to a proof of the following result:

▶ **Theorem 7.** *The set $\mathcal{S}$, defined in Section 3, has strictly positive lower density.*

Recall from Section 3 that $\mathcal{S} := \bigcup_{X \geq 1} \mathcal{S}(X)$, where $\mathcal{S}(X) \subseteq [X, 2X]$. Hence, to prove that $\mathcal{S}$ has positive lower density, it suffices to show that $|\mathcal{S}(X)| \gg X$ for X sufficiently large. We first argue that for sufficiently large X we have

$$|\{n \in [X, 2X] : r(n) > \log_4 X\}| \gg X\,. \tag{8}$$

After that, we deal with those n that have overlapping representations (cf. the definition of $\mathcal{S}(X)$ in Section 3).

To prove (8) we use the moment method. To set this up, let

$$S_i := \sum_{\substack{n\in[X,2X]\\ r(n)>\log_4 X}} r(n)^i \qquad \text{for } i \in \{0,1,2\}\,.$$

The inequality (8) is equivalent to the assertion that $S_0 \gg X$. Thinking of S_1 as the inner product of the vector

$$\langle r(n) : n \in [X,2X], r(n) > \log_4 X\rangle$$

and the constant all 1's vector, applying the Cauchy-Schwartz inequality we get that

$$S_2 S_0 \geq S_1^2.$$

To show that $S_0 \gg X$ we will use a lower bound on S_1 and an upper bound on S_2.

5.1 Lower bound on S_1

In this section we show that $S_1 \gg X\sqrt{\log_3 X}$.

Fix $q \in A(X)$. Then for $P \in \left[\frac{X}{q}, \frac{1.5X}{q}\right]$ we have that $qP \in [X, 1.5X]$. Furthermore, if $a \in B(X)$, then for sufficiently large X we have $a < \log X < 0.5X$, so that $qP + a \in [X, 2X]$. Thus, for large X and each fixed prime $q \in A(X)$, the number of representations (q, P, a) with $qP + a \in [X, 2X]$ is at least the product of the number of primes $P \in \left[\frac{X}{q}, \frac{1.5X}{q}\right]$ and the cardinality of the set $B(X)$. But, for large enough X, the number of primes $P \in \left[\frac{X}{q}, \frac{1.5X}{q}\right]$ is

$$\pi\left(\frac{1.5X}{q}\right) - \pi\left(\frac{X}{q}\right) > \frac{0.3X}{q\log X}\,.$$

Thus, for fixed q, the number of representations (q, P, a) with $qP + a \in [X, 2X]$ is at least

$$\frac{0.3X}{q\log X}\,\frac{\log X}{\sqrt{\log_3 X}} \gg \frac{X}{q\sqrt{\log_3 X}}\,, \tag{9}$$

where the implied constant is independent of q.

Summing the lower bound (9) over $q \in A(X)$, we have

$$\begin{aligned}\sum_{n\in[X,2X]} r(n) \quad &\gg \quad \frac{X}{\sqrt{\log_3 X}}\left(\sum_{q\in A(X)} \frac{1}{q}\right)\\ &\gg \quad X\sqrt{\log_3 X} \quad \text{(by Proposition 6).}\end{aligned}$$

Finally, in order to get a lower bound on S_1 we must remove from the left-hand side above, the summands $r(n)$ for which $r(n) \leq \log_4 X$. But the contribution of these to the total sum is $O(X\log_4 X) = o(X\sqrt{\log_3 X})$, and so we conclude that $S_1 \gg X\sqrt{\log_3 X}$.

For future reference, we observe that essentially the same argument shows that $S_1 \ll X\sqrt{\log_3 X}$. Indeed, for each fixed $q \in A(X)$, the number of primes P such that $qP \in [X, 2X]$ is $\ll \frac{X}{q\log X}$. Since the number of $a \in B(X)$ is at most $\frac{\log X}{\sqrt{\log_3 X}}$, the number of representations (q, P, a) such that $qP + a \in [X, 2X]$ is $\ll \frac{X}{q\sqrt{\log_3 X}}$. Now, summing over $q \in A(X)$ and using Proposition 6, we obtain the bound $S_1 \ll X\sqrt{\log_3 X}$.

5.2 Upper bound on S_2

Our goal is to show that $S_2 \ll X \log_3 X$. To this end we consider S_2 as the number of pairs of representations $(q, P, a), (q', P', a')$ such that $qP + a = q'P' + a'$, with the common sum lying in the interval $[X, 2X]$. We break the count down into three cases.

Case (i). Let us first consider the number of such pairs with $a = a'$. In this case we have $qP = q'P'$. But then, since q and q' are small and P and P' are large, we get that $q = q'$ and $P = P'$, and hence $(q, P, a) = (q', P', a')$. Thus the number of such pairs is equal to S_1. But, as noted at the conclusion of Section 5.1, we have $S_1 \ll X\sqrt{\log_3 X}$.

Case (ii). We next consider the number of pairs of representations with $a \neq a'$ and $q = q'$. In this case we have

$$P - P' = m, \qquad \text{where} \qquad m := \frac{a - a'}{q}.$$

By Theorem 5, for each fixed m the number of primes $P \le X/q$ such that $P + m$ is also prime is

$$\ll \frac{X}{q(\log X)^2} \frac{m}{\varphi(m)}.$$

Furthermore, for fixed q and m, the number of choices of a, a' with $m = \frac{a-a'}{q}$ is at most the number of choices of a, that is, at most $\frac{2\log X}{\sqrt{\log_3 X}}$. Thus, for fixed q and m, the total number of pairs of representations $(q, P, a), (q', P', a')$ with $qP + a = q'P' + a' \in [X, 2X]$ and $m = \frac{a-a'}{q}$ is

$$\begin{aligned} &\ll \frac{X}{q(\log X)^2} \frac{m}{\varphi(m)} \frac{2\log X}{\sqrt{\log_3 X}} \\ &= \frac{2X}{q(\log X)\sqrt{\log_3 X}} \frac{m}{\varphi(m)}. \end{aligned}$$

From the fact that $a, a' \in A(X)$ we have $m \le \frac{2\log X}{q\sqrt{\log_3 X}}$. Summing up over all values of m, we get that the total number of solutions for fixed q is

$$\begin{aligned} &\ll \frac{2X}{q(\log X)\sqrt{\log_3 X}} \left(\sum_{m \le \frac{2\log X}{q\sqrt{\log_3 X}}} \frac{m}{\varphi(m)} \right) \\ &\ll \frac{X}{q^2 \log_3 X}. \end{aligned}$$

In the above, we used the fact (see Equation (7)) that $\sum_{m \le t} m/\varphi(m) \ll t$.

Now we sum up over $q > \log_2 X$, getting that the number of such solutions is at most

$$\frac{X}{\log_3 X} \sum_{q > \log_2 X} \frac{1}{q^2} \ll \frac{X}{(\log_2 X)\log_3 X} = o(X).$$

Case (iii). Finally, let us count the rest of the solutions; namely the ones for which $a \neq a'$ and $q \neq q'$. Fixing a, a', q, q' we have

$$qP - q'P' = a' - a.$$

The general solution of the above equation in integers P and P' can be written in the form $P = p_0 + q't$ and $P' = p_0' + qt$, where t is an integer parameter and p_0, p_0' is a particular solution, chosen to be minimal among positive integer solutions (simultaneously, in both

coordinates). The condition that $Pq + a \leq 2X$ implies that $P \leq 2X/q$ and hence that $t \leq \frac{2X}{qq'}$.
Using the assumption $a \neq a'$, we can apply Theorem 5 to deduce that the number of $t \leq \frac{2X}{qq'}$ such that both $p_0 + q't$ and $p_0' + qt$ are prime is

$$\ll \frac{X}{qq'(\log X)^2} \frac{|a-a'|}{\varphi(|a-a'|)} .$$

We keep a and q fixed and sum up over $q' \neq q$ and $a' \neq a$, getting a bound of

$$\begin{aligned} &\ll \frac{X}{q(\log X)^2} \left(\sum_{q' \in A(X)} \frac{1}{q'} \right) \sum_{\substack{a' \in B(X) \\ a \neq a}} \frac{|a-a'|}{\varphi(|a-a'|)} \\ &\ll \frac{X}{q(\log X)^2} \cdot \log_3 X \cdot \frac{2\log X}{\sqrt{\log_3 X}} \quad \text{(by Proposition 6 and Equation 7)} \\ &= \frac{X\sqrt{\log_3 X}}{q \log X} . \end{aligned}$$

To conclude the argument, we sum up over all a's and all q's, getting an upper bound

$$\begin{aligned} &\ll \frac{X\sqrt{\log_3 X}}{\log X} \frac{2\log X}{\sqrt{\log_3 X}} \left(\sum_{q \in A(X)} \frac{1}{q} \right) \\ &\ll X \log_3 X \quad \text{(by Proposition 6).} \end{aligned}$$

Combining the bounds in the above three cases, we conclude that $S_2 \ll X \log_3 X$.

5.3 Putting Things Together

From the Cauchy-Schwarz inequality $S_0 S_2 \geq S_1^2$ and the above-established bounds $S_1 \gg X\sqrt{\log_3 X}$ and $S_2 \ll X \log_3 X$, we get that $S_0 \gg X$.

To transform the lower bound on S_0 to one on $\mathcal{S}(X)$, it remains to estimate the number of $n \in [X, 2X]$ that admit two overlapping representations. We claim that the total number of such n is $o(X)$. From this we conclude that $|\mathcal{S}(X)| \gg X$, which was our ultimate goal.

We conclude by justifying the preceding claim. For this it suffices to show that the number of pairs of overlapping representations in the interval $[X, 2X]$ is $o(X)$. To this end, consider two representations $(q, P, a) \neq (q', P', a')$ of the same number $n \in [X, 2X]$. Assume that $q + a = q' + a'$ (the argument in case $q - a = q' - a'$ requires only minor changes).

Now $n - (q + a) = q(P - 1) = q'(P' - 1)$, and so $qP - q'P' = (q' - q)$. Furthermore, as noted in Section 3, for two such overlapping representations, we have $q \neq q'$. Thus the general solution of the equation $qP - q'P' = (q' - q)$ in positive integers P and P' has the form $P = 1 + tq'$ and $P' = 1 + qt$ for a nonnegative integer parameter $t \ll X/qq'$. By Theorem 5, the number of t such that both $1 + tq$ and $1 + tq'$ are prime is

$$\begin{aligned} &\ll \frac{X}{qq'(\log X)^2} \frac{|q-q'|}{\varphi(|q-q'|)} \\ &\ll \frac{X \log_3 X}{qq'(\log X)^2} . \end{aligned}$$

In the above we have used the inequality $m/\varphi(m) \ll \log_2 m$ [12, Theorem 328], with $m = |q - q'| < \log X$ for large X.

The argument above shows that for $q \neq q'$, the number of pairs of primes P, P' for which there exist pairs of overlapping representations $(q, P, a), (q', P, a')$ of some $n \in [X, 2X]$ is at most $\frac{X \log_3 X}{qq'(\log X)^2}$. Summing up over the at most $\log X$ possible values of $a \in B(X)$, we see that for fixed $q \neq q'$ the total number of such pairs of overlapping representations is

$$\ll \frac{X \log_3 X}{qq' \log X}\,.$$

Here we used the fact that, thanks to the equation $q + a = q' + a'$, the choice of q, q', a uniquely determines a'.

Summing up over q, q', we get that the number of such possibilities is

$$\begin{aligned} &\ll \quad \frac{X \log_3 X}{\log X} \left(\sum_{q \in A(X)} \frac{1}{q} \right)^2 \\ &\ll \quad \frac{X (\log_3 X)^3}{\log X} \quad \text{(by Proposition 6)} \\ &= \quad o(X)\,. \end{aligned}$$

This concludes the proof of the claim.

6 Conclusion

We have defined a set $\mathcal{S} \subseteq \mathbb{N}$ of positive lower density relative to which the Skolem Problem for simple LRS is decidable. In an extended version of this paper we will show that we can solve the Skolem Problem relative to (a slight variant of) $\mathcal{S}$ for all LRS, not just the simple ones. For this we use methodology of Amoroso and Viada [2] to give effective upper bounds on the number of solutions of a class of bivariate polynomial-exponential Diophantine equations, generalising the analysis in Section 4. We are also continuing to study the lower density of $\mathcal{S}$. In particular, a future work will show that the set has lower density one, under certain heuristic assumptions on the distributions of primes, similar to the Cramér heuristic (as used to justify Cramér's conjecture on prime gaps).

References

1 S. Akshay, N. Balaji, A. Murhekar, R. Varma, and N. Vyas. Near-optimal complexity bounds for fragments of the Skolem problem. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 154 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

2 F. Amoroso and E. Viada. Small points on subvarieties of a torus. *Duke Mathematical Journal*, 150(3), 2009.

3 A. Baker. *A Comprehensive Course in Number Theory*. Cambridge University Press, 2012.

4 D. Beauquier, A. M. Rabinovich, and A. Slissenko. A logic of probability with decidable model checking. *J. Log. Comput.*, 16(4), 2006.

5 J. Berstel and C. Reutenauer. *Noncommutative Rational Series with Applications*. Cambridge University Press, 2010.

6 V. Blondel and J. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000. doi:10.1016/S0005-1098(00)00050-9.

7 J.-Y. Cai, R. J. Lipton, and Y. Zalcstein. The complexity of the A B C problem. *SIAM J. Comput.*, 29(6), 2000.

8 G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward. *Recurrence Sequences*. American Mathematical Society, 2003.

9 J.-H. Evertse. On sums of s-units and linear recurrences. *Compositio Mathematica*, 53(2):225–244, 1984.

10 N. Fijalkow, J. Ouaknine, A. Pouly, J. Sousa Pinto, and J. Worrell. On the decidability of reachability in linear time-invariant systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC*, pages 77–86. ACM, 2019.

11 H. Halberstam and H.-E. Richert. *Sieve methods.* LMS Monographs. Academic Press, 1974.

12 G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers.* Oxford, at the Clarendon Press, 1954. 3rd ed.

13 R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *JACM*, 33(4), 1986.

14 C. Lech. A note on recurring series. *Ark. Mat.*, 2, 1953.

15 F. Luca, J. Ouaknine, and J. Worrell. Universal Skolem Sets. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–6. IEEE, 2021.

16 K. Mahler. Eine arithmetische Eigenschaft der Taylor Koeffizienten rationaler Funktionen. *Proc. Akad. Wet. Amsterdam*, 38, 1935.

17 D. W. Masser. Linear relations on algebraic groups. In *New Advances in Transcendence Theory.* Cambridge University Press, 1988.

18 M. Mignotte, T. Shorey, and R. Tijdeman. The distance between terms of an algebraic recurrence sequence. *J. für die reine und angewandte Math.*, 349, 1984.

19 J. Ouaknine and J. Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, 2015.

20 J. Piribauer and C. Baier. On Skolem-hardness and saturation points in markov decision processes. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 138:1–138:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

21 G. Rozenberg and A. Salomaa. *Cornerstones of Undecidability.* Prentice Hall, 1994.

22 H.P. Schlickewei and W.P. Schmidt. The number of solutions of polynomial-exponential equations. *Compositio Mathematica*, 120:193–225, January 2000.

23 T. Skolem. Ein Verfahren zur Behandlung gewisser exponentialer Gleichungen. In *Comptes rendus du congrès des mathématiciens scandinaves*, 1934.

24 T. Tao. *Structure and randomness: pages from year one of a mathematical blog.* American Mathematical Society, 2008.

25 A. J. Van Der Poorten and H. P. Schlickewei. Additive relations in fields. *Journal of the Australian Mathematical Society. Series A. Pure Mathematics and Statistics*, 51(1):154–170, 1991.

26 N. K. Vereshchagin. The problem of appearance of a zero in a linear recurrence sequence (in Russian). *Mat. Zametki*, 38(2), 1985.

Learning Deterministic Visibly Pushdown Automata Under Accessible Stack

Jakub Michaliszyn
University of Wrocław, Poland

Jan Otop
University of Wrocław, Poland

Abstract

We study the problem of active learning deterministic visibly pushdown automata. We show that in the classical L^*-setting, efficient active learning algorithms are not possible. To overcome this difficulty, we propose the accessible stack setting, where the algorithm has the read and write access to the stack. In this setting, we show that active learning can be done in polynomial time in the size of the target automaton and the counterexamples provided by the teacher. As counterexamples of exponential size are inevitable, we consider an algorithm working with words in a compressed representation via (visibly) Straight-Line Programs. Employing compression allows us to obtain an algorithm where the teacher and the learner work in time polynomial in the size of the target automaton alone.

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

Keywords and phrases visibly pushdown automata, automata inference, minimization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.74

Funding This work was supported by the National Science Centre (NCN), Poland under grant 2020/39/B/ST6/00521.

1 Introduction

Visibly pushdown automata (VPA), also known as input-driven pushdown automata, are a subclass of pushdown automata, in which the type of stack operation is determined by the input letter [33, 4]. This implies that the stack height, but not its content, is determined by the input word. This enables the construction of the *product* VPA from two given VPA. In consequence, the class of visibly pushdown languages recognized by VPA is closed under intersection, union and complement, which makes the universality problem decidable.

Even though VPA can be determinized, *deterministic* VPA (DVPA) enjoy better complexity properties: DVPA recognizing the union, the intersection or the complement of other DVPAs can be computed in polynomial time. In consequence, inclusion, equivalence and universality for DVPA can be decided in polynomial time. In contrast, the universality problem for VPA is **ExpTime**-complete, which implies that the exponential blow-up in the complementation (resp., determinization) of VPA is unavoidable. These properties render DVPA attractive for verification [16, 2, 28, 15], XML processing [31], approximating recurrent neural networks [12] and others [22].

For successful applications of DVPA, elimination of redundancy is essential. Furthermore, beyond minimization of a given DVPA [17], the automaton can be given implicitly, or before the minimization it can be too big to be constructed. The problem of minimization without explicit construction of the input DVPA can be addressed with active learning.

Active learning of automata has been originally proposed for deterministic finite-state automata (DFA) [6] with a *minimally adequate teacher*, which is an oracle for a hidden regular language L. *The teacher* answers *membership queries* whether a given word belongs to L, and *equivalence queries* whether a proposed automaton recognizes L, and if not it

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 74; pp. 74:1–74:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

returns a counterexample. There is a polynomial-time algorithm, called the L* algorithm, which only asks membership and equivalence queries and returns the minimal DFA $\mathcal{A}_L$ recognizing L. Therefore, to construct the minimal DFA $\mathcal{A}_L$, we only need to be able to answer the above queries; the explicit construction of the input automaton can be avoided.

While the L* algorithm is versatile and has been generalized to various types of automata, extensions of the L* algorithm to pushdown languages are elusive [7]. The main problems are the lack of the counterpart of the Myhill-Nerode theorem for pushdown languages (despite some attempts [13]) and the complexity of equivalence checking for DPDA, which is decidable but currently the only upper bound is primitive recursive [41, 24]. However, there is a counterpart of Myhill-Nerode theorem for visibly pushdown languages [3] and equivalence checking for DVPA is decidable in polynomial time. Thus, we focus our research on DVPA.

Still, active learning visibly pushdown languages is elusive. First, the canonical DVPA defined based on the right congruence relation from [3] is not a minimal-size DVPA recognizing a given language; it can be even exponentially larger than a minimal equivalent DVPA [3]. Second, a minimal-size DVPA recognizing a given visibly pushdown language is not unique, which indicates difficulty in improving the definition of the canonical DVPA. Finally, the minimization problem for DVPA is **NP**-complete, which rules out possible generalizations of the L* algorithm, which would return any minimal-size DVPA (unless **P**=**NP**). For these reasons, we adapt the active learning framework to learn DVPA rather than their languages, i.e., we consider queries, which depend on the automaton structure.

1.1 Our framework

In our framework, we consider a hidden DVPA, which we call the *target automaton*, and the teacher answering queries about it. The learning algorithm is expected to return a DVPA equivalent to the target DVPA. We assume that we can observe the stack content. To model this, we introduce a third type of queries, called *stack content queries*, which return the stack content of the hidden automaton upon reading a given input.

This alone is insufficient to circumvent the hardness inherited from minimization of DVPA (see Proposition 1). Thus, we additionally assume that we can change the stack content during the computation. To model writing, we employ *control words* consisting of standard letters and special *control letters*. A control letter is of the form $\underline{u}$, where u is the stack content. Upon reading $\underline{u}$, the automaton swaps its stack content to u, while retaining the current state. Control words can be sent by the algorithm as parameters for appropriate queries. The ability to swap the stack content to the one provided in the word makes it possible to reveal behavior of the target automaton under certain stack contents.

We first discuss the framework, in which there may be an arbitrary number of control letters altering the stack content. Such an approach directly reduces active learning of DVPA to active learning of regular languages. This technique can be employed for minimization of the product of DVPA while avoiding the need of the construction of the whole product in the first place (Section 5.1). The drawback is that the returned automata may contain states unreachable with any path that obeys stack operations (i.e., the usual runs in DVPA).

To avoid unreachable states, we propose another framework, in which all words contain at most one control letter (Section 6). In this setting, active learning can be done in polynomial time in the size of the target automaton and the counterexamples provided by the teacher. Unfortunately, the length of counterexamples may be exponential, making the whole learning process intractable. We consider two solutions: bounding the length of considered counterexamples (Section 7) and employing grammar-based compression (Section 8). This leads us to a scenario where the teacher and the learning algorithm work in time polynomial in the size of the target automaton alone, learning a DVPA equivalent to the target one.

1.2 Related work

Learning context-free languages has been intensively studied. Typically, the context-free grammars (CFGs) have been considered as the representation of a language rather than pushdown automata. On the negative side, it has been shown that learning a CFG from membership and equivalence queries is intractable under reasonable cryptographic assumptions [7]. To overcome this difficulty, as well as to learn grammars more humanly understandable, learning structurally equivalent grammars has been considered [40, 12]. Two grammars are strongly equivalent [37] if their sets of derivation trees with erased names of non-terminals are the same. The problem with that approach is the complexity of checking strong equivalence, which is in **ExpTime**, but **PSpace**-hard [39]. Furthermore, some grammar-based learning algorithms [40] are based on the L^* algorithm for regular tree languages. Basically, for a CFG G, one can learn a tree language of derivations in G of all words from $L(G)$. However, the minimal deterministic bottom-up tree automaton recognizing all derivation trees in G can be exponentially larger than G. This extends to visibly pushdown languages and such a tree automaton can be exponentially larger than a DVPA recognizing the visibly pushdown language.

Besides [40], there is a large body of work on learning CFGs [26, 43, 5, 42, 19]. Among them, [5] shares some similarities to our approach. However, the framework from [5] can be considered as learning the transitions of a pushdown automaton with known states, while in our approach we learn the set of states and the transitions.

While CFGs fit well into the active learning setting, pushdown automata are better suited and more popular in verification [1]. The above-mentioned algorithms cannot be used to learn automata. Active learning of automata is an active research topic. Initially, active learning has been developed for DFA in the seminal paper by Angluin [6]. Recently, there has been a renewed interest in learning various types of automata [11, 32, 9, 8, 10, 36, 34]. Learning based on syntactic properties of the target automaton has been recently proposed in [35], where a syntactic property called the *loop index* has been employed to overcome the difficulty of learning automata on infinite words.

Learning DVPA is studied in [23]. The algorithm provided there learns the canonical DVPA defined by the congruence relation introduced in [3]. It uses only membership and equivalence queries and works in polynomial time in the size of the canonical automaton. However, the size of the canonical DVPA can be exponential in the size of a minimal DVPA [3]. Therefore, the time and space complexity of the algorithm presented in [23] is exponential.

Another approach to learning VPA from [27] considers VPA with significant stack limitations so that every VPA has a unique minimal equivalent DVPA, computable in polynomial time. For this restricted class, polynomial-time learning using only membership and equivalence queries is possible.

A comprehensive survey of related work has been presented in [23, Chapter 7].

2 Preliminaries

Given a finite alphabet Σ of letters, a *word* w is a finite sequence of letters. For a word w, we define $w[i]$ as the i-th letter of w and $|w|$ as its length. We denote the set of all words over Σ by Σ^*. We use ϵ to denote the empty word.

The *index* of an equivalence relation is the number of its equivalence classes. An equivalence relation $\equiv$ on words Σ^* is a *right congruence* relation if and only if for all words $w_1, w_2 \in \Sigma^*$ and all $a \in \Sigma$, if $w_1 \equiv w_2$, then $w_1 a \equiv w_2 a$.

Real-time Pushdown automata. A *(non-deterministic) real-time pushdown automaton* (rt-PDA) is a tuple consisting of: (1) Σ, an input alphabet, (2) Γ, a finite stack alphabet, (3) Q, a finite set of states, (4) $Q_0 \subseteq Q$, a set of initial states, (5) $\delta \subseteq Q \times \Sigma \times (\Gamma \cup \{\bot\}) \times Q \times \Gamma^*$, a finite transition relation, and (6) $F \subseteq Q$, a set of accepting states. We assume that $\bot$ is never popped from the stack, i.e., for every $(q, a, \bot, q', u) \in \delta$, the stack update u starts with $\bot$. The size of an rt-PDA $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined as the number of states plus the number of transitions, i.e., $|Q| + |\delta|$. An rt-PDA is *deterministic* if $|Q_0| = 1$ and δ is a function from $Q \times \Sigma \times (\Gamma \cup \{\bot\})$ to $Q \times \Gamma^*$. Deterministic rt-PDA are denoted as rt-DPDA.

Semantics of rt-PDA. Consider an rt-PDA $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, \delta, F)$. A *configuration* of $\mathcal{A}$ is a pair consisting of the current state $q \in Q$ of $\mathcal{A}$ and its current stack content $u \in (\bot \cdot \Gamma^*)$. An rt-PDA defines an infinite-state transition system over its configurations $Q \times (\bot \cdot \Gamma^*)$, where configurations $Q_0 \times \{\bot\}$ are initial. For $a \in \Sigma$, let $\rightarrow^a_{\mathcal{A}}$ be an auxiliary relation defined such that $(q, uB) \rightarrow^a_{\mathcal{A}} (q', uv)$ if $(q, a, B, q', v) \in \delta$, where q, q' are states, uB, uv are stack contents with B being a single (top) symbol. For a word $w = a_1 \dots a_n$, we define $\rightarrow^w_{\mathcal{A}}$ as the composition of relation $\rightarrow^{a_1}_{\mathcal{A}} \dots \rightarrow^{a_n}_{\mathcal{A}}$, i.e., $(q, u_0) \rightarrow^w_{\mathcal{A}} (q', u_n)$ if there exist intermediate configurations $(q_1, u_1), \dots, (q_{n-1}, u_{n-1})$ such that for each $1 \leq i \leq n$ we have $(q_{i-1}, u_{i-1}) \rightarrow^{a_i}_{\mathcal{A}} (q_i, u_i)$.

A word w is *accepted* by $\mathcal{A}$ if there are $q_0 \in Q_0$, $q \in F$ and $u \in \bot \cdot \Gamma^*$ such that $(q_0, \bot) \rightarrow^w_{\mathcal{A}} (q, u)$. The *language recognized by* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. For an rt-DPDA $\mathcal{A}$, we define the stack content upon reading w, denoted by $SC_{\mathcal{A}}(w)$, as the unique u such that $(q_0, \bot) \rightarrow^w_{\mathcal{A}} (q', u)$ for some q'.

Visibly pushdown automata. Consider a partition of the alphabet Σ into three sets $(\Sigma_c, \Sigma_l, \Sigma_r)$ called, respectively, the sets of *call*, *local* and *return* letters. We say that a rt-PDA $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, \delta, F)$ is a *visibly pushdown automaton* (VPA) (with respect to the partition $(\Sigma_c, \Sigma_l, \Sigma_r)$) if the following conditions hold:

- transitions over letters from Σ_c (*call transitions*) push a single symbol on the stack and do not depend on the stack content, i.e., if $(q, a, B, q', v) \in \delta$, then $v = BG$, where $G \in \Gamma$ and for all $B' \in \Gamma \cup \{\bot\}$ we have $(q, a, B', q', B'G) \in \delta$,
- transitions over letters from Σ_l (*local transitions*) neither change nor depend on the stack content, i.e., if $(q, a, B, q', v) \in \delta$, then $v = B$ and for all $B' \in \Gamma \cup \{\bot\}$ we have $(q, a, B', q', B') \in \delta$,
- transitions over letters from Σ_r (*return transitions*) pop a single symbol from a nonempty stack; if the stack is empty they leave $\bot$ on the stack.

Since VPA are a subclass of rt-PDA, the definitions of size, determinacy, and semantics carry over from rt-PDA to VPA. Since DVPA are a subclass of rt-DPDA, the stack content upon reading w, $SC_{\mathcal{A}}(w)$, is well defined.

3 The L* algorithm

We briefly describe a variant of the L* algorithm for active learning of a hidden regular language L [6], which is similar to the algorithms we propose later on. To learn the language L, the algorithm asks the following queries to an oracle called *the teacher*:

- *membership queries*: given $w \in \Sigma^*$, is $w \in L$?, and
- *equivalence queries*: given a DFA $\mathcal{A}$, is $\mathcal{L}(\mathcal{A}) = L$? If not, return a counterexample, which is a word distinguishing $\mathcal{A}$ and L (i.e., from the symmetric difference of $\mathcal{L}(\mathcal{A})$ and L).

The L* algorithm constructs the minimal DFA $\mathcal{A}_L$ recognizing L by refining approximations of the right congruence relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ defined such that $u \sim_L v$ if and only if $\forall w(uw \in L \iff vw \in L)$. Due to the Myhill–Nerode theorem, the right congruence relation $\sim_L$ defines $\mathcal{A}_L$.

The algorithm maintains two sets of words: the set of *selectors* S containing ϵ, and the set of *test words* C. The set C defines an approximation $\sim_L^C$ of the right congruence relation defined such that $u \sim_L^C v$ if and only if $\forall w \in C(uw \in L \iff vw \in L)$. The set S contains exactly one selector of each equivalence class of $\sim_L^C$. More precisely, the algorithm maintains two properties: *separability*: different words from S are not $\sim_L^C$-equivalent, and *closedness*: for every $s \in S$ and $a \in \Sigma$, there is some word $s' \in S$ such that $sa \sim_L^C s'$. Note that if the pair S, C satisfies the above properties, we can construct a DFA corresponding to S, C, i.e., a DFA whose states are S and the transition relation is defined based on $\sim_L^C$, i.e., the successor of $s \in S$ over a is $s' \in S'$ such that $sa \sim_L^C s'$.

The algorithm starts with $S = \{\epsilon\}$ and $C = \emptyset$ and it iterates the following steps in the loop until it terminates in Step 2.

Step 1. Starting with separable S, C, close S, C, i.e., construct $S' \supseteq S$ and $C' \supseteq C$ that are closed and remain separable.

Step 2. Construct the DFA A corresponding to the separable and closed pair S', C'. Check whether A recognizes L; if yes, return A (and terminate).

Step 3. Otherwise, take the counterexample w and find $s \in S$, $a \in \Sigma$ and a suffix w' of w such that $S \cup \{sa\}, C \cup \{w'\}$ is separable, i.e., sa is a new selector for $\sim_L^{C \cup \{w'\}}$.

The L* algorithm returns the minimal DFA recognizing $\mathcal{A}_L$ and it works in polynomial time in $|\mathcal{A}_L|$ and the total length of returned counterexamples.

3.1 Difficulty of adapting L* to visibly pushdown languages

We discuss the difficulty of learning visibly pushdown languages, which motivates our learning framework presented in the following sections.

The key property used by the L*-algorithm is that for every regular language there is the canonical DFA recognizing it, which is also the minimal-size DFA for that language. Visibly pushdown languages do not have this property [3]. First, minimal-size DVPA are not unique; for a given visibly pushdown language $\mathcal{L}$ there can be multiple non-isomorphic minimal-size DVPA recognizing $\mathcal{L}$ [3, Proposition 1]. Second, a notion of canonical automaton for visibly pushdown languages was proposed in [3], but the canonical automaton can be exponentially larger than a minimal-size DVPA.

Furthermore, assuming that $\mathbf{P} \neq \mathbf{NP}$, there is no polynomial-time L*-type learning algorithm for visibly pushdown languages. L*-type algorithms learn a minimal-size automaton and run in polynomial time, and hence such an algorithm for DVPA can be used to minimize DVPA in polynomial time by running the learning algorithm as a black-box and computing answers to all its queries in polynomial time. The automaton returned by the algorithm would be a minimal-size DVPA language-equivalent to the input one $\mathcal{T}$. However, the minimization problem for DVPA (its decision version) is **NP**-complete [20]. For these reasons, we settle for learning automata rather than languages.

One may suspect that there is an L*-type learning algorithm, which only queries the stack content along the run on the input word. We show, however, that the **NP**-hardness proof from [20] can be adjusted to DVPA with singleton stack alphabets (i.e., where $|\Gamma| = 1$):

▶ **Proposition 1.** *The following problem is* $\mathbf{NP}$*-complete: given a DVPA $\mathcal{T}$ with a singleton stack alphabet (a deterministic visibly counter automaton) and $N > 0$, decide whether there is a DVPA with a singleton stack alphabet language-equivalent to $\mathcal{T}$ that has at most N states.*

For DVPA over a singleton stack alphabet, the stack height, which is the only information the stack provides, can be deduced from the input word. Therefore, allowing for stack observation alone does not lead to a polynomial-time learning algorithm.

4 Theoretical underpinnings

We introduce *control words*, which are words able to alter automata stack content. Then, we state and prove a Myhill-Nerode type theorem for DVPA using control words.

Control words. For a stack alphabet Γ, let $\underline{\Gamma}$ be defined as the infinite set $\{\underline{\bot \cdot u} \mid u \in \Gamma^*\}$, consisting of *control letters*. The intuitive meaning of the letter $\underline{\bot \cdot u}$ is "set the stack to $\bot \cdot u$". Given an alphabet Σ, a *control word* is a word over $\Sigma \cup \underline{\Gamma}$. The size of a control word w, denoted by $\mathrm{size}(w)$, is the sum of the number of letters from Σ and the sum of lengths of control letters from $\underline{\Gamma}$. We will sometimes use the term *standard word* to emphasize that the considered word does not contain control letters, i.e., it belongs to Σ^*.

Restricted control words. Let Σ be a finite alphabet and Γ be a finite stack alphabet. We define the set of *initializing words* $\mathrm{Init}^{\Sigma,\Gamma} = \underline{\Gamma} \cdot (\Sigma \cup \underline{\Gamma})^*$, the set of control words, in which the first letter is a control letter. Furthermore, we define *single-reset words* $\mathrm{Reset_1}^{\Sigma,\Gamma} = \Sigma^* \cdot \underline{\Gamma} \cdot \Sigma^*$ to be the set of control words with exactly one control letter, and the set of *single-reset initializing words* $\mathrm{Init_1}^{\Sigma,\Gamma} = \underline{\Gamma} \cdot \Sigma^*$, in which the first letter is a control letter and the remaining letters are standard. We will omit the superscript Σ, Γ for readability.

Automata processing control words. Let $\mathcal{T}$ be a DVPA over the alphabet Σ and the stack alphabet Γ. We extend the relation $\xrightarrow{w}_{\mathcal{T}}$ defined over words to control words as follows: let $v \in \bot \cdot \Gamma^*$, for every configuration (q, u) we have $(q, u) \xrightarrow{[\underline{v}]}_{\mathcal{T}} (q', u')$ if and only if $q = q'$ and $u' = v$, i.e., a control letter $\underline{v}$ resets the stack content of the automaton to v, while it retains the state. We straightforwardly generalize acceptance to control words. A control word w is *accepted* by $\mathcal{T}$ if there are $q_0 \in Q_0$, $q \in F$ and $u \in \bot \cdot \Gamma^*$ such that $(q_0, \bot) \xrightarrow{w}_{\mathcal{T}} (q, u)$. The language of control words accepted by $\mathcal{T}$ is denoted by $\underline{L}(\mathcal{T})$.

We define a counterpart of the right congruence of a regular language.

▶ **Definition 2.** *Let $C \subseteq Init$ be a set of initializing words. We define the relation $\equiv_{\mathcal{T}}^{C}$ on control words as follows: we have $w_1 \equiv_{\mathcal{T}}^{C} w_2$ if and only if (1) for each $v \in C$ we have $w_1 v \in \underline{L}(\mathcal{T}) \iff w_2 v \in \underline{L}(\mathcal{T})$ and (2) for each $v \in C$ we have $SC_{\mathcal{T}}(w_1 v) = SC_{\mathcal{T}}(w_2 v)$.*

For every C, the relation $\equiv_{\mathcal{T}}^{C}$ is an equivalence relation. The definition of $\equiv_{\mathcal{T}}^{C}$ is monotonic w.r.t. C, i.e., for all sets $C_1 \subseteq C_2$, the relation $\equiv_{\mathcal{T}}^{C_2}$ is a refinement of $\equiv_{\mathcal{T}}^{C_1}$. In particular, the index of $\equiv_{\mathcal{T}}^{C_1}$ does not exceed the index of $\equiv_{\mathcal{T}}^{C_2}$.

We require $C \subseteq \mathrm{Init}$, because otherwise the relation $\equiv_{\mathcal{T}}^{C}$ may have infinitely many equivalence classes, which correspond to configurations rather than states of $\mathcal{T}$.

For C being a strict subset of control words, $\equiv_{\mathcal{T}}^{C}$ need not be a right congruence; $w_1 \equiv_{\mathcal{T}}^{C} w_2$ does not imply $w_1 a \equiv_{\mathcal{T}}^{C} w_2 a$. Intuitively, words w_1, w_2 may lead to the same state in the automaton, but different configurations and after a single transition over letter a we obtain two configurations with different states. Thus, the right-congruence notion is incompatible with DVPA. We solve these problems by introducing another notion of congruence and restricting considered sets of control words.

Control right congruence. A relation $\simeq$ is a *control right congruence* if and only if for all words $w_1, w_2 \in (\Sigma \cup \underline{\Gamma})^*$ and $\alpha \in \underline{\Gamma} \cdot \Sigma$ we have $w_1 \simeq w_2$ implies $w_1\alpha \simeq w_2\alpha$.

▶ **Example 3.** Consider the *equal-state* relation $\simeq_{\mathcal{T}}$: for all $w_1, w_2 \in \Sigma^*$, we have $w_1 \simeq_{\mathcal{T}} w_2$ if and only if configurations reached by the DVPA $\mathcal{T}$ over w_1 and w_2 respectively, have the same state, i.e., there are a state q and stack contents u_1, u_2 such that $(q_0, \bot) \rightarrow^{w_1}_{\mathcal{T}} (q, u_1)$ and $(q_0, \bot) \rightarrow^{w_2}_{\mathcal{T}} (q, u_2)$. The relation $\simeq_{\mathcal{T}}$ need not be a right congruence relation as w_1 and w_2 can reach the same state with different symbols on the tops of stacks, and hence the states reached over w_1a and w_2a may differ, i.e., $w_1a \not\simeq_{\mathcal{T}} w_2a$. However, the relation $\simeq_{\mathcal{T}}$ is a control right congruence as the first control letter of $\alpha = ca$ resets the stack to the same value in w_1c and w_2c, and hence these words lead to the same configuration and the next transition over a leads to the same configuration. In consequence, $w_1ca \simeq_{\mathcal{T}} w_2ca$.

We show that $\equiv^{\text{Init}}_{\mathcal{T}}$ is a control right congruence with the index bounded by the number of states of $\mathcal{T}$. Therefore, there is a finite C such that $\equiv^{C}_{\mathcal{T}}$ and $\equiv^{\text{Init}}_{\mathcal{T}}$ coincide. We also show that $\equiv^{\text{Init}}_{\mathcal{T}}$ allows us to construct the transition relation of a DVPA corresponding to $\equiv^{\text{Init}}_{\mathcal{T}}$.

▶ **Lemma 4.** *Let $\mathcal{T}$ be a DVPA. (1) The relation $\equiv^{Init}_{\mathcal{T}}$ is a control right congruence, (2) the index of $\equiv^{Init}_{\mathcal{T}}$ is bounded by $|\mathcal{T}|$, and (3) the relation $\equiv^{Init}_{\mathcal{T}}$ defines a unique DVPA $\mathcal{A}$, which is language-equivalent to $\mathcal{T}$.*

▶ **Example 5.** We discuss why the second condition of the definition of $\equiv^{C}_{\mathcal{T}}$ is needed. Consider $\Sigma = \Sigma_c \cup \Sigma_l \cup \Sigma_r$ and a DVPA $\mathcal{A} = (\Sigma, \Gamma, Q, q_0, \delta, F)$. We assume that the sets Q, Σ, Γ and $\{\#, \$\}$ are pairwise disjoint. Let $\Sigma'_c = \Sigma_c \cup Q$, $\Sigma'_r = \Sigma_r \cup \{\$\}$, $\Sigma'_l = \Sigma_l \cup \{\#\}$ and $\Sigma' = \Sigma'_c \cup \Sigma'_r \cup \Sigma'_l$.

We consider a language $\mathcal{L}$ over the alphabet Σ' that consists of the words of the form $w_q\#w\#w_\$$, where $w_q \in Q^*$ and $w_\$ \in \{\$\}^*$. The idea is as follows: we want to accept words that make the stack empty (hence we use \$ at the end, to remove remaining stack elements) such that $\mathcal{A}$ after reading w is in one of the states listed in w_q.

The language $\mathcal{L}$ can be recognized by the automaton $\mathcal{A}'$ that works in three stages. First, it pushes all the symbols from w_q on the stack. Then, it emulates $\mathcal{A}$ on w. Finally, it pops stack symbols and checks whether the state of $\mathcal{A}$ after reading w is on the stack.

We argue that the automaton $\mathcal{A}'$ has the following property: its behaviour on the w_q part has no impact on its behaviour on the w part, but is crucial for the acceptance. Thus, the symbols that are put on the stack in the w_q part are significant only in processing the $w_\$$ part. This shows *non-locality*: the behaviour of $\mathcal{A}'$ on $w_\$$ is defined by w_q, which are separated by an arbitrary long word w.

Such non-locality is what often makes the learning problems intractable. To avoid this, we allow the learning algorithm to read the stack content, and we use the definition of the relation $\equiv^{C}_{\mathcal{T}}$. There, we require that for each $a \in \Sigma$ and for each $c \in \{\underline{\bot\gamma} \mid \gamma \in \Gamma \cup \{\epsilon\}\}$ we have $SC_{\mathcal{T}}(w_1ca) = SC_{\mathcal{T}}(w_2ca)$. In this way, the parts of the automaton reading w_q and $w_\$$ can be learned independently, as the results of reading letters on the stack are immediate.

If control letters are applied after each transition, DVPA behaves essentially as a DFA (see Section 5) and hence we attempt to minimize the use of control letters. In particular, we argue that considering $C = \text{Init}_1$ instead of Init is sufficient, i.e., single-reset initializing words are sufficient. We formalize it below.

We say that a DVPA $\mathcal{A}$ is *constructed from* the equivalence relation $\equiv^{\text{Init}_1}_{\mathcal{T}}$ if its states are selectors of the equivalence classes of $\equiv^{\text{Init}_1}_{\mathcal{T}}$ and if $\mathcal{A}$ has a transition $\delta(s, a, \alpha, s', \gamma)$, then either $s\underline{\bot\alpha}a \equiv^{\text{Init}_1}_{\mathcal{T}} s'$, or $\alpha = \bot$ and a is a return letter, and $s\underline{\bot\alpha}a$ and s' are equivalent over words $\underline{\bot}w'$ only. The stack content after reading $s\underline{\bot\alpha}a$ is updated according to γ. Such an automaton is not necessarily unique as $\equiv^{\text{Init}_1}_{\mathcal{T}}$ is not a control right congruence in general. However, we show that any such DVPA is admissible:

Table 1 Queries for learning with control words.

Query	membership query	stack-content query	equivalence query
Input	a control word w	a control word w	a DVPA $\mathcal{A}$
Output	whether $\mathcal{T}$ accepts w	$SC_{\mathcal{T}}(w)$	YES if $\mathcal{A}$ and $\mathcal{T}$ are language-equivalent, otherwise a control word w distinguishing $\mathcal{A}$ and $\mathcal{T}$

▶ **Lemma 6.** *Any DVPA constructed from $\equiv_{\mathcal{T}}^{Init_1}$ restricted to standard words is language-equivalent with $\mathcal{T}$.*

All equivalence classes of $\equiv_{\mathcal{T}}^{\mathrm{Init}}$, which do not contain standard words are irrelevant for language-equivalence as they correspond to unreachable states. Thus, Lemma 6 implies that in order to learn a DVPA language-equivalent to $\mathcal{T}$, we can construct $\equiv_{\mathcal{T}}^{\mathrm{Init}_1}$ restricted to standard words. This observation is the backbone of the learning algorithm presented in Section 6.

5 Learning with resets

We present the first active learning algorithm for DVPA based on control words and discuss its applications. We consider a learning framework with three types of queries: (modified) membership and equivalence queries and a new type called *stack-content queries*, which return the content of the stack upon reading a given word. The queries involve control letters, as we showed that using only standard words is insufficient (Section 3.1). Here, control words can have any number of control letters. In such a case, learning DVPA boils down to the standard learning of DFA. Indeed, with arbitrarily many control letters every path in a DVPA can be realized. Furthermore, we can address each transition and learn how the stack changes with stack-content queries. The solution for learning with unlimited control letters is a simple adaptation of the original L* algorithm.

We consider an oracle, called the *teacher*, which has access to the target DVPA $\mathcal{T}$ and answers the following three types of queries detailed in Table 1. Note that the counterexample can contain control letters as well.

We reduce active learning of DVPA to active learning of regular languages. Consider $\Sigma^{\mathrm{lab}} = (\Sigma_c \times \Gamma) \cup \Sigma_l \cup (\Sigma_r \times (\Gamma \cup \{\bot\}))$, which characterizes labels of transitions of $\mathcal{T}$: (1) call transitions are characterized by a call letter $a \in \Sigma_c$ and a stack symbol $B \in \Gamma$ pushed to the stack, (2) local transitions are characterized by a local letter alone $a \in \Sigma_l$, and (3) return transitions are characterized by a call letter $a \in \Sigma_r$ and the top of the stack symbol $\gamma \in \Gamma \cup \{\bot\}$.

Let $\mathcal{L}^{\mathrm{lab}}(\mathcal{T})$ be the language of words w over Σ^{lab}, which correspond to paths in $\mathcal{T}$ terminating in an accepting state. Note that paths need not obey the stack; for instance pushing a symbol A can be followed by pulling a symbol B, and hence not all paths correspond to runs of $\mathcal{T}$. However, the language $\mathcal{L}^{\mathrm{lab}}(\mathcal{T})$ is regular and hence can be learned with the L*-algorithm for DFA. The detailed constructions have been relegated to the appendix.

▶ **Theorem 7.** *Assume that the teacher returns minimal-size counterexamples. Then, active learning of DVPA with a teacher answering queries detailed in Table 1 can be done in time polynomial in the size of the target automaton.*

Table 2 Queries for learning with single resets.

Query	membership query	stack-content query	equivalence query
Input	a single-reset word w	a single-reset word w	a DVPA $\mathcal{A}$
Output	whether $\mathcal{T}$ accepts w	$SC_{\mathcal{T}}(w)$	YES if $\mathcal{A}$ and $\mathcal{T}$ are language-equivalent, otherwise, a (standard) word w distinguishing $\mathcal{A}$ and $\mathcal{T}$

5.1 Applications

The active learning of DVPA can be used for reducing the number of states of DVPA given implicitly. For instance, consider a model M consisting of k components $\mathcal{A}_1, \ldots, \mathcal{A}_k$ being DVPA. Models that are products of smaller components occur frequently in verification tasks (e.g., to express union or intersection of languages) and there is a large body of work on efficient model checking on such models [38, 21]. Even though the size of M is the product of sizes of $\mathcal{A}_1, \ldots, \mathcal{A}_k$, the number of reachable states may be considerably smaller, so it can be explicitly represented.

We can use the above active learning algorithm for DVPA to construct a reduced model of $\mathcal{A}_1 \times \ldots \times \mathcal{A}_k$ without the need of constructing the whole product. Now having DVPA $\mathcal{A}_1, \ldots, \mathcal{A}_k$, we can compute the values of membership and stack-content queries. For the equivalence queries with a given $\mathcal{A}$, we can check whether for all i we have $\mathcal{L}(\mathcal{A}_i) \supseteq \mathcal{L}(\mathcal{A})$, which implies that the language of the product of automata contains the language of $\mathcal{A}$. The opposite inclusion can be approximated with queries over random words, guided by various heuristics as is often the case with equivalence queries [12, 18]. It is difficult to answer equivalence queries for the product of automata as it subsumes the emptiness problem for intersections of regular languages, which is **PSpace**-complete.

Note that the constructed automaton can have states that are not reachable by any run (see Example 8). This is an inherent problem of considered words with multiple control letters; it diminishes the influence of the stack on reachability. To solve this problem, we focus on the single-reset approach presented in the following Section 6.

▶ **Example 8** (Automaton with redundant states)**.** Consider a DVPA $\mathcal{T}$ and its state q, which is reachable only with call transitions pushing A on the stack. Suppose that q has an outgoing return transition over $A' \neq A$, which leads to a component $\mathcal{B}$ of $\mathcal{T}$ not reachable with any other transition. Then, the algorithm from Theorem 7 explores the component $\mathcal{B}$ of $\mathcal{T}$ as it is reachable over words with multiple control letters and hence the returned DVPA contains redundant states.

6 Learning with a single reset

Here we study a learning framework with three types of queries as in Section 5, but restricted to words with one control letter. Those queries are detailed in Table 2. For a finite $C \subseteq \mathrm{Init}_1$, these queries are sufficient to decide whether $w_1 \equiv^C_{\mathcal{T}} w_2$ holds over $w_1, w_2 \in \Sigma^*$; the first condition from the definition of $\equiv^C_{\mathcal{T}}$ (Definition 2) can be implemented with membership queries, and the second one with stack-content queries.

We present an active learning algorithm for DVPA, which works in polynomial time in size of the target automaton and the size of counterexamples provided by the teacher. The algorithm is a modification of the L* algorithm, so we discuss only the differences with the vanilla L* algorithm. The main difference is that while L* constructs the right congruence relation of the target language $\mathcal{L}$, our algorithm constructs $\equiv^{\mathrm{Init}_1}_{\mathcal{T}}$ over standard words.

Our algorithm, similarly to L*, maintains two sets of words: a set of *selectors* S and a set of *test words* C such that S are selectors of equivalence classes of $\equiv^{C}_{\mathcal{T}}$. Selectors in $S \subset \Sigma^*$ are standard words over Σ and test words in $C \subset \text{Init}_1$ are single-reset initializing words. We start with $S = \{\epsilon\}$ and $C = \{\underline{\bot}\epsilon\}$.

We adjust the terminology to the current setting as follows. *Separability* means that different words from S are not $\equiv^{C}_{\mathcal{T}}$-equivalent. *Closedness* means that for every $s \in S$, $a \in \Sigma$ and $B \in \Gamma \cup \{\epsilon\}$, there is some word $s' \in S$ such that $s \cdot \underline{\bot \cdot B} \cdot a \equiv^{C}_{\mathcal{T}} s'$. The algorithm maintains separability and some restricted form of closedness. The latter is due to the fact that some transitions may be unreachable in a DVPA. For example, if the only way to reach some state is to push B on the stack, then all the transitions assuming that the top of the stack is not B are unreachable and thus irrelevant. For that reason, two steps in the L* algorithm: computing closure of S, C and generating the corresponding automaton, are interleaved and encapsulated in a single procedure *generate automaton*.

Generating automata. Given S, C and access to the teacher for $\mathcal{T}$, we construct a (partial) DVPA whose set of states is $S \subset \Sigma^*$, possibly extended with some additional states as discussed below. The initial state is $\epsilon \in S$. The accepting states are those states $w \in S$ such that the membership query on w returns true, i.e., w is accepted by $\mathcal{T}$. It remains to compute the transition function δ.

Consider a state $s \in S$, a letter $a \in \Sigma$ and $\gamma \in \Gamma \cup \{\bot\}$. To find $\delta(s, a, \gamma)$, we feed s to the target automaton $\mathcal{T}$, implant γ on the top of its stack and take the transition over a. Next, we identify the state from S equivalent to the current state of $\mathcal{T}$ or create a new one, if there is no fitting one in S. The details are in the appendix.

Equivalence and processing a counterexample. Given a partial DVPA $\mathcal{A}$, we complete it to a DVPA $\mathcal{A}^c$ by setting all undefined transitions so that their destination is ϵ and putting some designated $B_0 \in \Gamma$ on the stack over call transitions. Next, we test for equivalence and if it happens to be language-equivalent to $\mathcal{T}$ we return $\mathcal{A}^c$. Otherwise, we get a counterexample that distinguishes $\mathcal{A}^c$ and $\mathcal{T}$. We show how to use it to ensure progress of the algorithm. There are two possible cases of counterexamples: either the stack content diverges at some point, or the values of some suffixes mismatch. In both cases, we can modify S and C to ensure the progress. The details are in the appendix.

Irredundancy. Observe that all states in the returned automaton are reachable by some run (in contrast to the automaton from Example 8). To see this, note that we start with the initial state ϵ. Next, new successors are added either in the automata construction or in processing of a counterexample. In the former case, the new state is a successor of some reachable state over a consistent transition. That is, it is a call or local transition, which can be always executed, or it is a return transition and the stack content over w_s reaching $s \in S$ in the constructed automaton is consistent with the expected stack content in the transition. In the latter case, the successor va is added because the values of suffixes mismatch. There va is not $\equiv^{C'}_{\mathcal{T}}$-equivalent to any state from S and hence upon reading va the automaton reaches the new state va. Therefore, the algorithm with single-reset words avoids computation of redundant states in contrast to the algorithm from Section 5.

Complexity. The algorithm always terminates and works in polynomial time in the size of the target automaton and the size of teachers' responses. The details are in the appendix.

▶ **Theorem 9.** *Active learning of DVPA with teacher answering queries detailed in Table 2 can be done in time polynomial in the sizes of the target automaton and teachers' responses.*

For L*, if we assume that the teacher always returns a minimal-length counterexample, the algorithm works in polynomial time in the size of the target automaton alone. For DVPA, the shortest counterexample for language-equivalence can be of exponential-size in $|\mathcal{T}|$. In the following sections we propose two approaches to solve this problem: the first one is based on relaxed equivalence and the second one on employing grammar-based compression.

7 Active learning with bounded counterexamples

To avoid processing exponentially-long counterexamples, we can impose a bound on the length of admissible counterexamples. In some scenarios we may be interested in trading accuracy for performance and focus on words of length bounded by some fixed value t. A similar approach led to the development of *bounded model checking* methods, which are highly efficient [14, 25]. Furthermore, in applications of active learning of automata, equivalence queries, which are often costly to implement, are only approximated by equivalence over bounded-length words [12, 18].

Two DVPA $\mathcal{A}_1, \mathcal{A}_2$ are t-equivalent, where $t \in \mathbb{N}$, if and only if their languages agree on all words of the length bounded by t. We adjust the framework in a natural way: we replace equivalence queries with *t-equivalence* queries, which say whether a given DVPA $\mathcal{A}$ is t-equivalent to $\mathcal{T}$. If it is not, then we require a counterexample of length bounded by t.

Under the modified framework, we can run the algorithm described in Section 6. Note that each counterexample there is bounded by t and the algorithm terminates when $\mathcal{A}$ is t-equivalent to $\mathcal{T}$. Furthermore, each counterexample can be used to increase the size of S, which is bounded by the number of states of $\mathcal{T}$. In consequence, we have:

▶ **Theorem 10.** *Active learning of DVPA with teacher answering membership and stack-content queries defined in Table 2 and t-equivalence queries can be done in time polynomial in the size of the target automaton and t.*

Finally, observe that the teacher can answer t-equivalence queries in polynomial time in $|\mathcal{A}| + |\mathcal{T}| + t$. It suffices to construct a DFA $\mathcal{A}_t$ accepting all words of the length at most t and compute the emptiness of the intersection of the languages $\mathcal{A}$, $\mathcal{A}_t$ and $\mathcal{T}^C$, where $\mathcal{T}^C$ is the complemented $\mathcal{T}$.

8 Active learning with compressed representation

Here we propose to solve the problem of exponential counterexamples using compression by means of visibly context-free grammars to represent counterexamples as well as selectors and test words. We show that in this setting the teacher can always provide a counterexample of polynomial size in the size of the target automaton and a candidate automaton sent in the equivalence query. If the teacher does so, the algorithm we propose works in polynomial time in size of the target automaton.

Context-free grammars (which are equivalent to pushdown automata) can generate finite languages with exponential-length words. This observation led to the development of grammar-based compression, where a word w is represented through a context-free grammar (called Straight-Line Programs or SLPs) generating exactly one word w. Equivalence of SLPs with other compression methods and algorithmic problems are discussed in [30].

Table 3 Queries for learning with vSLP.

Query	membership query	stack-content query	equivalence query
Input	a vSLP representing a single-reset word w	a vSLP representing a single-reset word w	a DVPA $\mathcal{A}$
Output	whether $\mathcal{T}$ accepts w	$SC_{\mathcal{T}}(w)$	YES if $\mathcal{A}$ and $\mathcal{T}$ are language-equivalent, otherwise, a vSLP representing a standard word w distinguishing $\mathcal{A}$ and $\mathcal{T}$

We use polynomial-size SLPs to represent counterexamples as well as selectors. Note that processing SLPs can be expensive in general. There is a visibly pushdown language $\mathcal{L}_h$ such that the membership problem for this language over words given by SLPs is **PSpace**-complete [29]. However, if we restrict SLPs to those that are represented by *visibly pushdown grammars* G (over the same alphabet partition as the visibly pushdown language $\mathcal{L}_h$) defined below, then the membership problem is equivalent to the emptiness problem for the intersection of $\mathcal{L}_h$ and $\mathcal{L}(G)$, and hence it can be decided in polynomial time.

Visibly pushdown grammars. A *visibly pushdown grammar* (VPG) is a syntactically restricted pushdown grammar (see [4] for details). Consider a partition of Σ into $(\Sigma_c, \Sigma_l, \Sigma_r)$. A VPG is a triple $G = (V, S, P)$, where $V = V_0 \cup V_1$ is a set of *non-terminal symbols* partitioned into V_0 and V_1, $I \in V$ is the *start symbol*, and P is the set of *productions* of the following form:

- $X \to \epsilon$, where $X \in V$
- $X \to aY$, where $X, Y \in V$, and if $X \in V_0$, then $a \in \Sigma_l$ and $Y \in V_0$
- $X \to aYbZ$, where $X \in V$, $Y \in V_0$, $a \in \Sigma_c$, $b \in \Sigma_r$, and if $X \in V_0$, then $Z \in V_0$.
- $X \to YZ$, where $X, Y \in V$, and if $X \in V_0$, then $Y, Z \in V_0$.

The last rule is not present in the original definition, but it can be safely added as VPL are closed under concatenation.

VPA and VPG are language-wise polynomially equivalent [4] (i.e., there is a polynomial-time procedure that, given a PDA, outputs a CFG of the same language and vice versa).

For a VPG G, we define derivation $\to_G$ as a relation on $(\Sigma \cup V)^* \times (\Sigma \cup V)^*$ as follows: $w \to_G w'$ if and only if $w = w_1 X w_2$, with $X \in V$, and $w' = w_1 u w_2$ for some $u \in (\Sigma \cup V)^*$ such that $X \to u$ is a production from G. We define $\to_G^*$ as the transitive closure of $\to_G$. The *language generated by* G, denoted by $\mathcal{L}(G) = \{w \in \Sigma^* \mid s \to_G^* w\}$ is the set of words that can be derived from I. VPA and VPG are language-wise polynomially equivalent [4] (i.e., there is a polynomial-time procedure that, given a PDA, outputs a CFG of the same language and vice versa). The size of a VPG $G = (V, I, P)$, denoted by $|G|$, is $|V| + |P|$.

A *visibly Straight Line Program* (vSLP) is a visibly pushdown grammar G, which generates a single word denoted by w_G.

Now, selectors in S and test words in C will be represented by vSLPs. We introduce the framework and assume that words are represented as vSLPs, as detailed in Table 3. Importantly, the teacher can always return a polynomial size vSLP representing a counterexample.

▶ **Lemma 11.** *Given two DVPA $\mathcal{A}_1, \mathcal{A}_2$ such that $\mathcal{L}(\mathcal{A}_1) \neq \mathcal{L}(\mathcal{A}_2)$, there exists a polynomial size vSLP G such that w_G belongs to the symmetric difference of $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$. Furthermore, this vSLP can be computed in polynomial time in $|\mathcal{A}_1| + |\mathcal{A}_2|$.*

The words represented by vSLPs can be efficiently processed. The procedure generating a DVPA from S, C requires efficient computation of concatenation of vSLPs and compressed membership and stack-content queries. Observe that for a polynomial-size vSLP G, the length of $SC_{\mathcal{T}}(w_G)$ is polynomial in $|\mathcal{T}|$ even though $|w_G|$ can be exponential and hence the answers to these queries have polynomial size. Processing counterexamples is more difficult as it involves finding the prefixes of the counterexample w_G with certain properties. Note that these prefixes can be identified with binary search and hence only polynomially many candidates need to be considered. It suffices to compute, for a given vSLP G, the prefix of the first i letters of w_G, which is possible in polynomial time in $|G|$. In consequence, we have the following:

▶ **Theorem 12.**

(i) *The teacher can answer membership and equivalence queries with vSLP input in polynomial time in the input size and $|\mathcal{T}|$.*

(ii) *The teacher can answer equivalence queries with polynomial size vSLP encoding a counterexample in polynomial time (in the input size and $|\mathcal{T}|$).*

(iii) *For the teacher as in* **(i)** *and* **(ii)***, active learning of DVPA with teacher answering queries detailed in Table 3 can be done in time polynomial in the size of the target automaton alone.*

9 Future work

We have presented the first work on polynomial-time learning DVPA. The queries we use depend not only on the language of the automaton, but its structure as well. It allows us to bypass problems arising from the difficulty of minimizing DVPA.

There is an open field of possible future directions, which we discuss below.

Dropping the stack content queries. Control words are crucial in our solution, i.e, the ability to alter the stack. We also assume that the stack can be observed, using the stack content queries. While this is a reasonable assumption, we have not shown whether it is necessary. In particular, we may explore the scenario from Section 5.1, in which we attempt to minimize both the state space and the stack alphabet and hence we consider the stack alphabet as unknown. We leave this as an open question.

Queries with registers. Another interesting scenario, but closely related to the previous one, is where we cannot read and write the stack directly, but we can save its value to a register and then use it. This corresponds to the case when we have access to the device memory, but the stack is represented in a way that we cannot manipulate it (e.g. it is encrypted). Note that in Section 6, we either use the stack contents that have been produced by some runs with at most one symbol pushed on the top. Is active learning in polynomial time possible if the stack content can be only copied between runs?

Learning all rt-DPDA. In the algorithm presented in Section 6, we rely on the fact that call and local letters do not depend on the stack content. We can possibly improve the algorithm to omit that assumption at the cost of more complex algorithm. We leave this problem for further investigation.

Learning all DPDA. Another challenging area is learning DPDA. One of the key challenges is to deal with ϵ-transitions. The problem there is that for some DPDA there can be states that are reachable with only ϵ-transitions, i.e., states without selectors. A solution to this problem would require a new insight into the representation of states.

References

1 Rajeev Alur, Ahmed Bouajjani, and Javier Esparza. Model checking procedural programs. In *Handbook of Model Checking*, pages 541–572. Springer, 2018.

2 Rajeev Alur, Kousha Etessami, and Parthasarathy Madhusudan. A temporal logic of nested calls and returns. In *TACAS*, pages 467–481. Springer, 2004.

3 Rajeev Alur, Viraj Kumar, Parthasarathy Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In *ICALP 2005*, pages 1102–1114. Springer, 2005.

4 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

5 Dana Angluin. Learning k-bounded context-free grammars. *Research report. Yale University. Dept. of Computer Science*, 557, 1987.

6 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

7 Dana Angluin and Michael Kharitonov. When won't membership queries help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.

8 Borja Balle and Mehryar Mohri. Learning weighted automata. In *CAI 2015*, pages 1–21, 2015.

9 Borja Balle and Mehryar Mohri. On the Rademacher complexity of weighted automata. In *ALT 2015*, pages 179–193, 2015.

10 Borja Balle and Mehryar Mohri. Generalization bounds for learning weighted automata. *Theor. Comput. Sci.*, 716:89–106, 2018.

11 Borja Balle, Prakash Panangaden, and Doina Precup. A canonical form for weighted automata and applications to approximate minimization. In *LICS 2015*, pages 701–712, 2015.

12 Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Igor Khmelnitsky, Martin Leucker, Daniel Neider, Rajarshi Roy, and Lina Ye. Extracting context-free grammars from recurrent neural networks using tree-automata learning and a* search. In Jane Chandlee, Rémi Eyraud, Jeff Heinz, Adam Jardine, and Menno Zaanen, editors, *Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event*, volume 153 of *Proceedings of Machine Learning Research*, pages 113–129. PMLR, 2021. URL: https://proceedings.mlr.press/v153/barbot21a.html.

13 Jean Berstel and Luc Boasson. Towards an algebraic theory of context-free languages. *Fundamenta Informaticae*, 25(3, 4):217–239, 1996.

14 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, 58:117–148, 2003. doi:10.1016/S0065-2458(03)58003-2.

15 Laura Bozzelli, Aniello Murano, and Adriano Peron. Context-free timed formalisms: Robust automata and linear temporal logics. *Information and Computation*, page 104673, 2020.

16 Swarat Chaudhuri and Rajeev Alur. Instrumenting c programs with nested word monitors. In *International SPIN Workshop on Model Checking of Software*, pages 279–283. Springer, 2007.

17 Patrick Chervet and Igor Walukiewicz. Minimizing variants of visibly pushdown automata. In *MFCS 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2007. doi:10.1007/978-3-540-74456-6_14.

18 Hana Chockler, Pascal Kesseli, Daniel Kroening, and Ofer Strichman. Learning the language of software errors. *J. Artif. Intell. Res.*, 67:881–903, 2020. doi:10.1613/jair.1.11798.

19 Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars.* Cambridge University Press, New York, NY, USA, 2010.

20 Olivier Gauwin, Anca Muscholl, and Michael Raskin. Minimization of visibly pushdown automata is np-complete. *Log. Methods Comput. Sci.*, 16(1), 2020.

21 Dimitra Giannakopoulou, Kedar S. Namjoshi, and Corina S. Pasareanu. Compositional reasoning. In *Handbook of Model Checking*, pages 345–383. Springer, 2018. doi:10.1007/978-3-319-10575-8_12.

22 David Hopkins, Andrzej S Murawski, and C-H Luke Ong. A fragment of ML decidable by visibly pushdown automata. In *ICALP*, pages 149–161. Springer, 2011.

23 Malte Isberner. *Foundations of Active Automata Learning: An Algorithmic Perspective*. PhD thesis, Technischen Universität Dortmund, 2015. URL: https://eldorado.tu-dortmund.de/bitstream/2003/34282/1/Dissertation.pdf.

24 Petr Jancar. Equivalence of pushdown automata via first-order grammars. *J. Comput. Syst. Sci.*, 115:86–112, 2021. doi:10.1016/j.jcss.2020.07.004.

25 Kareem Khazem and Michael Tautschnig. CBMC path: A symbolic execution retrofit of the C bounded model checker. In *TACAS*, volume 11429 of *LNCS*, pages 199–203, 2019. doi:10.1007/978-3-030-17502-3_13.

26 Bruce Knobe and Kathleen Knobe. A method for inferring context-free grammars. *Information and Control*, 31(2):129–146, 1976.

27 Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Minimization, learning, and conformance testing of boolean programs. In *CONCUR 2006*, pages 203–217. Springer Berlin Heidelberg, 2006.

28 Christof Löding, Parthasarathy Madhusudan, and Olivier Serre. Visibly pushdown games. In *FSTTCS*, pages 408–420. Springer, 2004.

29 Markus Lohrey. Leaf languages and string compression. *Inf. Comput.*, 209(6):951–965, 2011. doi:10.1016/j.ic.2011.01.009.

30 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups-Complexity-Cryptology*, 4(2):241–299, 2012.

31 Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *ACM Transactions on Database Systems (TODS)*, 31(3):770–813, 2006.

32 Ines Marusic and James Worrell. Complexity of equivalence and learning for multiplicity tree automata. *Journal of Machine Learning Research*, 16:2465–2500, 2015.

33 Kurt Mehlhorn. Pebbling mountain ranges and its application of DCFL-recognition. In *ICALP 1980*, volume 85 of *LNCS*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2_89.

34 Jakub Michaliszyn and Jan Otop. Approximate learning of limit-average automata. In *CONCUR 2019*, pages 17:1–17:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.17.

35 Jakub Michaliszyn and Jan Otop. Learning deterministic automata on infinite words. In *ECAI 2020 – 24th European Conference on Artificial Intelligence*, volume 325, pages 2370–2377. IOS Press, 2020. doi:10.3233/FAIA200367.

36 Jakub Michaliszyn and Jan Otop. Non-deterministic weighted automata evaluated over Markov chains. *J. Comput. Syst. Sci.*, 108:118–136, 2020.

37 Marvin C Paull and Stephen H Unger. Structural equivalence of context-free grammars. *J. Comput. Syst. Sci.*, 2(4):427–463, 1968.

38 Doron Peled. Partial-order reduction. In *Handbook of Model Checking*, pages 173–190. Springer, 2018. doi:10.1007/978-3-319-10575-8_6.

39 Daniel J. Rosenkrantz and Harry B. Hunt III. The complexity of structural containment and equivalence. In Jeffrey D. Ullman, editor, *Theoretical Studies in Computer Science, to Seymour Ginsburg on the occasion of his 2^6. birthday*, pages 101–132. Academic Press, 1992. doi:10.1016/b978-0-12-708240-0.50009-5.

40 Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, 1992.

41 Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *ICALP 1997*, pages 671–681, 1997. doi:10.1007/3-540-63165-8_221.

42 Luzi Sennhauser and Robert C. Berwick. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018*, pages 115–124, 2018. doi:10.18653/v1/w18-5414.

43 Ehud Y Shapiro. Algorithmic program diagnosis. In *POPL*, pages 299–308, 1982.

Cohomology in Constraint Satisfaction and Structure Isomorphism

Adam Ó Conghaile
Computer Laboratory, Cambridge University, United Kingdom
The Alan Turing Institute, United Kingdom

Abstract

Constraint satisfaction (CSP) and structure isomorphism (SI) are among the most well-studied computational problems in Computer Science. While neither problem is thought to be in PTIME, much work is done on PTIME approximations to both problems. Two such historically important approximations are the k-consistency algorithm for CSP and the k-Weisfeiler-Leman algorithm for SI, both of which are based on propagating local partial solutions. The limitations of these algorithms are well-known – k-consistency can solve precisely those CSPs of bounded width and k-Weisfeiler-Leman can only distinguish structures which differ on properties definable in C^k. In this paper, we introduce a novel sheaf-theoretic approach to CSP and SI and their approximations. We show that both problems can be viewed as deciding the existence of global sections of presheaves, $\mathcal{H}_k(A, B)$ and $\mathcal{I}_k(A, B)$ and that the success of the k-consistency and k-Weisfeiler-Leman algorithms correspond to the existence of certain efficiently computable subpresheaves of these. Furthermore, building on work of Abramsky and others in quantum foundations, we show how to use Čech cohomology in $\mathcal{H}_k(A, B)$ and $\mathcal{I}_k(A, B)$ to detect obstructions to the existence of the desired global sections and derive new efficient cohomological algorithms extending k-consistency and k-Weisfeiler-Leman. We show that cohomological k-consistency can solve systems of equations over all finite rings and that cohomological Weisfeiler-Leman can distinguish positive and negative instances of the Cai-Fürer-Immerman property over several important classes of structures.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases constraint satisfaction problems, finite model theory, descriptive complexity, rank logic, Weisfeiler-Leman algorithm, cohomology

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.75

Related Version *Full Version*: https://arxiv.org/abs/2206.15253

Funding Funded in part by EPSRC grants EP/R513180/1 and EP/S03238X/1 and supported by a PhD Enrichment Award from the Alan Turing Institute.

Acknowledgements I thank Anuj Dawar and Samson Abramsky for useful discussions in writing this paper and am especially gratefully to Samson Abramsky for permission to use results from [1].

1 Introduction

Constraint satisfaction problems (CSP) and structure isomorphism (SI) are two of the most well-studied problems in complexity theory. Mathematically speaking, an instance of one of these problems takes a pair of structures (A, B) as input and asks whether there is a homomorphism $A \to B$ for CSP or an isomorphism $A \cong B$ for SI. These problems are not in general thought to be tractable. Indeed the general case of CSP is NP-Complete and restricting our structures to graphs the best known algorithm for SI is Babai's quasi-polynomial time algorithm [8]. As a result, it is common in complexity and finite model theory to study approximations of the relations → and ≅.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 75; pp. 75:1–75:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The k-consistency and k-Weisfeiler-Leman[1] algorithms efficiently determine two such approximations to $\rightarrow$ and $\cong$ which we call $\rightarrow_k$ and $\equiv_k$. These relations have many characterisations in logic and finite model theory, for example in [18] and [13]. One that is particularly useful is that of the existence of winning strategies for Duplicator in certain Spoiler-Duplicator games with k pebbles [28, 25]. For both of these games Duplicator's winning strategies can be represented as non-empty sets $S \subset \mathbf{Hom}_k(A, B)$ of k-local partial homomorphisms which satisfy some extension properties and connections between these games have been studied before. For example, a joint comonadic semantics is given by the pebbling comonad of Abramsky, Dawar and Wang [4].

The limitations of these approximations are well-known. In particular, it is known that k-consistency only solves CSPs of *bounded width* and k-Weisfeiler-Leman can only distinguish structures which differ on properties expressible in the infinitary counting logic C^k. Feder and Vardi [18] showed that CSP encoding linear equations over the finite fields do not have bounded width, while Cai, Fürer, and Immerman [13] demonstrated an efficiently decidable graph property which is not expressible in C^k for any k.

In the present paper, we introduce a novel approach to the CSP and SI problems based on presheaves of k-local partial homomorphisms and isomorphisms, showing that the problems can be reframed as deciding whether certain presheaves admit global sections. We show that the classic k-consistency and k-Weisfeiler-Leman algorithms can be derived by computing greatest fixpoints of presheaf operators which remove some efficiently computable obstacles to global sections. Furthermore, we show how invariants from sheaf cohomology can be used to find further obstacles to combining local homomorphisms and isomorphisms into global ones. We use these to construct new efficient extensions to the k-consistency and k-Weisfeiler-Leman algorithms computing relations $\rightarrow_k^{\mathbb{Z}}$ and $\equiv_k^{\mathbb{Z}}$ which refine $\rightarrow_k$ and $\equiv_k$.

The application of presheaves has been particularly successful in computer science in recent decades with applications in semantics [32, 19], information theory [33] and quantum contextuality [3, 5, 2]. This work draws in particular on the application of sheaf theory to quantum contextuality, pioneered by Abramsky and Brandenburger [3] and developed by Abramsky and others for example in [5] and [2].

Using this work, we prove that these new cohomological algorithms are strictly stronger than k-consistency and k-Weisfeiler-Leman. In particular, we show that cohomological k-consistency decides solvability of linear equations with k variables per equation over all finite rings and that there is a fixed k such that $\equiv_k^{\mathbb{Z}}$ distinguishes structures which differ on Cai, Fürer and Immerman's property.

It is also interesting to compare $\rightarrow_k^{\mathbb{Z}}$ and $\equiv_k^{\mathbb{Z}}$ with other well-studied refinements of $\rightarrow_k$ and $\equiv_k$. For $\rightarrow_k$, such refinements include the algorithms of Bulatov [12] and Zhuk [35] which decide all tractable CSPs and the algorithms of Brakensiek, Guruswami, Wrochna and Živný [11] and Ciardo and Živný [14] for Promise CSPs. For $\equiv_k$, comparable approximations to $\cong$ include linear Diophantine equation methods employed by Berkholz and Grohe [9] and the invertible-map equivalence of Dawar and Holm [17] which bounds the expressive power of rank logic. The latter was recently used by Lichter [30] to demonstrate a property which is decidable in PTIME but not expressible in rank logic. In our paper, we show that $\equiv_k^{\mathbb{Z}}$, for some fixed k, can distinguish structures which differ on this property. Comparing $\rightarrow_k^{\mathbb{Z}}$ to the Bulatov-Zhuk algorithm and algorithms for PCSPs remains a direction for future work.

[1] The algorithm we call "k-Weisfeiler-Leman" is more commonly called "$(k-1)$-Weisfeiler-Leman" in the literature, see for example [13]. We prefer "k-Weisfeiler-Leman" to emphasise its relationship to k-variable logic and sets of k-local isomorphisms.

The rest of the paper proceeds as follows. Section 2 establishes some background and notation. Section 3 introduces the presheaf formulation of CSP and SI and new formulations of k-consistency and k-Weisfeiler-Leman in this framework. Section 4 demonstrates how to apply aspects of sheaf cohomology to CSP and SI and defines new algorithms along these lines. Section 5 surveys the strength of these new cohomological algorithms. Section 6 concludes with some open questions and directions for future work. Major proofs and additional background are left to the full version.

2 Background and definitions

In this section, we record some definitions and background which are necessary for our work.

2.1 Relational structures & finite model theory

Throughout this paper we use the word *structure* to mean a relational structure over some finite relational signature σ. A structure A consists of an underlying set (which we also call A) and for each relational symbol R of arity r in σ a subset $R^A \subset A^r$ or tuples related by R. A *homomorphism* of structures A, B over a common signature is a function between the underlying sets $f\colon A \to B$ which preserves related tuples. An *isomorphism* of structures is a bijection between the underlying sets which both preserves and reflects related tuples. A partial function $s\colon A \rightharpoonup B$ (seen as a set $s \subset A \times B$) is a *partial homomorphisms* if it preserves the related tuples in $\mathbf{dom}(s)$. s is a *partial isomorphism* if it is a bijection onto its image and both preserves and reflects related tuples. A partial homomorphism or isomorphism is said to be *k-local* if $|\mathbf{dom}(s)| \leq k$. For two structures over the same signature we write $\mathbf{Hom}_k(A, B)$ and $\mathbf{Isom}_k(A, B)$ respectively for the sets of k-local homomorphisms and isomorphisms from A to B.

In the paper, we make reference to several important logics from finite model theory and descriptive complexity theory. The logics we make reference to in this paper are as follows.

- Fixed-point logic with counting (written FPC) is first-order logic extended with operators for inflationary fixed-points and counting, for example see [20].
- For any natural number k, C^k is infinitary first-order logic extended with counting quantifiers with at most k variables. This logic bounds the expressive power of FPC in the sense that, for each k' there exists k such that any FPC formula in k' variables is equivalent to one in C^k. We write C^ω for the union of these logics.
- Rank logic is first-order logic extended with operators for inflationary fixed-points and computing ranks of matrices over finite fields, see [34].
- Linear algebraic logic is first-order infinitary logic extended with quantifiers for computing *all* linear algebraic functions over finite fields, see [15]. This logic bounds rank logic in the sense described above.

At different points in the history of descriptive complexity theory, both FPC and rank logic were considered as candidates for "capturing `PTIME`" and thus refuting a well-known conjecture of Gurevich [23]. Each has since been proven not to capture `PTIME`, for FPC see Cai, Fürer and Immerman [13], for rank logic see Lichter [30]. Infinitary logics such as C^ω and linear algebraic logic are capable of expressing properties which are not decidable in `PTIME` but have been shown not to contain any logic which does not capture `PTIME`. For C^ω, see Cai, Fürer and Immerman [13] and for linear algebraic logic, see Dawar, Grädel, and Lichter [16].

2.2 Constraint satisfaction problems & Structure Isomorphism

Assuming a fixed relational signature σ, we write CSP for the set of all pairs of σ-structures (A, B) such that there is a homomorphism witnessing $A \to B$. We use $CSP(B)$ to denote the set of relational structures A such that $(A, B) \in CSP$. We also use CSP and $CSP(B)$ to denote the decision problem on these sets. For general B, $CSP(B)$ is well-known to be `NP-complete`. However for certain structures B the problem is in `PTIME`. Indeed, the Bulatov-Zhuk Dichotomy Theorem (formerly the Feder-Vardi Dichotomy Conjecture) states that, for any B, $CSP(B)$ is either `NP`-complete or it is `PTIME`. Working out efficient algorithms which decide $CSP(B)$ for larger and larger classes of B was an active area of research which culminated in Bulatov and Zhuk's exhaustive classes of algorithms [12, 35].

Similarly, we write SI for the set of all pairs of σ-structures (A, B) such that there is an isomorphism witnessing $A \cong B$. The decision problem for this set is also thought not to be in `PTIME` however there are no general hardness results known for this. The best known algorithm (in the case where σ is the signature of graphs) is Babai's [8] which is quasi-polynomial.

There are many efficient algorithms which approximate the decision problems of CSP and SI. Two such examples, which are of particular importance to this paper, are the k-consistency and k-Weisfeiler-Leman algorithms. Explicit modern presentations of these algorithms can be seen, for example, in [7] and [27]. We instead focus on equivalent formulations in terms of positional Duplicator winning strategies. These are given by Kolaitis and Vardi [28] for k-consistency and Hella [24] for k-Weisfeiler-Leman. In the case of k-consistency, a pair (A, B) is accepted by the algorithm if and only if there is a non-empty subset $S \subset \mathbf{Hom}_k(A, B)$ which is downward-closed and satisfies the so-called forth property. This means any $s \in S$ with $|\mathbf{dom}(s)| < k$ satisfies the property $\mathbf{Forth}(S, s)$ which is defined as

$$\forall a \in A,\ \exists b \in B \text{ s.t. } s \cup \{(a, b)\} \in S.$$

If such an S exists we write $A \to_k B$. The similar strategy-based characterisation of k-Weisfeiler-Leman is captured by non-empty downward-closed sets $S \subset \mathbf{Isom}_k(A, B)$ where each element satisfies the bijective forth property $\mathbf{BijForth}(S, s)$ defined by

$$\exists b_s \colon A \to B \text{ a bijection s.t. } \forall a \in A\ s \cup \{(a, b_s(a))\} \in S.$$

If such an S exists we write $A \equiv_k B$. For more details, see the full version of this paper.

2.3 Presheaves & cohomology

Here we give a brief account of the category-theoretic preliminaries for this paper. For a more comprehensive introduction to category theory we refer to Chapter 1 of Leinster's textbook [29] and for a complete account of presheaves we refer to Chapter 2 of MacLane and Moerdijk [31].

Given two categories $\mathbf{C}$ and $\mathbf{S}$, an $\mathbf{S}$-valued presheaf over $\mathbf{C}$ is a contravariant functor $\mathcal{F} \colon \mathbf{C}^{op} \to \mathbf{S}$. We will assume that $\mathbf{C}$ is some subset of the powerset of some set X with subset inclusion as the morphisms. We call X the underlying space of $\mathbf{C}$. For this reason, when $U' \subset U$ in $\mathbf{C}$ we write $(\cdot)_{|_{U'}}$ for the restriction map $\mathcal{F}(U' \subset U) \colon \mathcal{F}(U) \to \mathcal{F}(U')$. We assume $\mathbf{S}$ is either the category $\mathbf{Set}$ of sets or the category $\mathbf{AbGrp}$ of abelian groups. We call $\mathbf{AbGrp}$-valued presheaves, abelian presheaves. $\mathbf{Set}$-valued presheaves are just called presheaves or presheaves of sets where there is ambiguity.

For any $\mathbf{C}$ and $\mathbf{S}$ as above, the category of presheaves $\mathbf{PrSh}(\mathbf{C}, \mathbf{S})$ has as objects the presheaves $\mathcal{F}\colon \mathbf{C}^{op} \to \mathbf{S}$ and, as morphisms, natural transformations between these functors. If $\mathbf{S}$ has a terminal object 1 (as both $\mathbf{Set}$ and $\mathbf{AbGp}$ do) then the presheaf $\mathbb{I} \in \mathbf{PrSh}(\mathbf{C}, \mathbf{S})$ which sends all elements of $\mathbf{C}$ to 1 is a terminal object in $\mathbf{PrSh}(\mathbf{C}, \mathbf{S})$. For any $\mathcal{F} \in \mathbf{PrSh}(\mathbf{C}, \mathbf{S})$, a *global section* of $\mathcal{F}$ is a natural transformation $S\colon \mathbb{I} \implies \mathcal{F}$.

3 Presheaves of local homomorphisms and isomorphisms

Some important efficient algorithms for CSP and SI involve working with sets of k-local homomorphisms between the two structures in a given instance. These sets of partial homomorphisms of domain size $\leq k$ are useful for constructing efficient algorithms because computing the sets $\mathbf{Hom}_k(A, B)$ and $\mathbf{Isom}_k(A, B)$ can be done in polynomial time in $|A| \cdot |B|$. In this section, we see that these sets can naturally be given the structure of sheaves, that the CSP and SI problems can be seen as the search for global sections of these sheaves and that the k-consistency and k-Weisfeiler-Leman algorithms can both be seen as determining the existence of certain special subpresheaves. The framework of considering sheaves of local homomorphisms and isomorphisms is novel in this work and essential for the main cohomological algorithms later. The results in Section 3.3 are from a technical report of Samson Abramsky [1] and we thank him for his permission to include them here.

3.1 Defining presheaves of homomorphisms and isomorphisms

Let A and B be relational structures over the same signature. A *partial homomorphism* is a partial function $s\colon A \rightharpoonup B$ that preserves related tuples in $\mathbf{dom}(s)$. A *partial isomorphism* is a partial homomorphism $s\colon A \rightharpoonup B$ which is injective and reflects related tuples from $\mathbf{im}(s)$. A *k-local homomorphism (resp. isomorphism)* is a partial homomorphism (resp. isomorphism) s such that $|\mathbf{dom}(s)| \leq k$. We write $\mathbf{Hom}_k(A, B)$ (resp. $\mathbf{Isom}_k(A, B)$) for the sets of k-local homomorphisms (resp. isomorphisms). We write $\mathbf{Hom}(A, B)$ for the union $\bigcup_{1 \leq k \leq |A|} \mathbf{Hom}_k(A, B)$ and $\mathbf{Isom}(A, B)$ for the union $\bigcup_{1 \leq k \leq |A|} \mathbf{Isom}_k(A, B)$.

It is not hard to see that these sets can be given the structure of presheaves on the underlying space A. Indeed, we define the *presheaf of homomorphisms from A to B* $\mathcal{H}(A, B)\colon \mathbf{P}(\mathbf{A})^{op} \to \mathbf{Set}$ as $\mathcal{H}(A, B)(U) = \{s \in \mathbf{Hom}(A, B) \mid \mathbf{dom}(s) = U\}$ with restriction maps $\mathcal{H}(A, B)(U' \subset U)$ given by the restriction of partial homomorphisms $(\cdot)_{|_{U'}}$. Similarly, let $\mathcal{I}(A, B)$ be the subpresheaf of $\mathcal{H}(A, B)$ containing only partial isomorphisms. Now, consider the cover of A by subsets of size at most k, written $A^{\leq k} \subset P(A)$. We define the *presheaves of k-local homomorphisms and isomorphisms* $\mathcal{H}_k(A, B)$ and $\mathcal{I}_k(A, B)$ as the functors $\mathcal{H}(A, B)$ and $\mathcal{I}(A, B)$ restricted to the subcategory $(\mathbf{A}^{\leq \mathbf{k}})^{op} \subset \mathbf{P}(\mathbf{A})^{op}$.

We now see how these presheaves and their global sections encode the CSP and SI problems for the instance (A, B).

3.2 CSP and SI as search for global sections

Fix an instance (A, B) for the CSP or SI problem and let $\mathcal{H}$ and $\mathcal{I}$ stand for the presheaves of all partial homomorphisms and isomorphisms between A and B defined in the last section. For either of these presheaves $\mathcal{S}$ a global section $s\colon \mathbb{I} \implies \mathcal{S}$ is a collection $\{s_U \in \mathcal{S}(U)\}_{U \in P(A)}$ where naturality implies that for any subsets U and U' of A $(s_U)_{|_{U \cap U'}} = (s_{U'})_{|_{U \cap U'}}$. As the poset $P(A)$ has a maximal element, namely A, any such global section is determined by a choice of $s_A \in \mathcal{S}(A)$. This leads us to the following observation.

▶ **Observation 1.** *Given a pair (A, B) relational structures over the same signature then*

$$(A, B) \in CSP \iff \mathcal{H} \text{ has a global section}$$

and if $|A| = |B|$ then

$$(A, B) \in SI \iff \mathcal{I} \text{ has a global section.}$$

This observation reframes the CSP and SI problems in terms of presheaves but algorithmically this not a particularly useful restating as computing the full objects $\mathcal{H}$ and $\mathcal{I}$ requires solving the CSP and SI problems for all subsets of A and B. A much more interesting equivalent condition is that for large enough k, whether or not a particular instance (A, B) is in CSP or SI is determined by the global sections of the presheaves of k-local homomorphisms and isomorphisms.

▶ **Lemma 2.** *For a pair (A, B) relational structures over the same signature, σ, and k at least the arity of sigma then*

$$(A, B) \in CSP \iff \mathcal{H}_k \text{ has a global section}$$

and if $|A| = |B|$ then

$$(A, B) \in SI \iff \mathcal{I}_k \text{ has a global section.}$$

Proof. See full version. ◀

This is more interesting than the previous observation as $\mathcal{H}_k$ and $\mathcal{I}_k$ can be computed for any relational structures A and B in $\mathcal{O}(\text{poly}(|A| \cdot |B|))$. Indeed, we can just list all $\mathcal{O}(|A|^k \cdot |B|^k)$ possible k-local functions and check which ones preserve (and reflect) related tuples. This also gives us an interesting starting point for designing efficient algorithms for approximating CSP and SI. In particular, any efficient algorithms which finds obstacles to the existence of global sections in $\mathcal{H}_k$ and $\mathcal{I}_k$ will provide a tractable approximation to CSP and SI. We now see how this approach can be used to capture some classical approximations of these problems.

3.3 Algorithms and games in terms of presheaves

In this section, we consider the approximations $A \to_k B$ and $A \equiv_k B$ to CSP and SI which are computed respectively by the k-consistency and k-Weisfeiler-Leman algorithms and we show that these algorithms can be seen as searching for certain obstructions to global sections in $\mathcal{H}_k(A, B)$ and $\mathcal{I}_k(A, B)$. In particular, we define efficiently computable monotone operators on subpresheaves of $\mathcal{H}_k$ and $\mathcal{I}_k$ and show that they have non-empty greatest fixpoints if and only if (A, B) are accepted by k-consistency and k-Weisfeiler-Leman respectively. Proposition 3 is reproduced with permission from an unpublished technical report of Samson Abramsky and the formulation of the fixpoint operators is inspired by the same report.

3.3.1 Flasque presheaves and k-consistency

Recall that $A \to_k B$ if and only if there is a positional winning strategy for Duplicator in the existential k-pebble game [18] and that a presheaf $\mathcal{F}$ is flasque if all of the restriction maps $\mathcal{F}(U \subset U')$ are surjective. In a recent technical report, Abramsky [1] proves the following characterisation of these strategies in our presheaf setting.

▶ **Proposition 3.** *For A, B relational structures and any k there is a bijection between:*

- *positional strategies in the existential k-pebble game from A to B, and*
- *non-empty flasque subpresheaves $\mathcal{S} \subset \mathcal{H}_k(A, B)$.*

This gives an alternative description of the k-consistency algorithm as constructing the largest flasque subpresheaf $\overline{\mathcal{H}_k}$ of $\mathcal{H}_k$ and checking if it is empty. As pointed out by Abramsky [1], this is the process of coflasquification of the presheaf $\mathcal{H}_k$ and can be seen as dual to the Godement construction [21], an important early construction in homological algebra. $\overline{\mathcal{H}_k}$ can be computed efficiently as the greatest fixpoint of the presheaf operator $(\cdot)^{\uparrow\downarrow}$ which computes the largest subpresheaf of a presheaf $\mathcal{S} \subset \mathcal{H}_k$ such that every $s \in \mathcal{S}^{\uparrow\downarrow}(C)$ satisfies the forth property $\mathbf{Forth}(\mathcal{S}, s)$. For further details see the full version of this paper.

3.3.2 Greatest fixpoints and k-Weisfeiler-Leman

In a similar way to the k-consistency algorithm, k-Weisfeiler-Leman can be formulated as determining the existence of a positional strategy for Duplicator in the k-pebble bijection game between A and B. This inspires the definition of another efficiently computable presheaf operator $(\cdot)^{\#\downarrow}$ which computes the largest subpresheaf of a presheaf $\mathcal{S} \subset \mathcal{I}_k$ such that for every $s \in \mathcal{S}^{\#\downarrow}(C)$ satisfies the bijective forth property $\mathbf{BijForth}(\mathcal{S}, s)$. We call the greatest fixpoint of this operator $\overline{\overline{\mathcal{S}}}$ and we have that $A \equiv_k B$ if and only if $\overline{\overline{\mathcal{I}_k}}$ is non-empty. For more details, see the full version of this paper.

To conclude, in this section, we have seen how to reformulate the search for homomorphisms and isomorphisms between relational structures A and B as the search for global sections in the presheaves $\mathcal{H}_k(A, B)$ and $\mathcal{I}_k(A, B)$. We have also seen that well-known approximations of homomorphism and isomorphism, $\rightarrow_k$ and $\equiv_k$, can be computed as greatest fixpoints of presheaf operators which remove elements which cannot form part of any global section. In the next section, we look at sheaf-theoretic obstructions to forming a global section which come from cohomology and see how these can be used to define stronger approximations of homomorphism and isomorphism.

4 Cohomology of local homomorphisms and isomorphisms

As we showed in the previous section, an instance of CSP and SI with input (A, B) can be seen as determining the existence of a global section for the presheaf $\mathcal{H}_k(A, B)$ or $\mathcal{I}_k(A, B)$ respectively and that the classic k-consistency and k-Weisfeiler-Leman algorithms can be reformulated as computing greatest fixed points of presheaf operations which successively remove sections which are obstructed from being part of some global section. In this section, we extend these algorithms by considering further efficiently computable obstructions which arise naturally from presheaf cohomology. From this we derive new cohomological algorithms for CSP and SI.

4.1 Cohomology and local vs. global problems

The notion of computing cohomology valued in an **AbGp**-valued presheaf $\mathcal{F}$ on a topological space X has a long history in algebraic geometry and algebraic topology which dates back to Grothendieck's seminal paper on the topic [22]. The cohomology valued in $\mathcal{F}$ consists of a sequence of abelian groups $H^i(X, \mathcal{F})$ where $H^0(X, \mathcal{F})$ is the free $\mathbb{Z}$-module over global sections of $\mathcal{F}$. As seen in the previous section we may be interested in such global sections but their existence may be difficult to determine. This is where the functorial nature of cohomology is extremely useful. Indeed, any short exact sequence of presheaves

$$0 \rightarrow \mathcal{F}_L \rightarrow \mathcal{F} \rightarrow \mathcal{F}_R \rightarrow 0$$

lifts to a long exact sequence of cohomology groups

$$0 \to H^0(X, \mathcal{F}_L) \to H^0(X, \mathcal{F}) \to H^0(X, \mathcal{F}_R) \to H^1(X, \mathcal{F}_L) \to \dots .$$

This tells us that the global sections of $\mathcal{F}_R$ which are not images of global sections of $\mathcal{F}$ are mapped to non-trivial elements of the group $H^1(X, \mathcal{F}_L)$ by the maps in this sequence. This means that these higher cohomology groups can be seen as a source of obstacles to lifting "local" solutions in $\mathcal{F}_R$ to "global" solutions in $\mathcal{F}$.

An important recent example of such an application of cohomology to finite structures can be found in the work of Abramsky, Barbosa, Kishida, Lal and Mansfield [2] in quantum foundations. They show that cohomological obstructions of the type described above can be used to detect contextuality (locally consistent measurements which are globally inconsistent) in quantum systems which were earlier given a presheaf semantics by Abramsky and Brandenburger [3]. In the full version of this paper, we describe these obstructions in general and show how the presheaves we constructed in the last section admit the same cohomological obstructions. This similarity inspires the definitions and algorithms which follow in the next two sections.

4.2 $\mathbb{Z}$-local sections and $\mathbb{Z}$-extendability

Returning to presheaves of local homomorphisms and isomorphisms let $\mathcal{S}$ be a subpresheaf of $\mathcal{H}_k$. Then we define the presheaf of $\mathbb{Z}$-linear local sections of $\mathcal{S}$ to be the presheaf of formal $\mathbb{Z}$-linear sums of local sections of $\mathcal{S}$. This means that for any $C \in A^{\leq k}$

$$\mathbb{Z}\mathcal{S}(C) := \left\{ \sum_{s \in \mathcal{S}(C)} \alpha_s s \;\mid\; \alpha_s \in \mathbb{Z} \right\}.$$

This is an abelian presheaf on $A^{\leq k}$ and we call the global sections $\{r_U \in \mathbb{Z}\mathcal{S}(U)\}_{U \in A^{\leq k}}$ $\mathbb{Z}$-linear global sections of $\mathcal{S}$. We say that a local section $s \in \mathcal{S}(C)$ is $\mathbb{Z}$-extendable if there is a $\mathbb{Z}$-linear global section $\{r_U \in \mathbb{Z}\mathcal{S}(U)\}_{U \in A^{\leq k}}$ such that $r_C = s$. We write this condition as $\mathbb{Z}\mathbf{ext}(\mathcal{S}, s)$. As outlined by Abramsky, Barbosa and Mansfield [5], this condition corresponds to the absence of a cohomological obstruction to $\mathcal{S}$ containing a global section involving s.

Importantly for our purposes, deciding the condition $\mathbb{Z}\mathbf{ext}(\mathcal{S}, s)$ for any $\mathcal{S} \subset \mathcal{H}_k(A, B)$ is computable in polynomial time in the sizes of A and B. This is because the compatibility conditions for a collection $\{r_U \in \mathbb{Z}\mathcal{S}(U)\}_{U \in A^{\leq k}}$ being a global section of $\mathbb{Z}\mathcal{S}$ can be expressed as a system of polynomially many linear equations in polynomially many variables. Indeed, we write each r_U as $\sum_{s \in \mathcal{S}(U)} \alpha_s s$ where α_s is a variable for each $s \in \mathcal{S}(U)$. This gives a total number of variables bounded by $\mathcal{O}(|A|^k \cdot |B|^k)$, the size of $\mathbf{Hom}_k(A, B)$. For each of the $\mathcal{O}(|A|^{2k})$ pairs of contexts $U, U' \in A^{\leq k}$, the compatibility condition $(r_U)_{|U \cap U'} = (r_{U'})_{|U \cap U'}$ yields a linear equation in the α_s variables for each $s' \in \mathcal{S}(U \cap U')$, leading to a total number of equations bounded by $\mathcal{O}(|A|^{2k} \cdot |B|^k)$. By an algorithm of Kannan and Bachem [26] can be solved in polynomial time in the sizes of A and B. This allows us to define the following efficient algorithms for CSP and SI based on removing cohomological obstructions.

4.3 Cohomological algorithms for CSP and SI

We saw in Section 3 that the k-consistency and k-Weisfeiler-Leman algorithms can be recovered as greatest fixpoints of presheaf operators removing local sections which fail the forth and bijective-forth properties respectively. Now that we have from cohomological considerations a new necessary condition $\mathbb{Z}\mathbf{ext}(\mathcal{S}, s)$ for a local section to feature in a global section of $\mathcal{S}$, we can define natural extensions to the k-consistency and k-Weisfeiler-Leman algorithms as follows.

4.3.1 Cohomological k-consistency

To define the cohomological k-consistency algorithm, we first define an operator which removes those local sections which admit a cohomological obstruction. Let $(\cdot)^{\mathbb{Z}\downarrow}$ be the operator which computes for a given presheaf $\mathcal{S} \subset \mathcal{H}_k$ the subpresheaf $\mathcal{S}^{\mathbb{Z}\downarrow}$ where $\mathcal{S}^{\mathbb{Z}\downarrow}(C)$ contains exactly those local sections $s \in \mathcal{S}(C)$ which satisfy both the forth property $\mathbf{Forth}(\mathcal{S}, s)$ and the $\mathbb{Z}$-extendability property $\mathbb{Z}\mathbf{ext}(\mathcal{S}, s)$. As this process may remove the local sections in $\mathcal{S}$ which witness the extendability of other local sections we need to take a fixpoint of this operator to get a presheaf with the right extendability properties at every local section. So, we write $\overline{\mathcal{S}}^{\mathbb{Z}}$ for the greatest fixpoint of this operator starting from $\mathcal{S}$. As both $\mathbf{Forth}(\mathcal{S}, s)$ and $\mathbb{Z}\mathbf{ext}(\mathcal{S}, s)$ are both computable in polynomial time in the size of $\mathcal{S}$ and $\overline{\mathcal{S}}^{\mathbb{Z}}$ has a global section if and only if $\mathcal{S}$ has a global section, this allows us to define the following efficient algorithm for approximating CSP.

▶ **Definition 4.** *The* cohomological k-consistency algorithm *accepts an instance (A, B) if the greatest fixpoint $\overline{\mathcal{H}_k(A, B)}^{\mathbb{Z}}$ is non-empty and otherwise rejects.*
If (A, B) is accepted by this algorithm we write $A \to_k^{\mathbb{Z}} B$ and say that the instance (A, B) is cohomologically k-consistent.

We conclude this section by showing that the relation $\to_k^{\mathbb{Z}}$ is transitive.

▶ **Proposition 5.** *For all k, given A, B and C structures over a common finite signature*

$$A \to_k^{\mathbb{Z}} B \to_k^{\mathbb{Z}} C \implies A \to_k^{\mathbb{Z}} C.$$

Proof. See full version. ◀

4.3.2 Cohomological k-Weisfeiler-Leman

We now define cohomological k-equivalence to generalise k-WL-equivalence in the same way as we did for cohomological k-consistency, by removing local sections which are not $\mathbb{Z}$-extendable. As $\mathbb{Z}$-extendability in $S \subset \mathbf{Isom}_k(A, B)$ is not *a priori* symmetric in A and B we need to check that both s is $\mathbb{Z}$-extendable in S and s^{-1} is $\mathbb{Z}$-extendable in $S^{-1} = \{t^{-1} \mid t \in S\}$. We call this s being *$\mathbb{Z}$-bi-extendable* in S and write it as $\mathbb{Z}\mathbf{bext}(\mathcal{S}, s)$. We incorporate this into a new presheaf operator $(\cdot)^{\mathbb{Z}\#}$ as follows. Given a presheaf $\mathcal{S} \subset \mathcal{I}_k$ let $\mathcal{S}^{\mathbb{Z}\#}$ be the largest subpresheaf of $\mathcal{S}$ such that every $s \in \mathcal{S}^{\mathbb{Z}\#}(C)$ satisfies both the bijective forth property $\mathbf{BijForth}(\mathcal{S}, s)$ and the $\mathbb{Z}$-bi-extendability property $\mathbb{Z}\mathbf{bext}(\mathcal{S}, s)$. We write $\overline{\overline{\mathcal{S}}}^{\mathbb{Z}}$ for the greatest fixpoint of this operator starting from $\mathcal{S}$. As both $\mathbf{BijForth}(\mathcal{S}, s)$ and $\mathbb{Z}\mathbf{bext}(\mathcal{S}, s)$ are computable in polynomial time in the size of $\mathcal{S}$ and $\overline{\overline{\mathcal{S}}}^{\mathbb{Z}}$ has a global section if and only if $\mathcal{S}$ has a global section, this allows us to define the following efficient algorithm for approximating SI.

▶ **Definition 6.** *The* cohomological k-Weisfeiler-Leman *accepts an instance (A, B) if the greatest fixpoint $\overline{\overline{\mathcal{I}_k(A, B)}}^{\mathbb{Z}}$ is non-empty and otherwise rejects.*
If (A, B) is accepted by this algorithm we write $A \equiv_k^{\mathbb{Z}} B$ and say that the instance (A, B) is cohomologically k-equivalent.

Finally, we observe that the existence of a non-empty subpresheaf of $\mathcal{I}_k$ satisfying the $\mathbf{BijForth}$ and $\mathbb{Z}\mathbf{bext}$ properties also satisfies the conditions for witnessing cohomological k-consistency of the pairs (A, B) and (B, A). Formally we have

▶ **Observation 7.** *For any two structures A and B, $A \equiv_k^{\mathbb{Z}} B$ implies that $A \to_k^{\mathbb{Z}} B$ and $B \to_k^{\mathbb{Z}} A$.*

In Section 5, we will demonstrate the power of these new algorithms by showing that both cohomological k-consistency and cohomological k-Weisfeiler-Leman can solve instances of CSP and SI on which the non-cohomological versions fail. Before doing this, we briefly review some other algorithms for CSP and SI which involve solving systems of linear equations and establish a possible connection to be explored in future work.

4.4 Other algorithms for CSP and SI

While the connections to cohomology in approximating CSP and SI are novel in this paper, the algorithms introduced here are not the first to use solving systems of linear equations to approximate these problems.

On the CSP side, some examples of such algorithms include the BLP+AIP [11] and CLAP [14] algorithms studied in the Promise CSP community. One difference here is that for an instance (A, B) the variables in BLP and AIP are indexed by valid assignments to each variable and to each related tuple instead of being indexed by valid k-local homomorphisms as in the algorithm derived above. This means that directly comparing these algorithms as stated is not straightforward and is beyond the scope of this paper. However, it seems likely that these algorithms can also be expressed in terms of appropriate presheaves. For example, let $\mathbf{C}(\mathbf{A})$ be the category whose objects are the elements of A and the related tuples of A and with maps for each projection from a related tuple to an element, and let the **Set**-valued presheaf $\mathcal{H}_C(A, B)$ on $\mathbf{C}(\mathbf{A})$ map any $a \in A$ to the set of all elements in B and any $\mathbf{a} \in R^A$ to the set of all related tuples R^B. Then, in a similar way to above, we can see that global sections of $\mathcal{H}_C$ are homomorphisms from A to B. In future work, we will compare the fixpoints $\overline{\mathcal{H}_C}$ and $\overline{\mathcal{H}_C}^{\mathbb{Z}}$ with solutions to the BLP and AIP systems of equations and we will explore a possible presheaf representation for CLAP.

On the SI side, Berkholz and Grohe [9] have studied $\mathbb{Z}$-linear versions of the Sherali-Adams hierarchy of relaxations of the graph isomorphism problem. They establish that no level of this hierarchy decides the full isomorphism relation on graphs. Their algorithm for the k^{th} level of the hierarchy appears similar to checking the $\mathbb{Z}$-extendability in $\mathcal{H}_k^{A,B}$ of the empty solution $\epsilon \in \mathcal{H}_k^{A,B}(\emptyset)$. A full comparison of this algorithm and the algorithm described above is an interesting direction for future work.

5 The (unreasonable) effectiveness of cohomology in CSP and SI

In this section, we prove that the new algorithms arising from this cohomological approach to CSP and SI are substantially more powerful than the k-consistency and k-Weisfeiler-Leman algorithms. In particular, we show that cohomological k-consistency resolves CSP over all domains of arity less than or equal to k which admit a ring representation and that for a fixed small k cohomological k-Weisfeiler-Leman can distinguish structures which differ on a very general form of the CFI property, in particular, showing that cohomological k-Weisfeiler-Leman can distinguish a property which Lichter [30] claims not to be expressible in rank logic.

5.1 Cohomological k-consistency solves all affine CSPs

In this section, we demonstrate the power of the cohomological k-consistency algorithm by proving that it can decide the solvability of systems of equations over finite rings.

To express the main theorem of this section in terms of the finite relational structures on which our algorithm is defined, we first need to fix a notion of ring representation of a relational structure. Let A be a relational structure over signature σ with relations given by $\{R^A\}_{R\in\sigma}$. We say that A has a *ring representation* if we can give the set A a ring structure $(A,+,\cdot,0,1)$ such that for every relational symbol $R \in \sigma$ the set $R^A \subset A^m$ is an affine subset of the ring $(A,+,\cdot,0,1)$, meaning that there exists $b_1^R,\dots,b_m^R,a^R \in A$ such that

$$R^A = \{\mathbf{x} \in A^m \mid \sum_{i\in[m]} b_i^R \cdot x_i = a^R\}$$

With this necessary background we state the main theorem of this section.

▶ **Theorem 8.** *For any structure B with a ring representation, there is a k such that the cohomological k-consistency algorithm decides* $\mathbf{CSP}(B)$.
Alternatively stated, there exists a k such that for all σ-structures A

$$A \rightarrow_k^{\mathbb{Z}} B \iff A \rightarrow B$$

Proof. See full version. ◀

This theorem is notable because there are relational structures B with ring representations for which there are families of structures A_k such that $A_k \rightarrow_k B$ but $A_k \not\rightarrow B$, see for example the examples given by Feder and Vardi [18]. Furthermore, there exist pairs (A_k, B_k) where $A_k \equiv_k B_k$, $B_k \rightarrow B$ and $A_k \rightarrow_k B$ but $A_k \not\rightarrow B$, see for example the work of Atserias, Bulatov and Dawar [6]. As the sequence of relations $\equiv_k$ bounds the expressive power of FPC, this effectively proves that the solvability of systems of linear equations over $\mathbb{Z}$, which is central to the cohomological k-consistency algorithm, is not expressible in FPC. This result was not previously known to the author.

5.2 Cohomological k-Weisfeiler-Leman decides the CFI property

The Cai-Fürer-Immerman construction [13] on ordered finite graphs is a very powerful tool for proving expressiveness lower bounds in descriptive complexity theory. While it was originally used to separate the infinitary k variable logic with counting from `PTIME`, it has since been used in adapted forms to prove bounds on invertible maps equivalence [15], computation on Turing machines with atoms [10] and rank logic [30]. In this section, we show that $\equiv_k^{\mathbb{Z}}$ separates a very general form of this

The version we consider in this paper is parameterised by a prime power q and takes any totally ordered graph $(G,<)$ and any map $g\colon E(G) \rightarrow \mathbb{Z}_q$ to a relational structure $\mathbf{CFI}_q(G,g)$. The construction effectively encodes a system of linear equations over $\mathbb{Z}_q$ based on the edges of G and the "twists" introduced by the labels g. The result is the following well-known fact.

▶ **Fact 9.** *For any prime power, q, ordered graph G, and functions $g,h\colon E(G) \rightarrow \mathbb{Z}_q$,*

$$\mathbf{CFI}_q(G,g) \cong \mathbf{CFI}_q(G,h) \iff \sum g = \sum h$$

We say that the structure $\mathbf{CFI}_q(G,g)$ has the *CFI property* if $\sum g = 0$. For more details on this construction we refer to the recent paper of Lichter [30] whose presentation we follow in the full version of this paper.

We now recall the two major separation results based on this construction. The first is a landmark result of descriptive complexity from the early 1990's.

▶ **Theorem 10** (Cai, Fürer, Immerman [13]). *There is a class of ordered (3-regular) graphs $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$ such that in the respective class of CFI structures*

$$\mathcal{K} = \{\mathbf{CFI}_2(G,g) \mid G \in \mathcal{G}, g\colon V(G) \to \mathbb{Z}_2\}$$

the CFI property is decidable in polynomial-time but cannot be expressed in FPC.

The second is a recent breakthrough due to Moritz Lichter.

▶ **Theorem 11** (Lichter [30]). *There is a class of ordered graphs $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$ such that in the respective class of CFI structures*

$$\mathcal{K} = \{\mathbf{CFI}_{2^k}(G,g) \mid G \in \mathcal{G}\}$$

the CFI property is decidable in polynomial-time (indeed, expressible in choiceless polynomial time) but cannot be expressed in rank logic.

Despite this CFI property proving to be inexpressible in both FPC and rank logic, we show that (perhaps surprisingly) there is a fixed k such that cohomological k-Weisfeiler-Leman algorithm can separate structures which differ on this property in the following general way. The proof of this theorem relies the on showing that $\equiv_k^{\mathbb{Z}}$ behaves well with logical interpretations and the details are left to the full version of this paper.

▶ **Theorem 12.** *There is a fixed k such that for any q given $\mathbf{CFI}_q(G,g)$ and $\mathbf{CFI}_q(G,h)$ with $\sum g = 0$ we have*

$$\mathbf{CFI}_q(G,g) \equiv_k^{\mathbb{Z}} \mathbf{CFI}_q(G,h) \iff \mathbf{CFI}_q(G,g) \cong \mathbf{CFI}_q(G,h)$$

Proof. See full version. ◀

As a direct consequence of this result, there is some k such that the set of structures with the CFI property in Lichter's class $\mathcal{K}$ from Theorem 11 is closed under $\equiv_k^{\mathbb{Z}}$. This means that, by the conclusion of Theorem 11, the equivalence relation $\equiv_k^{\mathbb{Z}}$ can distinguish structures which disagree on a property that is not expressible in rank logic. Indeed, Dawar, Grädel and Lichter [16] show further that this property is also inexpressible in linear algebraic logic. By the definition of our algorithm for $\equiv_k^{\mathbb{Z}}$ this implies that solvability of systems of $\mathbb{Z}$-linear equations is not definable in linear algebraic logic.

6 Conclusions & future work

In this paper, we have presented novel approach to CSP and SI in terms of presheaves and have used this to derive efficient generalisations of the k-consistency and k-Weisfeiler-Leman algorithms, based on natural considerations of presheaf cohomology. We have shown that the relations, $\to_k^{\mathbb{Z}}$ and $\equiv_k^{\mathbb{Z}}$, computed by these new algorithms are strict refinements of their well-studied classical counterparts $\to_k$ and $\equiv_k$. In particular, we have shown in Theorem 8 that cohomological k-consistency suffices to solve linear equations over all finite rings and in Theorem 12 that cohomological k-Weisfeiler-Leman distinguishes positive and negative instances of the CFI property on the classes of structures studied by Cai, Fürer and Immerman [13] and more recently by Lichter [30]. These results have important consequences for descriptive complexity theory showing, in particular, that the solvability of systems of linear equations over $\mathbb{Z}$ is not expressible in FPC, rank logic or linear algebraic logic. Furthermore, the results of this paper demonstrate the unexpected effectiveness of a cohomological approach to constraint satisfaction and structure isomorphism, analogous to that pioneered by Abramsky and others for the study of quantum contextuality.

The results of this paper suggest several directions for future work to establish the extent and limits of this cohomological approach. We ask the following questions which connect it to important themes in algorithms, logic and finite model theory.

Cohomology and constraint satisfaction. Firstly, Bulatov and Zhuk's recent independent resolutions of the Feder-Vardi conjecture [12, 35], show that for all domains B either $\mathbf{CSP}(B)$ is `NP-Complete` or B admits a weak near-unanimity polymorphism and $\mathbf{CSP}(B)$ is tractable. As the cohomological k-consistency algorithm expands the power of the k-consistency algorithm which features as one case of Bulatov and Zhuk's general efficient algorithms, we ask if it is sufficient to decide all tractable **CSP**s.

▶ **Question 13.** *For all domains B which admit a weak near-unanimity polymorphism, does there exists a k such that for all A*

$$A \to B \iff A \to_k^{\mathbb{Z}} B?$$

Cohomology and structure isomorphism. Secondly, as cohomological k-Weisfeiler-Leman is an efficient algorithm for distinguishing some non-isomorphic relational structures we ask if it distinguishes all non-isomorphic structures. As the best known structure isomorphism algorithm is quasi-polynomial [8], we do not expect a positive answer to this question but expect that negative answers would aid our understanding of the hard cases of structure isomorphism in general.

▶ **Question 14.** *For every signature σ does there exists a k such that for all σ-structures A, B*

$$A \cong B \iff A \equiv_k^{\mathbb{Z}} B?$$

Cohomology and game comonads. Thirdly, as $\to_k$ and $\equiv_k$ have been shown by Abramsky, Dawar, and Wang [4] to be correspond to the coKleisli morphisms and isomorphisms of a comonad $\mathbb{P}_k$, we ask whether a similar account can be given to $\to_k^{\mathbb{Z}}$ and $\equiv_k^{\mathbb{Z}}$. As the coalgebras of the $\mathbb{P}_k$ comonad relate to the combinatorial notion of treewidth, an answer to this question could provide a new notion of "cohomological" treewidth.

▶ **Question 15.** *Does there exist a comonad $\mathbb{C}_k$ for which the notion of morphism and isomorphism in the coKleisli category are $\to_k^{\mathbb{Z}}$ and $\equiv_k^{\mathbb{Z}}$?*

The search for a logic for `PTIME`. Finally, as the algorithms for $\to_k^{\mathbb{Z}}$ and $\equiv_k^{\mathbb{Z}}$ are likely expressible in rank logic extended with a quantifier for solving systems of linear equations over $\mathbb{Z}$ and as $\equiv_k^{\mathbb{Z}}$ distinguishes all the best known family separating rank logic from `PTIME`, we ask if solving systems of equations over $\mathbb{Z}$ is enough to capture all `PTIME` queries.

▶ **Question 16.** *Is there a logic FPC+rk+$\mathbb{Z}$ incorporating solvability of $\mathbb{Z}$-linear equations into rank logic which captures `PTIME`?*

References

1 Samson Abramsky. Notes on presheaf representations of strategies and cohomological refinements of k-consistency and k-equivalence, 2022. doi:10.48550/ARXIV.2206.12156.

2 Samson Abramsky, Rui Soares Barbosa, Kohei Kishida, Raymond Lal, and Shane Mansfield. Contextuality, Cohomology and Paradox. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 211–228, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2015.211.

3 Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 13(11):113036, November 2011. doi:10.1088/1367-2630/13/11/113036.

4 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '17. IEEE Press, 2017.

5 Samson Abramsky, Shane Mansfield, and Rui Soares Barbosa. The cohomology of non-locality and contextuality. In B. Jacobs, P. Selinger, and B. Spitters, editors, *Proceedings of 8th International Workshop on Quantum Physics and Logic (QPL 2011)*, volume 95, pages 1–14, 2011. doi:10.4204/EPTCS.95.1.

6 Albert Atserias, Andrei Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. In *In ICALP'07, volume 4596 of LNCS*, pages 558–570, 2007.

7 Albert Atserias, Andrei A. Bulatov, and Víctor Dalmau. On the power of k -consistency. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2007. doi:10.1007/978-3-540-73420-8_26.

8 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 684–697, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897542.

9 Christoph Berkholz and Martin Grohe. Linear diophantine equations, group CSPs, and graph isomorphism. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 327–339. SIAM, 2017. doi:10.1137/1.9781611974782.21.

10 Mikolaj Bojanczyk, Bartek Klin, Slawomir Lasota, and Szymon Torunczyk. Turing machines with atoms. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 183–192, 2013. doi:10.1109/LICS.2013.24.

11 Joshua Brakensiek, Venkatesan Guruswami, Marcin Wrochna, and Stanislav Živný. The power of the combined basic linear programming and affine relaxation for promise constraint satisfaction problems. *SIAM Journal on Computing*, 49(6):1232–1248, 2020. doi:10.1137/20M1312745.

12 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

13 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.

14 Lorenzo Ciardo and Stanislav Zivný. CLAP: A new algorithm for promise CSPs. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1057–1068. SIAM, 2022. doi:10.1137/1.9781611977073.46.

15 Anuj Dawar, Erich Grädel, and Wied Pakusa. Approximations of Isomorphism and Logics with Linear-Algebraic Operators. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112:1–112:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.112.

16 Anuj Dawar, Erich Grädel, and Moritz Lichter. Limitations of the invertible-map equivalences, 2021. arXiv:2109.07218.

17 Anuj Dawar and Bjarki Holm. Pebble games with algebraic rules. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 251–262, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

18 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, 1998. doi:10.1137/S0097539794266766.

19 Marcelo Fiore and Sam Staton. Comparing operational models of name-passing process calculi. *Information and Computation*, 204(4):524–560, 2006. Seventh Workshop on Coalgebraic Methods in Computer Science 2004. doi:10.1016/j.ic.2005.08.004.

20 Jörg Flum and Martin Grohe. On fixed-point logic with counting. *The Journal of Symbolic Logic*, 65(2):777–787, 2000. URL: http://www.jstor.org/stable/2586569.

21 Roger Godement. Topologie algébrique et théorie des faisceaux. *The Mathematical Gazette*, 44:69, 1960.

22 Alexander Grothendieck. Sur quelques points d'algèbre homologique, I. *Tohoku Mathematical Journal*, 9(2):119–221, 1957. doi:10.2748/tmj/1178244839.

23 Yuri Gurevich. Logic and the challenge of computer science, 1988.

24 Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

25 Neil Immerman and Eric Lander. *Describing Graphs: A First-Order Approach to Graph Canonization*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

26 Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–9, November 1979. Copyright - Copyright] © 1979 © Society for Industrial and Applied Mathematics; Last updated - 2021-09-11. URL: https://ezp.lib.cam.ac.uk/login?url=https://www.proquest.com/scholarly-journals/polynomial-algorithms-computing-smith-hermite/docview/918468506/se-2?accountid=9851.

27 Sandra Kiefer. *Power and limits of the Weisfeiler-Leman algorithm.* Dissertation, RWTH Aachen University, Aachen, 2020. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2020. doi:10.18154/RWTH-2020-03508.

28 P.G. Kolaitis and M.Y. Vardi. On the expressive power of Datalog: Tools and a case study. *Journal of Computer and System Sciences*, 51(1):110–134, 1995. doi:10.1006/jcss.1995.1055.

29 Tom Leinster. *Categories, functors and natural transformations*, pages 9–40. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2014. doi:10.1017/CBO9781107360068.003.

30 Moritz Lichter. Separating rank logic from polynomial time. 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/LICS52264.2021.9470598.

31 Saunders Mac Lane and Ieke Moerdijk. *Sheaves of Sets*, pages 64–105. Springer New York, New York, NY, 1994. doi:10.1007/978-1-4612-0927-0_4.

32 Cristina Matache, Sean K. Moss, and Sam Staton. Recursion and sequentiality in categories of sheaves. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 25:1–25:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.25.

33 Baudot P. and Bennequin D. The homological nature of entropy. *Entropy*, 17(5):3253–3318, 2015. Cited by: 27; All Open Access, Gold Open Access, Green Open Access. doi:10.3390/e17053253.

34 Wied Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen University, 2016. URL: `https://logic.rwth-aachen.de/~pakusa/diss.pdf`.

35 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5), August 2020. `doi:10.1145/3402029`.

Deciding Emptiness for Constraint Automata on Strings with the Prefix and Suffix Order

Dominik Peteler ✉
Universität Leipzig, Germany

Karin Quaas ✉
Universität Leipzig, Germany

Abstract

We study constraint automata that accept data languages on finite string values. Each transition of the automaton is labelled with a constraint restricting the string value at the current and the next position of the data word in terms of the prefix and the suffix order. We prove that the emptiness problem for such constraint automata with Büchi acceptance condition is NL-complete. We remark that since the constraints are formed by two partial orders, prefix and suffix, we cannot exploit existing techniques for similar formalisms. Our decision procedure relies on a decidable characterization for those infinite paths in the graph underlying the automaton that can be completed with string values to yield a Büchi-accepting run. Our result is - to the best of our knowledge - the first work in this context that considers both prefix and suffix, and it is a first step into answering an open question posed by Demri and Deters.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Data Languages, Strings, Constraints, Prefix, Suffix, Automata, Linear Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.76

Funding *Karin Quaas*: Funded by the Deutsche Forschungsgemeinschaft (DFG), project 406907430.

Acknowledgements We would like to thank the anonymous reviewers for their very careful reading and many insightful comments that led to the improvement of this article.

1 Introduction

Motivated by applications in formal verification, automated reasoning and databases, logics and automata over *infinite* alphabets are in the focus of active and broad research activities in theoretical computer science. A typical example for logics over infinite alphabets is *constraint linear temporal logic*, CLTL for short [2, 10, 11, 9, 6, 5, 14]. CLTL extends classical LTL with a finite set of variables ranging over the domain of some infinite relational structure like $(\mathbb{N};=)$ or $(\mathbb{Q};<)$. Atomic formulas in CLTL are *constraints* in terms of the variables and the relation symbols from the structure; atomic formulas can be combined with Boolean operations and the usual temporal modalities. Models of CLTL formulas are *data words*, that is, infinite sequences of data values coming from the domain of the relational structure. For instance, the CLTL formula $\mathsf{G}(\mathsf{X}x < x)$ over the relational structure $(\mathbb{Z};<)$ states: "globally, the value of the variable x at the next position is smaller than the value of x at the current position". The data word $4, 3, 2, 1, 0, -1, -2, \ldots$ is a model of this formula. Note that the same formula has no model if instead of $(\mathbb{Z};<)$ we evaluate the formula over the relational structure $(\mathbb{N};<)$ – this to illustrate that deciding the satisfiability of a given formula in this logic heavily relies on the considered relational structure.

A natural counterpart to CLTL are *constraint automata* [7, 15, 28, 19]. Like CLTL, constraint automata are parameterized over a relational structure. Transitions are labelled with constraints in terms of the variables of the automaton and the relation symbols from the

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 76; pp. 76:1–76:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

structure; the satisfaction of the constraints along a transition determines the behaviour of the automaton. Constraint automata generalize Büchi automata and accept data languages, that is sets of data words. Following the classical automata-theoretic approach by Vardi and Wolper, one can reduce the satisfiability problem for CLTL to the non-emptiness problem for constraint automata (cf. [29, 14]). We remark that constraint automata are very much related to the well known class of register automata [17, 24, 13, 4, 8].

For CLTL, constraint automata, and other formalisms parameterized over relational structures, a lot of remarkable results concerning satisfiability, model-checking, and the emptiness problem have been achieved, see [14] for a recent survey. This includes results for specific relational structures – for instance, the satisfiability problem for CLTL over $(\mathbb{Z}; <)$ is PSPACE-complete [2, 10] – but there are also noteworthy unifying approaches that capture logics and automata over certain *classes* of, e.g., linear orders [28, 6, 5] or oligomorphic data domains [3].

In contrast to linear orders, relatively little is known about relational structures where the domain equals the set of strings A^* over some fixed (finite or countably infinite) alphabet A, and relations defined over A^*, like the *prefix order* or the *subsequence order*. While relational structures over linear orders are useful for analysing systems that manipulate counters or constrain real-timed variables, relational structures over strings are interesting for reasoning about systems that manipulate pushdown stacks, queues, or other data structures that involve strings. Reasoning on string variables has a long tradition in theoretical computer science, with roots in algebra and combinatorics on words, and recent developments in the area of string constraint solving (see [1] for a recent survey). Several works concern first-order (FO) logics over finite strings [20, 18, 16, 21]. Thereof, a recent undecidability result [16] for the Σ_1-fragment of FO logic over $(\Sigma^*; \leq_{\text{sub}}, (=w)_{w\in\Sigma^*})$, where Σ is a finite alphabet and $\leq_{\text{sub}}$ denotes the subsequence order over Σ^*, immediately implies the undecidability of the satisfiability problem for CLTL over that structure. Regarding the relational structure $(A^*; <_p, =, (=w)_{w\in A^*})$, where $<_p$ is the prefix order over A^*, we know: the satisfiability problem for constraint LTL is PSPACE-complete (by an interesting reduction to the same problem for $(\mathbb{N}; <, =, (=n)_{n\in\mathbb{N}})$) [9]. The emptiness problem for constraint automata is PSPACE-complete [19]. On the other hand, a unifying, model-theoretic approach for a large family of temporal logics, including ECTL*, which is applicable to linear orders, fails for the prefix order over finite strings [5].

Demri and Deters proposed to study the satisfiability problem for CLTL when evaluated over the structure $(A^*; <_p, <_s, =, (=w)_{w\in A^*})$ with both the prefix and the suffix order [9]. This enables us to express properties like "the beginning of the content of a string is equal to the end of some other string". Using the obvious symmetry between the prefix order and the suffix order, one can conclude that the above mentioned results for the prefix order hold for the relational structure where $<_p$ is *replaced* by $<_s$ [9]. However, the situation changes drastically when both $<_p$ and $<_s$ are in the relational structure. For instance, the FO theory on the prefix order alone is decidable [27], but becomes undecidable for the relational structure containing both prefix and suffix (this follows from the undecidability result for the FO theory for the substring (infix) order [20], and the fact that the substring order is FO-definable using prefix and suffix). For finite strings over a finite alphabet, it has been remarked in [9] that the Σ_1-fragment of FO logics is decidable, using an algorithm based on the word equation approach by Makanin [22, 26]. It is thus far from clear whether satisfiability for CLTL, or, equivalently, the emptiness problem for constraint automata, is decidable or not. The techniques used in other works for, e.g. the prefix order alone, or linear orders, turn out to be not applicable at all.

In this paper, we prove that the emptiness problem for constraint automata over prefix and suffix is decidable in NL if the automaton uses only a single variable. This is a standard restriction, comparable to one-counter automata [12] or single-clock timed automata [25]. Our decision procedure relies on a reduction to reachability queries on the finite graph underlying the automaton, and it applies to finite strings over both finite and countably infinite alphabets. We may also test whether the string equals the empty string (similar to a zero test in one-counter automata). We further obtain NL-completeness for the emptiness problem for single-register automata over this relational structure. Last but not least, our result implies PSPACE-completeness for the satisfiability of CLTL for the case that the formulas in CLTL only use a single variable.

We leave open the decidability status for the case where equality with arbitrary finite strings over A and/or constraints involving more than one variable are allowed, that is, by now we cannot fully answer the question raised by Demri and Deters. We remark that both extensions may be harmful: for instance, while emptiness is decidable for one-counter automata [12], it is undecidable for two-counter automata [23]; while the Σ_1-fragment of FO logic for finite strings over a finite alphabet with the subsequence order *without constants* is decidable [20], it is undecidable as soon as we allow constants in the relational structure [16].

2 Preliminaries

A *relational signature* $\sigma = \{R_1, R_2, \dots\}$ is a countable set of relation symbols. Each symbol R_i is associated with some non-negative arity k_i. A *relational structure over* σ, or σ*-structure* for short, is a tuple $\mathcal{D} = (D; R_1^{\mathcal{D}}, R_2^{\mathcal{D}}, \dots)$, where D is the domain of the structure, and $R_i^{\mathcal{D}} \subseteq D^{k_i}$ is the interpretation of the symbol R_i in D. We will often omit the symbol $\mathcal{D}$ in $R_i^{\mathcal{D}}$ and simply write R_i instead.

We use Σ to denote a finite alphabet, $\mathbb{N}$ as a countably infinite alphabet, and A as finite or countably infinite alphabet. We use A^* to denote the set of finite strings over A. The symbol ε denotes the empty string, and we use A^+ to denote the set $A^* \setminus \{\varepsilon\}$ of non-empty strings over A. Given $u, v \in A^*$, we say that u is a *strict prefix* (*strict suffix*, respectively) of v, written $u <_p v$ ($u <_s v$, respectively), if $v = u \cdot u'$ ($v = u' \cdot u$, respectively) for some $u' \in A^+$. We say that u and v are *incomparable with respect to the prefix order*, written $u \perp_p v$, if $u = w \cdot a \cdot u'$ and $v = w \cdot b \cdot v'$ for some $w \in A^*$, $a, b \in A$ such that $a \neq b$, and $u', v' \in A^*$. *Incomparability with respect to the suffix order*, written $u \perp_s v$, is defined analogously.

Let σ^{ps} be the signature consisting of the binary symbols $<_p$, $<_s$, and $=$. In this paper, we are interested in the σ^{ps}-structures $(\Sigma^*; <_p, <_s, =)$ and $(\mathbb{N}^*; <_p, <_s, =)$, where, in both structures, $<_p$ and $<_s$ are interpreted as the prefix and the suffix order over the set of strings over Σ and $\mathbb{N}$, respectively, and $=$ is interpreted as the identity. If the context is clear, we may write Σ^* and $\mathbb{N}^*$ to denote the respective structures, and A^* to denote any of these structures.

Constraint automata are generalizations of Büchi automata that are parameterized by σ-structures, where σ is a relational signature. The transitions are labelled with Boolean combinations of atomic formulas, called *constraints*, in terms of the relations of the σ-structure. A constraint automaton processes *data words*. A data word is a finite or infinite sequence $d_1, d_2, d_3 \dots$, where $d_i \in D$ is a data value from the domain of the σ-structure. A transition of a constraint automaton can be taken if the current and the next data value of the processed data word satisfy the constraint labelling the transition.

In the following, we assume that constraint automata are parameterized by the σ^{ps}-structures Σ^* or $\mathbb{N}^*$, and the transitions are labelled by Boolean combinations of atomic formulas of the form $z \bowtie z'$, where $z, z' \in \{x, y\}$ and $\bowtie \in \sigma^{ps}$. Intuitively, x stands for

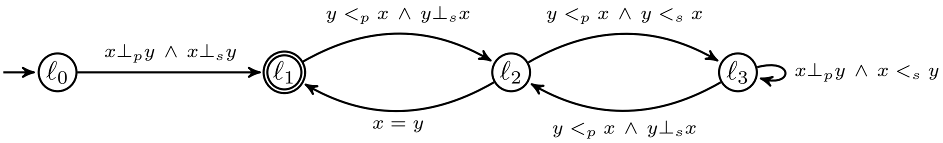

ℓ_0
ℓ_1
ℓ_2
ℓ_3
$x\perp_p y \wedge x\perp_s y$
$y <_p x \wedge y\perp_s x$
$x = y$
$y <_p x \wedge y <_s x$
$y <_p x \wedge y\perp_s x$
$x\perp_p y \wedge x <_s y$

Figure 1 Example of a constraint automaton over A^*

the string at the current position, and y stands for the string at the next position of the processed data word. For the sake of readability we will restrict the labels of the transitions to be *maximally consistent*. Formally, we define Ψ to be the set of formulas $\psi(x, y)$ of the following form:

$$\begin{array}{cccc} x = y, & x <_p y \wedge x <_s y, & x <_p y \wedge x\perp_s y, & x\perp_p y \wedge x <_s y \\ y <_p x \wedge y <_s x, & y <_p x \wedge y\perp_s x, & y\perp_p x \wedge y <_s x, & x\perp_p y \wedge x\perp_s y. \end{array}$$

Each of these formulas is called *constraint*. Constraints that contain the formula $y <_p x$ or $y <_s x$ are called *reducing* as they reduce the length of the string at the next position with regard to the string at the current position. All other constraints except for $x = y$ are called *generous*, because they allow for infinitely many choices of the string value at the next position. The satisfaction relation $\models$ is defined in the obvious way. For instance, if ψ is of the form $x <_p y \wedge x <_s y$, then we have $\mathbb{N}^* \models \psi(0, 01210)$ and $\Sigma^* \not\models \psi(a, abab)$.

A *constraint automaton* over a σ^{ps}-structure A^* is a tuple $\mathcal{A} = (\mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$, where

- $\mathcal{L}$ is a finite set of locations (control states);
- $\ell_{\text{in}} \in \mathcal{L}$ is the initial location;
- $\mathcal{L}_{\text{acc}} \subseteq \mathcal{L}$ is the set of accepting locations; and
- $E \subseteq \mathcal{L} \times \Psi \times \mathcal{L}$ is the set of edges.

A *path* of $\mathcal{A}$ is a finite or infinite sequence $\ell_0, \ell_1, \ell_2, \ldots$ of locations satisfying, for all $i \geq 1$, $(\ell_{i-1}, \psi_i, \ell_i) \in E$ for some constraint $\psi_i \in \Psi$. We may sometimes also write $\ell_0 \xrightarrow{\psi_1} \ell_1 \xrightarrow{\psi_2} \ell_2, \ldots$ to indicate the precise edges that are used. A finite path $\ell_0 \xrightarrow{\psi_1} \ell_1 \xrightarrow{\psi_2} \ell_2 \ldots \xrightarrow{\psi_n} \ell_n$ is *stable* if ψ_i is of the form $x = y$ for all $1 \leq i \leq n$; it is *generous* if there exists some $1 \leq i \leq n$ such that ψ_i is generous. A path as above is a *cycle starting in* ℓ_0 if $\ell_0 = \ell_n$.

A *state* of $\mathcal{A}$ is a pair (ℓ, w), where $\ell \in \mathcal{L}$ and $w \in A^*$ is the current value of the string variable. We postulate a labelled transition relation $\rightarrow$ over the set $\mathcal{L} \times A^*$ of states of $\mathcal{A}$, as follows: $(\ell, w) \rightarrow (\ell, w')$ if there exists a transition $(\ell, \psi(x, y), \ell') \in E$ such that $A^* \models \psi(w, w')$. A *run* of $\mathcal{A}$ is a finite or infinite sequence of transitions of $\mathcal{A}$. A run $(\ell_0, w_0) \rightarrow (\ell_1, w_1) \rightarrow (\ell_2, w_2) \ldots$ is *initialized* if $\ell_0 = \ell_{\text{in}}$. A run is *Büchi-accepting* if it is initialized and it contains infinitely many locations in $\mathcal{L}_{\text{acc}}$. We define the language of $\mathcal{A}$ by $L(\mathcal{A}) = \{(w_0 w_1 w_2 \cdots \mid (\ell_0, w_0) \rightarrow (\ell_1, w_1) \rightarrow (\ell_2, w_2) \ldots$ is a Büchi-accepting run of $\mathcal{A}\}$.

For an example, consider the constraint automaton $\mathcal{A} = (\{\ell_0, \ell_1, \ell_2, \ell_3\}, \ell_0, E, \{\ell_1\})$ over $\mathbb{N}^*$, where E is as depicted in Figure 1. A finite initialized run of this automaton is $(\ell_0, 20) \rightarrow (\ell_1, 346345346343) \rightarrow (\ell_2, 34634534634) \rightarrow (\ell_3, 34634) \rightarrow (\ell_2, 346)$ (cf. Example 6).

The *emptiness problem* for constraint automata is to decide, given a constraint automaton $\mathcal{A}$, whether $L(\mathcal{A}) = \emptyset$. In section 4, we prove that this problem is NL-complete.

3 Rewriting Operation

For deciding the emptiness problem, we will prove the existence of string values that satisfy the constraints that occur in a given path. During the process of defining such string values, we will need to change already defined string values using a *rewriting operation*. In this section, we define this operation and prove some important properties.

Recall that A denotes a finite or countably infinite alphabet. Given $w \in A^*$ and two non-empty strings $u, u' \in A^+$, we define the *left-to-right rewriting operation of u to u' in w*, denoted by $w[u \leftarrow u']_\triangleright$, to be the string that is obtained from w by replacing, from left to right, every occurrence of u in w by u'. Formally, assume $w = a_1 a_2 \ldots a_n$ and define, recursively, $w[u \leftarrow u']_\triangleright := w$ if $a_i a_{i+1} \ldots a_{i+|u|-1} \neq u$ for all $1 \leq i \leq n$ (that is, u does not occur in w), $w[u \leftarrow u']_\triangleright := a_1 \ldots a_{i-1} \cdot u' \cdot ((a_{i+|u|} \ldots a_n)[u \leftarrow u']_\triangleright)$ if $1 \leq i \leq n$ is the minimal index such that $a_i a_{i+1} \ldots a_{i+|u|-1} = u$. Note that if u occurs in $a_1 \ldots a_{i-1} \cdot u'$, then u is not replaced in any further steps of the recursive definition. For instance, $1100210[10 \leftarrow 1]_\triangleright = 11021$. We define completely analogously the right-to-left version of this operation, that is, $w[u \leftarrow v]_\triangleleft := w$ if u does not occur in w, and $w[u \leftarrow v]_\triangleleft := ((a_1 \ldots a_{i-1})[u \leftarrow u']_\triangleright) \cdot u' \cdot a_{i+|u|} \ldots a_n$, if $i \geq 1$ is the maximal index such that $a_i \ldots a_{i+|u|-1} = u$. Note that $w[u \leftarrow v]_\triangleright$ may be different from $w[u \leftarrow v]_\triangleleft$; for instance, $w = 111$, $u = 11$ and $v = 0$ yields $w[u \leftarrow v]_\triangleright = 01$ and $w[u \leftarrow v]_\triangleleft = 10$. It is easy to see that this is the case if there exist two overlapping occurrences of u in w. Formally, we say that *u is overlapping in w* if there exist $1 \leq i < j < i + |u| \leq n$ such that $a_i a_{i+1} \ldots a_{i+|u|-1} = a_j a_{j+1} \ldots a_{j+|u|-1} = u$. The proof of the following lemma is simple.

▶ **Lemma 1.** *For all $w \in A^*$ and $u, u' \in A^+$, if u is not overlapping in w, then we have $w[u \leftarrow u']_\triangleright = w[u \leftarrow u']_\triangleleft$.*

In Subsection 4.1 we will guarantee that the rewriting operation is only applied to strings w and u such that u is not overlapping in w, so that the left and right versions of rewriting yield the same string. The reason why we still define both the left and right version of rewriting is that certain properties of the prefix order – stated in the next lemma – can be proved very conveniently using the left-to-right rewriting operation, and the same properties can be proved symmetrically for the suffix order using the right-to-left rewriting operation (Lemma 3).

▶ **Lemma 2.** *For all $u, u' \in A^+$ with $u <_p u'$, for all $w, w' \in A^*$, and for all $\bowtie \in \{=, <_p, \perp_p\}$ we have*

$$w \bowtie w' \iff w[u \leftarrow u']_\triangleright \bowtie w'[u \leftarrow u']_\triangleright.$$

Proof. Let $u, u' \in A^+$ be such that $u <_p u'$, that is, there exists some $u'' \in A^+$ such that $u' = u \cdot u''$. Let $w, w' \in A^*$ be of the form $w = a_1 a_2 \ldots a_m$ and $w' = a'_1 a'_2 \ldots a'_n$. Let N and N', respectively, be the number of (non-overlapping, from left to right) occurrences of u in w and w', respectively. The proof is by induction on the sum $i := N + N'$. For the induction base, assume $i = 0$. But then $w[u \leftarrow u']_\triangleright = w$ and $w'[u \leftarrow u']_\triangleright = w'$, so that the claim clearly holds. For the induction step, suppose that the claim holds for all $0 \leq j < i$. We prove the claim for i. We distinguish three cases:

1. $N = i$ and $N' = 0$. By $N = i > 0$, w contains u. Since $u <_p u'$, also $w[u \leftarrow u']_\triangleright$ contains u. By $N' = 0$, w' does not contain u and $w'[u \leftarrow u']_\triangleright = w'$, so that $w'[u \leftarrow u']_\triangleright$ does not contain u either. Using this, it is easy to see that none of the following cases can hold: $w = w'$, $w <_p w'$, $w[u \leftarrow u']_\triangleright = w'[u \leftarrow u']_\triangleright$ and $w[u \leftarrow u']_\triangleright <_p w'[u \leftarrow u']_\triangleright$. So let us prove $w \perp_p w' \iff w[u \leftarrow u']_\triangleright \perp_p w'$. For this suppose $w[u \leftarrow u']_\triangleright$ is of the form

$b_1 b_2 \dots b_q$. Let $1 \leq d \leq e \leq m$ be such that $a_d \dots a_e = u$ is the *first* occurrence of u in w. By $u <_p u'$, $b_d \dots b_e = u$ is also the first occurrence of u in $w[u \leftarrow u']_{\triangleright}$, and hence $a_1 \dots a_e = b_1 \dots b_e$. We hence obtain

$$\begin{aligned} & w \perp_p w' \\ \iff \quad & \exists k \leq e \text{ such that } a_1 \dots a_{k-1} = a'_1 \dots a'_{k-1} \text{ and } a_k \neq a'_k \\ \iff \quad & \exists k \leq e \text{ such that } b_1 \dots b_{k-1} = a'_1 \dots a'_{k-1} \text{ and } b_k \neq a'_k \\ \iff \quad & w[u \leftarrow u']_{\triangleright} \perp_p w' \end{aligned}$$

where $k \leq e$ holds by the fact that u is not contained in w'.

2. $N = 0$ and $N' = i$. By $N = 0$, w does not contain u and hence neither does $w[u \leftarrow u']_{\triangleright}$ as $w[u \leftarrow u']_{\triangleright} = w$. By $N' = i > 0$, w' contains u. Using this, it is easy to see that the following two cases cannot hold: $w = w'$ and $w[u \leftarrow u']_{\triangleright} = w'[u \leftarrow u']_{\triangleright}$. The proof for $w \perp_p w' \iff w \perp_p w'[u \leftarrow u']_{\triangleright}$ is symmetric to the proof in the previous case. So let us prove $w <_p w' \iff w <_p w'[u \leftarrow u']_{\triangleright}$. For this let $w'[u \leftarrow u']_{\triangleright}$ be of the form $b'_1 \dots b'_q$. Let $1 \leq d \leq e \leq n$ be such that $a'_d \dots a'_e = u$ is the *first* occurrence of u in w'. By $u <_p u'$, $b'_d \dots b'_e = u$ is also the first occurrence of u in $w'[u \leftarrow u']_{\triangleright}$, and hence $a'_1 \dots a'_e = b'_1 \dots b'_e$. We hence obtain

$$\begin{aligned} & w <_p w' \\ \iff \quad & a_1 \dots a_m = a'_1 \dots a'_m \\ \iff \quad & a_1 \dots a_m = b'_1 \dots b'_m \\ \iff \quad & w <_p w'[u \leftarrow u']_{\triangleright} \end{aligned}$$

where the second equivalence holds because $m < e$ (as otherwise u would be contained in w).

3. $N > 0$ and $N' > 0$. Let $1 \leq d \leq e \leq m$ be such that $a_d \dots a_e = u$ is the first occurrence of u in w, and similarly, let $1 \leq d' \leq e' \leq n$ be such that $a'_{d'} \dots a'_{e'} = u$ is the first occurrence of u in w'. In other words, we can write

$$w = a_1 \dots a_{d-1} \cdot u \cdot v \quad \text{and} \quad w' = a'_1 \dots a'_{d'-1} \cdot u \cdot v',$$

where $v = a_e \dots a_m$ and $v' = a'_{e'} \dots a'_n$. By definition and $u' = u \cdot u''$, we have

$$w[u \leftarrow u']_{\triangleright} = a_1 \dots a_{d-1} \cdot u \cdot u'' \cdot (v[u \leftarrow u']_{\triangleright})$$

and

$$w'[u \leftarrow u']_{\triangleright} = a'_1 \dots a'_{d'-1} \cdot u \cdot u'' \cdot (v'[u \leftarrow u']_{\triangleright}).$$

We distinguish four cases:

a. $a_1 \dots a_{d-1} \cdot u = a'_1 \dots a'_{d'-1} \cdot u$. This implies

$$w \bowtie w' \iff v \bowtie v' \quad \text{and} \quad w[u \leftarrow u']_{\triangleright} \bowtie w'[u \leftarrow u']_{\triangleright} \iff v[u \leftarrow u']_{\triangleright} \bowtie v'[u \leftarrow u']_{\triangleright}$$

for $\bowtie \in \{<_p, =, \perp_p\}$. The sum $M + M'$ of the occurrences of u in v and v' must be strictly smaller than i. By induction hypothesis,

$$v \bowtie v' \iff v[u \leftarrow u']_{\triangleright} \bowtie v'[u \leftarrow u']_{\triangleright}.$$

Hence the result.

b. $a_1 \dots a_{d-1} \cdot u <_p a'_1 \dots a'_{d'-1} \cdot u$. This contradicts the minimality of d', and hence this case cannot happen.

c. $a'_1 \dots a'_{d'-1} \cdot u <_p a_1 \dots a_{d-1} \cdot u$. This contradicts the minimality of d, and hence this case cannot happen.

d. $a_1 \dots a_{d-1} \cdot u \perp_p a'_1 \dots a'_{d'-1} \cdot u$. Hence there exists some $1 \leq j \leq \min(d-1, d'-1)$ such that $a_1 \dots a_{j-1} = a'_1 \dots a'_{j-1}$ and $a_j \neq a'_j$. This immediately implies $w \perp_p w'$ and also $w[u \leftarrow u']_\triangleright \perp_p w'[u \leftarrow u']_\triangleright$, hence the result. ◀

A proof for the following lemma can be done symmetrically to the proof of Lemma 2.

▶ **Lemma 3.** *For all $u, u' \in A^+$ and $w, w' \in A^*$, if $u <_s u'$, then $w \bowtie w'$ if, and only if, $w[u \leftarrow u']_\triangleleft \bowtie w'[u \leftarrow u']_\triangleleft$ for all $\bowtie \in \{=, <_s, \perp_s\}$.*

The following lemma will be crucial in Subsection 4.1.

▶ **Lemma 4.** *For all $v, w \in A^*$ and $u, u' \in A^+$, if u is not overlapping in v, u is not overlapping in w, $u <_p u'$, and $u <_s u'$, then $A^* \models \psi(v, w)$ if, and only if, $A^* \models \psi(v[u \leftarrow u']_\triangleright, w[u \leftarrow u']_\triangleright)$ for all $\psi \in \Psi$.*

Proof. The proof is an easy case distinction depending on the form of ψ. We give the proof for ψ being of the form $x <_p y \wedge x \perp_s y$.

$$\begin{aligned}
& \mathbb{N}^* \models \psi(v, w) \\
\iff & v <_p w \text{ and } v \perp_s w \quad \text{(by definition)} \\
\iff & v[u \leftarrow u']_\triangleright <_p w[u \leftarrow u']_\triangleright \text{ and } v \perp_s w \quad \text{(by Lemma 2)} \\
\iff & v[u \leftarrow u']_\triangleright <_p w[u \leftarrow u']_\triangleright \text{ and } v[u \leftarrow u']_\triangleleft \perp_s w[u \leftarrow u']_\triangleleft \quad \text{(by Lemma 3)} \\
\iff & v[u \leftarrow u']_\triangleright <_p w[u \leftarrow u']_\triangleright \text{ and } v[u \leftarrow u']_\triangleright \perp_s w[u \leftarrow u']_\triangleright \quad \text{(by Lemma 1)} \\
\iff & \mathbb{N}^* \models \psi(v[u \leftarrow u']_\triangleright, w[u \leftarrow u']_\triangleright) \quad \text{(by definition)}
\end{aligned}$$

The proofs for the other cases are completely analogous. ◀

4 Deciding Emptiness for Constraint Automata over $\mathbb{N}^*$

In this section, we solve the emptiness problem for constraint automata over $\mathbb{N}^*$. We start in the next subsection with presenting an algorithm that returns for every *finite* sequence of constraints $\psi_1, \dots, \psi_n$ a sequence of string values $w_0, w_1, \dots, w_n$ such that $\mathbb{N}^* \models \psi_i(w_{i-1}, w_i)$ for all $1 \leq i \leq n$. The sequence $\psi_1, \dots, \psi_n$ may correspond to the sequence of constraints occurring in a finite path $\pi = \ell_0 \xrightarrow{\psi_1} \dots \xrightarrow{\psi_n} \ell_n$ of a constraint automaton $\mathcal{A}$, and by constructing string values $w_0, w_1, \dots, w_n$ we actually prove that π can be completed to a finite run $(\ell_0, w_0) \xrightarrow{\psi_1} \dots \xrightarrow{\psi_n} (\ell_n, w_n)$ of $\mathcal{A}$. This already implies NL-membership of the reachability problem for constraint automata (Corollary 8).

We remark that for *infinite* sequences of constraints $\psi_1, \psi_2, \dots$ it is not the case that we can always find string values $w_0, w_1, \dots$ satisfying $\mathbb{N}^* \models \psi_i(w_{i-1}, w_i)$. Consider for instance the sequence $(x \perp_p y \wedge x \perp_s y)\,(y <_p x \wedge y \perp_s x)^\omega$ (cf. Figure 1). The constraint $y <_p x \wedge y \perp_s x$ is reducing and requires the strings in the ω-sequence to become shorter infinitely often, which is impossible. In Subsection 4.2 we give a characterization for when infinite paths can be extended to infinite runs, which, together with the results obtained before, yields a decision procedure for the emptiness problem.

For the rest of this section, let $\mathcal{A} = (\mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$ be a constraint automaton over $\mathbb{N}^*$.

4.1 Extending Finite Paths to Finite Runs

The main part of this subsection is dedicated to prove the following result.

▶ **Proposition 5.** *For every sequence $\psi_1, \ldots, \psi_n$ of constraints in Ψ and non-empty string $w_{\text{in}} \in \mathbb{N}^+$, we can define non-empty strings $w_0, w_1, \ldots, w_n \in \mathbb{N}^+$ such that $\mathbb{N}^* \models \psi_i(w_{i-1}, w_i)$ for all $1 \leq i \leq n$. Moreover, if ψ_1 is generous, then $w_0 = w_{\text{in}}$.*

Let $\psi_1, \ldots, \psi_n$ be a finite sequence of constraints in Ψ, and let w_{in} be an initial non-empty string value. The idea is to construct, one after the other, string values that satisfy the constraints. During the process, formerly defined string values may need to be rewritten using the left-to-right-rewriting operation defined in Section 3 (so that w_0 may not be equal to the input string w_{in}). We take advantage of the fact that we have an unbounded supply of "fresh" letters as we operate on the infinite alphabet $\mathbb{N}$: we can assign string values in such a way that constraints that are satisfied *before* a rewriting still hold true *after* a rewriting. Given a string $w \in \mathbb{N}^*$, we let $\max(w)$ be the maximal number occurring as a letter in w if $w \neq \varepsilon$ and $\max(w) = 0$ otherwise.

For $1 \leq i \leq n$, suppose we have already defined string values $w_0^{i-1}, w_1^{i-1}, \ldots, w_{i-1}^{i-1}$ such that $\mathbb{N}^* \models \psi_j(w_{j-1}^{i-1}, w_j^{i-1})$ for all $1 \leq j < i$, where $w_0^0 = w_{\text{in}}$. Define $M_i = \max\{w_0^{i-1}, \ldots, w_{i-1}^{i-1}\} + 1$, so that M_i is a "fresh" letter not occurring in any of the already defined string values. Depending on the form of ψ_i, we define $w_0^i, w_1^i, \ldots, w_i^i$ such that $\mathbb{N}^* \models \psi_j(w_{j-1}^i, w_j^i)$ for all $1 \leq j \leq i$. We consider the following cases:

1. ψ_i is of the form $x = y$. Define $w_i^i = w_{i-1}^{i-1}$, and $w_j^i = w_j^{i-1}$ for all $0 \leq j < i$.
2. ψ_i is of the form $x <_p y \wedge x <_s y$. Define $w_i^i = w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}$, and $w_j^i = w_j^{i-1}$ for all $0 \leq j < i$.
3. ψ_i is of the form $x <_p y \wedge x \bot_s y$. Define $w_i^i = w_{i-1}^{i-1} \cdot M_i$, and $w_j^i = w_j^{i-1}$ for all $0 \leq j < i$.
4. ψ_i is of the form $x \bot_p y \wedge x <_s y$. Define $w_i^i = M_i \cdot w_{i-1}^{i-1}$, and $w_j^i = w_j^{i-1}$ for all $0 \leq j < i$.
5. ψ_i is of the form $y <_p x \wedge y <_s x$. Define $w_i^i = w_{i-1}^{i-1}$, and for all $0 \leq j < i$ define $w_j^i = w_j^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]_\triangleright$.
6. ψ_i is of the form $y <_p x \wedge y \bot_s x$. Define $w_i^i = w_{i-1}^{i-1} \cdot M_i$, and for all $0 \leq j < i$ define $w_j^i = w_j^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]_\triangleright$.
7. ψ_i is of the form $y \bot_p x \wedge y <_s x$. Define $w_i^i = M_i \cdot w_{i-1}^{i-1}$, and for all $0 \leq j < i$ define $w_j^i = w_j^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]_\triangleright$.
8. ψ_i is of the form $x \bot_p y \wedge x \bot_s y$. Define $w_i^i = M_i$ and $w_j^i = w_j^{i-1}$ for all $0 \leq j < i$.

▶ **Example 6.** Let us illustrate the construction with the sequence ψ_1, ψ_2, ψ_3 and $w_{\text{in}} = 3$, where $\psi_1 = \psi_3 = (y <_p x \wedge y \bot_s x)$, and $\psi_2 = (y <_p x \wedge y <_s x)$. For $i = 1$, we are in case **6**. We have $M_1 = 4$ and obtain $w_1^1 = w_0^0 \cdot M_1 = 34$ and $w_0^1 = w_0^0[3 \leftarrow 343]_\triangleright = 343$. Clearly $\mathbb{N}^* \models \psi_1(w_0^1, w_1^1)$. For $i = 2$, we are in case **5** and define $w_2^2 = w_1^1 = 34$. We further rewrite $w_1^2 = w_1^1[34 \leftarrow 34534]_\triangleright = 34534$, and $w_0^2 = w_0^1[34 \leftarrow 34534]_\triangleright = 345343$. So even after rewriting, we have $\mathbb{N}^* \models \psi_i(w_{i-1}^2, w_i^2)$ for $i = 1, 2$. For $i = 3$, we are again in case **6**. We obtain $w_3^3 = 346$; w_2^3, w_1^3 and w_0^3, respectively, are rewritten to 34634, 34634534634 and 346345346343, respectively. All constraints are indeed satisfied.

Let us state an important property of the construction, which will be key for the correctness of the construction.

▶ **Invariant 7.** *For every $0 \leq i \leq n$ and every $0 \leq j \leq i$, w_i^i is not overlapping in w_j^i.*

Proof. The proof is by induction on i. The induction base, $i = 0$, is trivial. So assume that the claim holds for all $0 \leq k < i$. We prove it for i. We consider different cases, based on the form of ψ_i. Let $0 \leq j < i$ (the case $j = i$ is trivial).

1. Suppose we are in cases **1**, **2**, **3**, **4**, or **8**, that is, $w_j^i = w_j^{i-1}$ (no rewriting happens). In case **1**, $w_i^i = w_{i-1}^{i-1}$, we can apply the induction hypothesis to obtain the result. In the remaining four cases, the string w_i^i contains the letter M_i, which, by definition, does not occur in w_j^{i-1}. Hence w_i^i cannot occur at all in w_j^{i-1}.
2. Suppose we are in cases **5**, **6**, or **7**, that is, $w_j^i = w_j^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]$ (rewriting happens). By induction hypothesis, w_{i-1}^{i-1} is not overlapping in w_j^{i-1}. If N is the number of occurrences of w_{i-1}^{i-1} in w_j^{i-1}, we can hence write

   $$w_j^{i-1} = u_0 \cdot w_{i-1}^{i-1} \cdot u_1 \cdot w_{i-1}^{i-1} \cdot u_2 \ldots u_{N-1} \cdot w_{i-1}^{i-1} \cdot u_N$$

   for some $u_0, \ldots, u_N \in \mathbb{N}^*$. By definition,

   $$w_j^i = u_0 \cdot w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1} \cdot u_1 \cdot w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1} \cdot u_2 \ldots u_{N-1} \cdot w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1} \cdot u_N.$$

   In case **5**, $w_i^i = w_{i-1}^{i-1}$, so that w_i^i does not contain M_i by definition. The only way for w_i^i to be overlapping in w_j^i is in $u_0 \cdot w_{i-1}^{i-1}$, in $w_{i-1}^{i-1} \cdot u_k \cdot w_{i-1}^{i-1}$ for some $1 \le k < N$, or in $w_{i-1}^{i-1} \cdot u_N$. But this would contradict that w_{i-1}^{i-1} is not overlapping in w_j^{i-1}. In case **6**, $w_i^i = w_{i-1}^{i-1} \cdot M_i$. By definition, w_j^{i-1} does not contain M_i. Hence the only way for w_i^i to be contained at all in w_j^i is so that no overlap can occur. The reasoning for case **7**, where $w_i^i = M_i \cdot w_{i-1}^{i-1}$, is analogous. ◀

Let us finally prove the correctness of the construction, that is, for all $1 \le i \le n$, we have $\mathbb{N}^* \models \psi_j(w_{j-1}^i, w_j^i)$ for all $1 \le j \le i$. The proof is by induction on i. For the base case $i = 1$ observe that w_0^1 and w_1^1 are defined such that $\mathbb{N}^* \models \psi_1(w_0^1, w_1^1)$. So suppose that the claim holds for all $1 \le k < i$. We prove it for i. For $j = i$, it is again easy to see that $\mathbb{N}^* \models \psi_i(w_{i-1}^i, w_i^i)$. So let $1 \le j < i$. By induction hypothesis, we have $\mathbb{N}^* \models \psi_j(w_{j-1}^{i-1}, w_j^{i-1})$. Depending on the form of ψ_i, we either have

- $w_{j-1}^i = w_{j-1}^{i-1}$ and $w_j^i = w_j^{i-1}$, so that we have $\mathbb{N}^* \models \psi_j(w_{j-1}^i, w_j^i)$; or
- $w_{j-1}^i = w_{j-1}^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]_\triangleright$ and $w_j^i = w_j^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]_\triangleright$. By Invariant 7, w_{i-1}^{i-1} is not overlapping in w_{j-1}^{i-1}, w_{i-1}^{i-1} is not overlapping in w_j^{i-1}, and $w_{i-1}^{i-1} <_p w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}$, and $w_{i-1}^{i-1} <_s w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}$. By Lemma 4 we obtain $\mathbb{N}^* \models \psi_j(w_{j-1}^i, w_j^i)$.

Setting $w_i = w_i^n$ for all $0 \le i \le n$, we are done with the proof of the first claim of Proposition 5.

Let us prove the second claim and suppose that ψ_1 is generous. We prove below that for all $1 \le i \le n$, w_i^i contains some letter not occurring in w_0^i. Note that this implies $w_0^0 = w_0^1 = \cdots = w_0^n$. The proof is by induction on i. For the induction base, set $i = 1$. Since ψ_1 is generous, we are in one of the cases **2**, **3**, **4**, or **8**. Here, w_1^1 contains M_1, which, by definition, is not occurring in w_0^0, and $w_0^1 = w_0^0$. Hence w_1^1 contains a letter not occurring in w_0^1. For the induction step, suppose the claim holds for all $1 \le j < i$. We prove it for i. Note that depending on the form of ψ_i, w_i^i is defined as w_{i-1}^{i-1}, a fresh letter M_i, or a composition of these two. For w_0^i, we either have $w_0^i = w_0^{i-1}$, in which case the induction hypothesis and/or freshness of M_i immediately establishes the claim, or we have $w_0^i = w_0^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}]$. But by induction hypothesis, w_{i-1}^{i-1} contains some letter not occurring in w_0^{i-1}, so that w_{i-1}^{i-1} cannot occur in w_0^{i-1} and thus $w_0^{i-1}[w_{i-1}^{i-1} \leftarrow w_{i-1}^{i-1} \cdot M_i \cdot w_{i-1}^{i-1}] = w_0^{i-1}$. Hence w_i^i contains some letter not occurring in w_0^i. This finishes the proof of Proposition 5.

The *(control-state) reachability problem for constraint automata* is the problem to decide, given a constraint automaton $\mathcal{A} = (\mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$ over $\mathbb{N}^*$ and some target location $\ell \in \mathcal{L}$, whether there exists a run from (ℓ_{in}, w_0) to (ℓ, w), for some $w_0, w \in \mathbb{N}^*$.

▶ **Corollary 8.** *The reachability problem for constraint automata is NL-complete.*

Proof. For the upper bound, it suffices to decide whether there exists a path from ℓ_{in} to ℓ, which can be done in NL. If no such path exists, then there exists no run from (ℓ_{in}, w_0) to (ℓ, w) for some $w_0, w \in \mathbb{N}^*$. If such a path, say $\ell_0 \xrightarrow{\psi_1} \dots \xrightarrow{\psi_n} \ell_n$, exists, we use Proposition 5 with $\psi_1, \dots, \psi_n$ and some $w_{\text{in}} \in \mathbb{N}^+$ to obtain $w_0, w_1, \dots, w_n \in \mathbb{N}^+$ such that $\mathbb{N}^* \models \psi_i(w_{i-1}, w_i)$ for all $1 \leq i \leq n$. Then $(\ell_0, w_0) \xrightarrow{\psi_1} \dots \xrightarrow{\psi_n} (\ell_n, w_n)$ is a finite run of $\mathcal{A}$. A reduction from the reachability problem for finite directed graphs yields the lower bound. ◀

4.2 Characterization for Büchi-accepting Runs

As mentioned above, there are infinite sequences of constraints $\psi_1, \psi_2, \dots$ for which it may not be possible to find $w_0, w_1, w_2 \dots$ such that $\mathbb{N}^* \models \psi(w_{i-1}, w_i)$ for all $i \geq 1$. In the following proposition, we give a decidable characterization for when an infinite path of $\mathcal{A}$ can be completed with string values to obtain an infinite run of $\mathcal{A}$.

▶ **Proposition 9.** *The following three statements are equivalent:*

1. *There exists a Büchi-accepting run of $\mathcal{A}$.*

2. *There exists an infinite path π of $\mathcal{A}$ satisfying the following conditions:*

  a. *π starts in ℓ_{in},*

  b. *π contains infinitely many occurrences of ℓ_{acc}, for some $\ell_{\text{acc}} \in \mathcal{L}_{\text{acc}}$, and*

  c. *if π contains only finitely many generous constraints, then π contains only finitely many reducing constraints.*

3. *There exists a path from ℓ_{in} to ℓ_{acc}, for some $\ell_{\text{acc}} \in \mathcal{L}_{\text{acc}}$, and one of the following holds:*

  a. *there exists some stable cycle starting in ℓ_{acc}, or*

  b. *there exists some generous cycle starting in ℓ_{acc}.*

Proof. For the proof from **1.** to **2.**, let $(\ell_0, w_0) \xrightarrow{\psi_1} (\ell_1, w_1) \xrightarrow{\psi_2} \dots$ be a Büchi-accepting run of $\mathcal{A}$. Define π to be the infinite path $\ell_0 \xrightarrow{\psi_1} \ell_1 \xrightarrow{\psi_2} \dots$. Clearly, π satisfies conditions **2.a** and **2.b**; we prove that condition **2.c** also holds. Towards contradiction, suppose that π contains finitely many generous constraints but infinitely many reducing constraints. Then there exists some $i \geq 1$ such that ψ_j is reducing or of the form $x = y$, for all $j \geq i$. Note that this implies $|w_j| \geq |w_{j+1}|$ for all $j \geq i$. Moreover, since there are infinitely many reducing constraints, there exists an infinite sequence $i \leq i_1 < i_2 < i_3 \dots$ of indices such that ψ_{i_j} is reducing and hence $|w_{i_j}| > |w_{i_j+1}|$. Since $|w_j|$ is finite, this leads to a contradiction.

For the proof from **2.** to **1.**, let $\pi = \ell_0 \xrightarrow{\psi_1} \ell_1 \xrightarrow{\psi_2} \dots$ be an infinite path of $\mathcal{A}$ satisfying the three conditions stated in **2.** Using condition **2.c**, we prove that we can complete π with string values to yield an infinite run of $\mathcal{A}$; that this run is Büchi-accepting, follows by conditions **2.a** and **2.b**. We distinguish two cases.

- Suppose π contains only finitely many generous constraints. By condition **2.c**, π contains only finitely many reducing constraints. Then there exists some $i \geq 0$ such that ψ_j is of the form $x = y$ for all $j > i$. Use Proposition 5 with the sequence $\psi_1, \dots, \psi_i$ and $w_{\text{in}} = 0$ to obtain string values $w_0, w_1, \dots, w_i \in \mathbb{N}^+$ such that $\mathbb{N}^* \models \psi_j(w_{j-1}, w_j)$ for all $1 \leq j < i$. Then $(\ell_0, w_0) \xrightarrow{\psi_1} (\ell_1, w_1) \xrightarrow{\psi_2} \dots \xrightarrow{\psi_i} (\ell_i, w_i) \xrightarrow{\psi_{i+1}} (\ell_{i+1}, w_i) \xrightarrow{\psi_{i+2}} (\ell_{i+2}, w_i) \xrightarrow{\psi_{i+3}} \dots$ is an infinite run of $\mathcal{A}$.
- Suppose that π contains infinitely many generous constraints. Let $i_1, i_2, i_3 \dots$ be the sequence of all $j \geq 1$ such that ψ_{i_j} is generous. Use Proposition 5 with the sequence $\psi_1, \dots, \psi_{i_1 - 1}$ and $w_{\text{in}} = 0$ to obtain string values $w_0, w_1, \dots, w_{i_1 - 1} \in \mathbb{N}^+$ such that $\mathbb{N}^* \models \psi_k(w_{k-1}, w_k)$ for all $1 \leq k < w_{i_1 - 1}$. For every $j \geq 1$, use Proposition 5 with the

sequence $\psi_{i_j}, \dots, \psi_{i_{j+1}-1}$ and the initial string value $w_{\text{in}}^j := w_{i_j-1}$ to obtain string values $w_{i_j-1}, w_{i_j}, \dots, w_{i_{j+1}-1} \in \mathbb{N}^+$ such that $\mathbb{N}^* \models \psi_k(w_{k-1}, w_k)$ for all $i_j \leq k < i_{j+1} - 1$. By the second claim of Proposition 5, since ψ_{i_j} in π_j is generous, the initial string value w_{i_j-1} is never rewritten. Hence $\Pi_{i \geq 0} \rho_i$ is an infinite run of $\mathcal{A}$, where $\rho_0 := (\ell_0, w_0) \xrightarrow{\psi_1} (\ell_1, w_1) \xrightarrow{\psi_2} \dots \xrightarrow{\psi_{i_1-1}} (\ell_{i_1-1}, w_{i_1-1})$, and $\rho_j := (\ell_{i_j-1}, w_{i_j-1}) \xrightarrow{\psi_{i_j}} (\ell_{i_j}, w_{i_j}) \xrightarrow{\psi_{i_j+1}} \dots \xrightarrow{\psi_{i_{j+1}-1}} (\ell_{i_{j+1}-1}, w_{i_{j+1}-1})$ for all $j \geq 1$.

For the proof from **2.** to **3.**, let $\pi = \ell_0 \xrightarrow{\psi_1} \ell_1 \xrightarrow{\psi_2} \dots$ be an infinite path of $\mathcal{A}$ satisfying the three conditions stated in **2.** We distinguish two cases:

- Suppose π contains only finitely many generous constraints. By condition **2.c**, π contains only finitely many reducing constraints. Then there exists some $i \geq 0$ such that ψ_j is of the form $x = y$ for all $j > i$. By condition **2.b**, there are infinitely many indices $j > i$ such that $\ell_j = \ell_{\text{acc}}$. Pick two such indices $j < k$ satisfying $\ell_j = \ell_k = \ell_{\text{acc}}$. We have $\ell_0 = \ell_{\text{in}}$ by condition **2.a**, so that clearly, the path $\ell_0 \xrightarrow{\psi_1} \dots \xrightarrow{\psi_j} \ell_j$ is a path from ℓ_{in} to ℓ_{acc}, and the path $\ell_j \xrightarrow{\psi_{j+1}} \dots \xrightarrow{\psi_{k-1}} \ell_k$ is a stable cycle starting in ℓ_{acc}.
- Suppose π contains infinitely many generous constraints. By condition **2.b**, there are infinitely many indices $i \geq 0$ such that $\ell_i = \ell_{\text{acc}}$, so that we can clearly pick three indices j, k, n such that $0 \leq j < k \leq n$, $\ell_j = \ell_n = \ell_{\text{acc}}$, and ψ_k is generous. We have $\ell_0 = \ell_{\text{in}}$ by condition **2.a**, so that clearly, the path $\ell_0 \xrightarrow{\psi_1} \dots \xrightarrow{\psi_j} \ell_j$ is a path from ℓ_{in} to ℓ_{acc}, and the path $\ell_j \xrightarrow{\psi_{j+1}} \dots \xrightarrow{\psi_{k-1}} \ell_k$ is a generous cycle starting in ℓ_{acc}.

For the proof from **3.** to **2.**, suppose π_{in} is a path from ℓ_{in} to ℓ_{acc} for some accepting location $\ell_{\text{acc}} \in \mathcal{L}_{\text{acc}}$, and π_{cyc} is a cycle starting in ℓ_{acc}. Clearly $\pi_{\text{in}} \cdot (\pi_{\text{cyc}})^\omega$ is an infinite path of $\mathcal{A}$ satisfying conditions **2.a** and **2.b**. If π_{cyc} is stable, then this path contains only finitely many reducing constraints. If π_{cyc} is generous, then this path contains infinitely many generous constraints. Hence, condition **2.c** holds, too. ◀

▶ **Theorem 10.** *The emptiness problem for constraint automata over $\mathbb{N}^*$ is NL-complete.*

Proof. For the upper bound, by Proposition 9, it suffices to decide whether there exists some $\ell_{\text{acc}} \in \mathcal{L}_{\text{acc}}$ such that there exists a path from ℓ_{in} to ℓ_{acc}, and one of the following two conditions hold:

- there exists a stable cycle starting in ℓ_{acc}, or
- there exists some genereous transition $(\ell, \psi(x,y), \ell') \in E$ such that there exists a path from ℓ_{acc} to ℓ, and there exists a path from ℓ' to ℓ_{acc}.

All conditions can be checked in NL. A reduction from the emptiness problem for Büchi automata yields the lower bound. ◀

5 Further Results

5.1 Testing Equality with the Empty String

We extend the signature σ^{ps} by a new symbol $=\varepsilon$, which is interpreted as *equality with the empty string*. This enables us to test whether the string value equals the empty string – very similar to testing whether the value of a counter in a counter automaton is equal to zero, or whether the stack of a pushdown automaton is empty. Let us use $\sigma^{ps\varepsilon}$ to denote this signature. We can give a decidable characterization for Büchi-accepting runs of $\mathcal{A}$ and hence obtain:

▶ **Theorem 11.** *The emptiness problem for constraint automata over the extended signature $\sigma^{ps\varepsilon}$ with domain $\mathbb{N}^*$ is NL-complete.*

5.2 Emptiness for Constraint Automata over Σ^*

The decision procedure for solving the emptiness problem for constraint automata over $\mathbb{N}^*$ relies heavily on the existence of "fresh" letters, of which there are unboundedly many in $\mathbb{N}$. We can clearly not apply this algorithm if the constraint automaton is over the structure Σ^*, where Σ is a *finite* alphabet.

Let σ be a relational signature, and let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two σ-structures. A mapping $h : \mathcal{D}_1 \to \mathcal{D}_2$ is a σ-embedding if h is injective, and for all symbols R of arity k, and all $a_1, \dots, a_k \in \mathcal{D}_1$, $R^{\mathcal{D}_1}(a_1, \dots, a_k)$ holds if, and only if, $R^{\mathcal{D}_2}(h(a_1), \dots, h(a_k))$.

Let $\Sigma = \{a, b\}$. Define the mapping $g : \mathbb{N} \to \Sigma$ by $n \mapsto ab^n a$ for all $n \in \mathbb{N}$, and let $h : \mathbb{N}^* \to \Sigma^*$ be its homomorphic extension, that is, $h(n_1 \dots n_k) = g(n_1) \dots g(n_k)$ for all $n_1, \dots, n_k \in \mathbb{N}$, and $h(\varepsilon) = \varepsilon$. One can easily see that h is a $\sigma^{ps\varepsilon}$-embedding. We can conclude that a constraint automaton over Σ^* is a positive instance of the emptiness problem iff the same constraint automaton over $\mathbb{N}^*$ is a positive instance; thus:

▶ **Theorem 12.** *The emptiness problem for constraint automata over the extended signature $\sigma^{ps\varepsilon}$ with domain Σ^* is NL-complete.*

5.3 Emptiness for Single-Register Automata over A^*

Register automata (also known as *finite-memory automata*) [17, 13, 24] are a very popular computational model for the analysis of data languages. Like constraint automata, register automata are parameterized by a σ-structure; in contrast to constraint automata, register automata are "fed" with some input data word, that is a finite or infinite sequence of data values in the domain of the σ-structure. The data language accepted by such an automaton is the set of input data words for which there is an accepting run. Different to the transitions in constraint automata, the transitions of register automata are labelled with constraints of the form $r \bowtie d$, where r corresponds to one of finitely many *registers* of the automaton, d corresponds to the current datum of the input data word, and $\bowtie$ is a binary relation in σ. Further, the current input data value can be stored into one of the registers of the automaton after a transition has been taken.

So far, register automata have mostly been studied for the structure $(\mathbb{N}; =)$ and linear dense orders like $(\mathbb{Q}; <, =)$ [17, 24, 13, 4, 8]. The emptiness problem for register automata is decidable and PSPACE-complete (NL-complete if only one register is used) [13]; the decision procedure relies on a finite abstraction of the infinite state space induced by the input register automaton. This abstraction cannot be applied to register automata over $\sigma^{ps\varepsilon}$-structures Σ^* and $\mathbb{N}^*$.

A register automaton with a single register that stores the current input datum *in every transition* into the register can actually be regarded as a constraint automaton as defined in Section 2: the current value of the register r corresponds to the value of the variable x, and the input datum d corresponds to the value of the variable y at the next position in a run. However, it might be the case that some of the transitions in the register automaton may compare the value of the register *without storing the input datum into the register.* There is no direct way to translate this into constraint automata as defined above. However, an easy extension of our model where we compare x with y, but then set the value of y to x, would make such a translation possible. It can be easily seen that this extension does not cause any problems when applying the developed decision procedure for solving the emptiness problem, so that we can conclude:

▶ **Theorem 13.** *The emptiness problem for single-register automata over the extended signature $\sigma^{ps\varepsilon}$ (with domains $\mathbb{N}^*$ or Σ^*) is NL-complete.*

5.4 Constraint LTL with a Single Variable over A^*

Our result for constraint automata can be used to partially answer the question raised by Demri and Deters [9] concerning the decidability status for CLTL over $\sigma^{ps\varepsilon}$. More detailed, we can prove PSPACE-completeness for the fragment of CLTL that only uses a single variable.

Let $\mathbb{P}$ be a countably infinite set of propositional variables. The set of formulas in CLTL$_1$ is defined by the following grammar

$$\varphi ::= p \mid \psi \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi,$$

where $p \in \mathbb{P}$ and $\psi \in \Psi$. CLTL$_1$ formulas are evaluated over data words over $2^{\mathbb{P}}$ and A^*. Formally, let $u = (a_1, w_1)(a_2, w_2)\dots$ and $i \geq 1$. The satisfaction relation $\models$ is defined as follows:

$$\begin{aligned}
(u,i) \models p \quad &\Leftrightarrow \quad p \in a_i \\
(u,i) \models \psi \quad &\Leftrightarrow \quad A^* \models \psi(w_i, w_{i+1}) \\
(u,i) \models \neg\varphi \quad &\Leftrightarrow \quad \text{not}(u,i) \models \varphi \\
(u,i) \models \varphi_1 \vee \varphi_2 \quad &\Leftrightarrow \quad (u,i) \models \varphi_1 \text{ or } (u,i) \models \varphi_2 \\
(u,i) \models \mathsf{X}\varphi \quad &\Leftrightarrow \quad (u,i+1) \models \varphi \\
(u,i) \models \varphi_1\mathsf{U}\varphi_2 \quad &\Leftrightarrow \quad \exists j \geq i(u,j) \models \varphi_2, \forall i \leq k < j(u,k) \models \varphi_1
\end{aligned}$$

We define $L(\varphi) = \{u \in (2^{\mathbb{P}} \times A^*)^\omega \mid (u,1) \models \varphi\}$. Following the standard translation from LTL to Büchi automata by Vardi and Wolper [29], one can construct from every formula φ a constraint automaton $\mathcal{A}_\varphi$ such that $L(\mathcal{A}_\varphi)$ corresponds to $L(\varphi)$ (cf. [14]). In other words, deciding the satisfiability of φ can be reduced to deciding the non-emptiness of $L(\mathcal{A}_\varphi)$, so that we obtain the following result.

▶ **Theorem 14.** *The satisfiability problem for CLTL$_1$ over the extended signature $\sigma^{ps\varepsilon}$ (with domains $\mathbb{N}^*$ or Σ^*) is PSPACE-complete.*

References

1 Roberto Amadini, Graeme Gange, Peter Schachte, Harald Søndergaard, and Peter J. Stuckey. String constraint solving: Past, present and future. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325, pages 2875–2876. IOS Press, 2020. doi:10.3233/FAIA200431.

2 Philippe Balbiani and Jean-François Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *Frontiers of Combining Systems*, pages 162–176, 2002.

3 Mikolaj Bojanczyk. *Atom Book*. https://www.mimuw.edu.pl/ bojan/paper/atom-book, 2019. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.

4 Mikolaj Bojanczyk, Bartek Klin, and Joshua Moerman. Orbit-finite-dimensional vector spaces and weighted register automata. 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/LICS52264.2021.9470634.

5 Claudia Carapelle, Shiguang Feng, Alexander Kartzow, and Markus Lohrey. Satisfiability of ECTL* with local tree constraints. *Theory Comput. Syst.*, 61(2):689–720, 2017. doi:10.1007/s00224-016-9724-y.

6 Claudia Carapelle, Alexander Kartzow, and Markus Lohrey. Satisfiability of ECTL* with constraints. *J. Comput. Syst. Sci.*, 82(5):826–855, 2016. doi:10.1016/j.jcss.2016.02.002.

7 Karlis Cerans. Deciding properties of integral relational automata. In *Automata, Languages and Programming, 21st International Colloquium, ICALP94*, pages 35–46, 1994. doi:10.1007/3-540-58201-0_56.

8 Wojciech Czerwinski, Antoine Mottet, and Karin Quaas. New techniques for universality in unambiguous register automata. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 129:1–129:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.129.

9 Stéphane Demri and Morgan Deters. Temporal logics on strings with prefix relation. *J. Log. Comput.*, 26(3):989–1017, 2016. doi:10.1093/logcom/exv028.

10 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Inf. Comput.*, 205(3):380–415, 2007. doi:10.1016/j.ic.2006.09.006.

11 Stéphane Demri and Régis Gascon. Verification of qualitative Z constraints. *Theor. Comput. Sci.*, 409(1):24–40, 2008. doi:10.1016/j.tcs.2008.07.023.

12 Stéphane Demri and Régis Gascon. The effects of bounding syntactic resources on presburger LTL. *J. Log. Comput.*, 19(6):1541–1575, 2009. doi:10.1093/logcom/exp037.

13 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

14 Stéphane Demri and Karin Quaas. Concrete domains in logics: a survey. *ACM SIGLOG News*, 8(3):6–29, 2021. doi:10.1145/3477986.3477988.

15 Régis Gascon. An automata-based approach for CTL* with constraints. *Electr. Notes Theor. Comput. Sci.*, 239:193–211, 2009. doi:10.1016/j.entcs.2009.05.040.

16 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005141.

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 Prateek Karandikar and Philippe Schnoebelen. Decidability in the logic of subsequences and supersequences. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, volume 45 of *LIPIcs*, pages 84–97. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.84.

19 Alexander Kartzow and Thomas Weidner. Model checking constraint LTL over trees. *CoRR*, abs/1504.06105, 2015. arXiv:1504.06105.

20 Dietrich Kuske. Theories of orders on the set of words. *RAIRO Theor. Informatics Appl.*, 40(1):53–74, 2006. doi:10.1051/ita:2005039.

21 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019*, volume 11425 of *LNCS*, pages 348–364. Springer, 2019. doi:10.1007/978-3-030-17127-8_20.

22 Gennady S. Makanin. The problem of solvability of equations in a free semi-group. *Math. USSR Sbornik*, 32(2):129–198, 1977.

23 Marvin Minsky. *Computation, Finite and Infinite Machines*. Prentice Hall, 1967.

24 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

25 Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 54–63. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319600.

26 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004. doi:10.1145/990308.990312.

27 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.

28 Luc Segoufin and Szymon Torunczyk. Automata based verification over linearly ordered data domains. In *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 81–92. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. A technical report containing proofs that we refer here to can be found under `https://www.mimuw.edu.pl/~szymtor//papers/regdata.pdf`.

29 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)*, pages 332–344, 1986.

On Uniformization in the Full Binary Tree

Alexander Rabinovich
The Blavatnik School of Computer Science, Tel Aviv University, Israel

Abstract

A function f uniformizes a relation R(X,Y) if R(X,f(X)) holds for every X in the domain of R. The uniformization problem for a logic L asks whether for every L-definable relation there is an L-definable function that uniformizes it. Gurevich and Shelah proved that no Monadic Second-Order (MSO) definable function uniformizes relation “Y is a one element subset of X” in the full binary tree. In other words, there is no MSO definable choice function in the full binary tree.

The cross-section of a relation R(X,Y) at D is the set of all E such that R(D,E) holds. Hence, a function that uniformizes R chooses one element from every non-empty cross-section. The relation “Y is a one element subset of X” has finite and countable cross-sections.

We prove that in the full binary tree the following theorems hold:

▶ **Theorem** (Finite cross-sections). *If every cross-section of an MSO definable relation is finite, then it has an MSO definable uniformizer.*

▶ **Theorem** (Uncountable cross-section). *There is an MSO definable relation R such that every MSO definable relation included in R and with the same domain as R has an uncountable cross-section.*

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Monadic Second-Order Logic, Uniformization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.77

1 Introduction

Let $R \subseteq D_1 \times D_2$ be a binary relation. For $d_1 \in D_1$ the cross-section of R at d_1 is the set $R_{d_1} := \{d \in D_2 \mid (d_1, d) \in R\}$. The domain of R is the set $\{d_1 \in D_1 \mid \exists d_2 (d_1, d_2) \in R\}$.

A uniformizer of R is a subset R^* of R such that for all x: $\exists y R(x,y) \Leftrightarrow \exists! y R^*(x,y)$ (where $\exists!$ stands for “there exists unique”). Hence, a uniformizer for R is a partial function that chooses an element from each non-empty cross-section of R and has the same domain as R (see Fig. 1).

In other words, given an input x for which the original relation R has a non-empty cross-section, a uniformizer returns a single value from the cross-section at x. This is a special case of a choice function.

The axiom of choice implies that a uniformizer always exists; however, it is often important that a uniformizer has some “nice” properties. This is where the uniformization theorems come into action. They guarantee that, if R satisfies certain properties, then it has a uniformizer that has other desirable properties.

A set of relations $\mathfrak{R}$ has the uniformization property if every $R \in \mathfrak{R}$ has a uniformizer in $\mathfrak{R}$.

We consider the set of relations over the full binary tree definable in the Monadic Second-Order Logic (MSO). The seminal Rabin’s theorem states that the Monadic Second-Order Logic is decidable over the full binary tree [8].

The monadic second-order logic is an extension of first-order logic by set variables (which range over the subsets of the domain of a structure), and the quantifiers over the set variables (see Section 2.1).

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 77; pp. 77:1–77:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

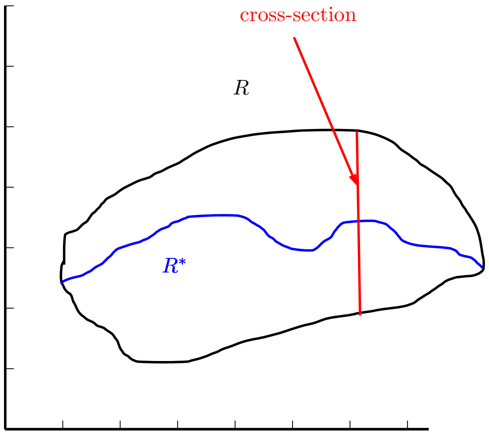


Figure 1 R^* uniformizes R.

An MSO formula $\alpha(X, Y)$ with free set (second-order) variables X and Y defines a binary relation on the subsets of the full binary tree. Rabin [8] proved the basis theorem that states: every non-empty relation definable by an MSO formula $\alpha(Y)$ contains an MSO definable set. The basis theorem is a simple (degenerate) instance of the uniformization property. Rabin asked whether the class of relations definable in the monadic second-order logic over the full binary tree has the uniformization property. Gurevich and Shelah [6] gave a negative solution to Rabin's question with the formula $\alpha(X, Y)$ saying "if X is non-empty, then Y is a one element subset of X" as a counter-example. A function that uniformizes "Y is a one element subset of X" is a choice function; it chooses one element from every non-empty set X. The Gurevich-Shelah Theorem states that there is no MSO definable choice function in the full binary tree.

Since MSO definable relations over the tree do not have the uniformization property, a natural task is to decide whether an MSO definable relation has a (MSO-definable) uniformizer; another natural task is to provide a characterization of those relations which have a uniformizer. The decidability whether a relation has a uniformizer is open. However, we provide a sufficient condition for a relation to have a uniformizer.

▶ **Theorem** (Finite cross-sections). *If the cross-sections of an MSO definable relation over the full binary tree are finite, then it has an MSO definable uniformizer.*

Note that the Gurevich-Shelah Theorem and the above Theorem imply that there are MSO definable relations that do not include MSO definable relations with the same domain and only finite cross-sections.

A natural question is whether an MSO definable relation always includes an MSO definable relation with the same domain and at most countable cross-sections. We provide a negative answer to this question.

▶ **Theorem** (Uncountable cross-section). *There is an MSO definable relation R such that any MSO definable relation which is included in R and has the same domain as R has an uncountable cross-section.*

Hence, it is even impossible to choose in a definable way a countable subset from every non-empty cross-section.

The paper is organized as follows. Section 2 recalls standard definitions about monadic second-order logics, trees, and presents elements of composition method which are used in our proofs. Section 4 provides a proof of Finite cross-section theorem. Section 5 discusses the consequences of undefinability of choice function; it also introduces some notions which are helpful for the proof of Uncountable cross-section theorem given in Section 6. Section 7 contains conclusions and further results.

2 Preliminaries

We use standard notations and terminology. We use n, k, m, l for natural numbers. We use capital letters A, B, C for sets, and lower case letters a, b, c for elements of sets. The powerset of a set D is denoted by $\mathbb{P}(D)$. We use the expressions "chain" and "linear order" interchangeably; we use ω for the order type of the natural numbers.

2.1 Monadic Second-Order Logic

We use standard notations and terminology about Monadic Second-Order logic (MSO) [8, 12, 10].

Let σ be a relational signature. A structure (for σ) is a tuple $\mathcal{M} = (D, \{R^{\mathcal{M}} \mid R \in \sigma\})$ where D is a domain, and each symbol $R \in \sigma$ is interpreted as a relation $R^{\mathcal{M}}$ on D.

MSO formulas use first-order variables, which are interpreted as elements of the structure, and monadic second-order variables, which are interpreted as sets of elements. Atomic MSO formulas are of the following form:

- $R(x_1, \dots, x_n)$ for an n-ary relational symbol R and first-order variables $x_1, \dots, x_n$
- $x = y$ for two first-order variables x and y
- $x \in X$ for a first-order variable x and a second-order variable X

MSO formulas are constructed from the atomic formulas, using boolean connectives, the first-order quantifiers, and the second-order quantifiers.

We write $\psi(X_1, \dots, X_n, x_1, \dots, x_m)$ to indicate that the free variables of the formula ψ are $X_1, \dots, X_n$ (second-order variables) and $x_1, \dots, x_m$ (first-order variables). We write $\mathcal{M}, A_1, \dots, A_n, a_1, \dots, a_m \models \psi$ or $\mathcal{M} \models \psi(A_1, \dots, A_n, a_1, \dots a_m)$ if ψ holds in $\mathcal{M}$ when subsets A_i are assigned to X_i for $i = 1, \dots, n$ and elements a_i are assigned to variables $x_1, \dots, x_m$ for $i = 1, \dots, m$. Whenever $\mathcal{M}$ is clear from the context we will further abbreviate this to $\psi(A_1, \dots, A_n, a_1, \dots a_m)$. Sometimes, we abuse notations and use X for a variable and for the set assigned to it.

▶ **Definition 2.1** (Definability). *Let $\psi(X_1, \dots, X_n)$ be an MSO formula and $\mathcal{M}$ a structure. The relation defined by ψ in $\mathcal{M}$ is the set*

$$\psi\mathcal{M} := \{(D_1, \dots, D_n) \in \mathbb{P}(D)^n \mid \mathcal{M} \models \psi(D_1 \dots, D_n)\}.$$

A relation is MSO-definable in $\mathcal{M}$ if it is equal to $\psi\mathcal{M}$ for an MSO formula ψ. A set $U \subseteq D$ is MSO-definable in $\mathcal{M}$ if there is a formula $\psi(X)$ such that $\psi\mathcal{M} = \{U\}$. A function is MSO-definable in $\mathcal{M}$ if its graph is.

Let $\psi(X_1, \dots, X_n, Y_1, \dots Y_l)$ be an MSO formula and $\mathcal{M}$ a structure. Let $\overline{C}$ be an l-tuple of subsets of the domain D of $\mathcal{M}$. The relation defined by ψ in $\mathcal{M}$ with parameters *$\overline{C}$ is the set*

$$\psi(\mathcal{M}, \overline{C}) := \{(D_1, \dots, D_n) \in \mathbb{P}(D)^n \mid \mathcal{M} \models \psi(D_1 \dots, D_n, \overline{C}\}.$$

The definability of a subset and of a function in $\mathcal{M}$ with parameters is defined in a similar way.

2.2 Trees

We view the set $\{l, r\}^*$ of finite words over the alphabet $\{l, r\}$ as the domain of the full binary tree, where the empty word ϵ is the root of the tree, and for each node $v \in \{l, r\}^*$, we call $v \cdot l$ the left child of v, and $v \cdot r$ the right child of v.

We define a tree order "$\leq$" as a partial order such that $\forall u, v \in \{l, r\}^* : u \leq v$ iff u is a prefix of v.

Nodes u and v are incomparable - denoted by $u \perp v$ - if neither $u \leq v$ nor $v \leq u$; a set U of nodes is an **antichain**, if its elements are incomparable with each other.

We say that an infinite sequence $\pi = v_0, v_1, \ldots$ is a **tree branch** if $v_0 = \epsilon$ and $\forall i \in \mathbb{N} : v_{i+1} = v_i \cdot l$ or $v_{i+1} = v_i \cdot r$.

A path is a finite or infinite sequence $v_0 \ldots v_i \ldots$ such that if v_i is not the last node, then $v_{i+1} = v_i \cdot l$ or $v_{i+1} = v_i \cdot r$.

We consider the full binary tree as a structure for a signature $\{\leq, left, right\}$ where unary relation symbols $left$ and $right$ are interpreted as $\{wl | w \in \{l, r\}^*\}$ and $\{wr | w \in \{l, r\}^*\}$, respectively; $\leq$ is interpreted as the prefix relation.

A k-tree is an expansion of the full binary tree by k unary predicates. Whenever k is clear from the context or unimportant we will use "labelled tree" for "k-tree."

Given a k-tree $\mathfrak{T} := (T, P_1, \ldots, P_k)$ and a node v in $\mathfrak{T}$, the k-tree $\mathfrak{T}_{\geq v} := (T_{\geq v}, P'_1, \ldots, P'_k)$ (called the subtree of $\mathfrak{T}$, rooted at v) is defined by $T_{\geq v}$ is the full binary tree, and $u \in P'_i$ iff $vu \in P_i$ for $i = 1, \ldots, k$.

Let F be an antichain in a k-tree $\mathfrak{T}$ and let $\mathfrak{T}_1$ be a k-tree. The **grafting** of $\mathfrak{T}_1$ at F in $\mathfrak{T}$ is denoted by $\mathfrak{T}\{\mathfrak{T}_1/F\}$ and is the k-tree obtained from $\mathfrak{T}$ when the subtrees rooted at F are replaced by $\mathfrak{T}_1$.

Formally, $\mathfrak{T}' := \mathfrak{T}\{\mathfrak{T}_1/F\}$ is defined as follows: let $u \in \{l, r\}^*$ be a node. (i) if $\neg\exists f \in F(f \leq u)$ then $P_i^{\mathfrak{T}'}(u)$ iff $P_i^{\mathfrak{T}}(u)$; else (ii) $u = fv$ for a unique $f \in F$, and $P_i^{\mathfrak{T}'}(u)$ iff $P_i^{\mathfrak{T}_1}(v)$.

2.3 Composition Method

The proofs presented in this paper make use of the technique known as the "composition method." To fix notations and to aid the reader not familiar with this technique, we briefly review those definitions and results that we need. A more detailed presentation can be found in [11] or [5, 7].[1]

Let σ be a finite relational signature. Write σ_k for the signature σ with k (fresh) unary predicate $P_1, \cdots, P_k$ symbols. Thus, a σ_k-structure $\mathfrak{A}$ has the form $(A, P_1^{\mathfrak{A}}, \cdots, P_k^{\mathfrak{A}})$, where A is a σ-structure. The quantifier rank of a formula φ, denoted $\mathrm{qr}(\varphi)$, is the maximum depth of the nesting of quantifiers in φ. For example, if ψ and φ are quantifier-free, then the quantifier rank of $(\exists x \forall Y \forall Z \psi) \wedge (\forall Y \exists u \varphi)$ is 3. For $r, k \in \mathbb{N}$ we denote by $\mathfrak{Form}_k^r$ the set of formulas of quantifier rank $\leq r$ and with free variables among $X_1, \ldots, X_k$ in signature σ.

For σ_k-structures $\mathfrak{A}, \mathfrak{B}$ write $\mathfrak{A} \equiv_k^r \mathfrak{B}$ if for every $\varphi \in \mathfrak{Form}_k^r$,

$$\mathfrak{A} \models \varphi(P_1^{\mathfrak{A}}, \cdots, P_k^{\mathfrak{A}}) \text{ if and only if } \mathfrak{B} \models \varphi(P_1^{\mathfrak{B}}, \cdots, P_k^{\mathfrak{B}}).$$

Clearly, $\equiv_k^r$ is an equivalence relation and the set $\mathfrak{Form}_k^r$ is infinite. Since the signature σ_k is finite and relational, the set $\mathfrak{Form}_k^r$ contains only finitely many semantically distinct formulas, so there are only finitely many $\equiv_k^r$-classes of σ_k-structures. The following lemma isolates maximally consistent formulas.

[1] In [9], [6], and several other papers, the technique is further developed and a much deeper application of it is made than will be made here.

▶ **Lemma 2.2** (Hintikka lemma). *Let σ be a finite relational signature. For $r, k \in \mathbb{N}$, there is* a finite *set $H_k^r \subseteq \mathfrak{Form}_k^r$ such that:*

1. *For every σ_k-structure $\mathfrak{A}$ there is a* unique *$\tau \in H_k^r$ with $\mathfrak{A} \models \tau(P_1^{\mathfrak{A}}, \cdots, P_k^{\mathfrak{A}})$.*
2. *If $\tau \in H_k^r$ and $\varphi \in \mathfrak{Form}_k^r$, then $\tau \models \varphi$ or $\tau \models \neg\varphi$.*[2]

Elements of H_k^r are called (r, k)-*Hintikka formulas.*

▶ **Definition 2.3** (type of a structure). *For a σ_k-structure $\mathfrak{A}$, write* $\text{type}_k^r(\mathfrak{A})$ *for the unique $\tau(X_1, \cdots, X_k) \in H_k^r$ such that $\mathfrak{A} \models \tau(P_1^{\mathfrak{A}}, \cdots, P_k^{\mathfrak{A}})$, and call it the* (r, k)-type *of $\mathfrak{A}$.*

Thus, $\text{type}_k^r(\mathfrak{A})$ determines for which formulas $\varphi \in \mathfrak{Form}_k^r$ it holds that $\mathfrak{A} \models \varphi(P_1^{\mathfrak{A}}, \cdots, P_k^{\mathfrak{A}})$. Since k is often clear, we may drop it, and write $\text{type}^r(\mathfrak{A})$ and call it the r-type of $\mathfrak{A}$.

We now state weak versions of the composition theorem for MSO over chains and trees, see [9] or [6, 5] for details. The first deals with a representation of ω-chains as a concatenation of finite chains; the second considers a branch in a tree.

▶ **Lemma 2.4** (Weak Composition Theorem for ω-chains). *Let $(L, <)$ be a linear order isomorphic to ω. Let $v_1 < v_2 < \cdots < v_n < \cdots$ be a sequence of elements in L, where v_1 is the minimal element of L. Let $\mathcal{L} := (L, <, P_1, \cdots, P_k)$ be an expansion of $(L, <)$ by unary predicates, and let $\mathcal{L}_i$ be the substructure of $\mathcal{L}$ over $\{v \mid v_i \leq v < v_{i+1}\}$. Then, type$^m(\mathcal{L})$ is determined by the sequence type$^m(\mathcal{L}_i)$ $(i = 1, \ldots)$.*

▶ **Lemma 2.5** (Weak Composition Theorem for a Tree Branch). *Let T be the full binary tree and let P_1, P_2, Q_1, Q_2 be subsets of the nodes of T. Let $T^1 := (T, P_1, Q_1)$ and $T^2 := (T, P_2, Q_2)$ be the expansion of T by the unary predicates. Let $\pi := v_1 v_2 \ldots$ be a branch in T. Let u_i be a child of v_{i-1} different from v_i. Let $\tau_1^i :=$ type$^n(T^1_{\geq u_i})$ and $\tau_2^i :=$ type$^n(T^2_{\geq u_i})$. Assume that $\forall i. \tau_1^i = \tau_2^i$ and $\pi \cap P_1 = \pi \cap P_2$ and $\pi \cap Q_1 = \pi \cap Q_2$. Then type$^n(T^1) =$ type$^n(T^2)$.*

A lemma similar to Lemma 2.5 holds when pairs $\langle P_i, Q_i \rangle$ are replaced by tuples $\langle P_i, Q_i, R_i, \cdots \rangle$.

3 Uniformization

▶ **Definition 3.1** (Uniformization). *Let $\varphi(\overline{X}, \overline{Y})$, $\psi(\overline{X}, \overline{Y})$ be formulas and $\mathcal{C}$ a class of structures. We say that ψ* uniformizes *(or is a* uniformizer *for) φ* over *$\mathcal{C}$ iff for all $\mathcal{M} \in \mathcal{C}$:*

1. *$\mathcal{M} \models \forall \overline{X} \exists^{\leq 1} \overline{Y} \psi(\overline{X}, \overline{Y})$,*
2. *$\mathcal{M} \models \forall \overline{X} \forall \overline{Y} (\psi(\overline{X}, \overline{Y}) \to \varphi(\overline{X}, \overline{Y}))$, and*
3. *$\mathcal{M} \models \forall \overline{X} \big(\exists \overline{Y} \varphi(\overline{X}, \overline{Y}) \to \exists \overline{Y} \psi(\overline{X}, \overline{Y})\big)$.*

Here, $\overline{X}, \overline{Y}$ are tuples of distinct variables and "$\exists^{\leq 1}\overline{Y} \ldots$" stands for "there exists at most one ...". Hence, the first item says that ψ is a graph of a partial function.

The class $\mathcal{C}$ is said to have the uniformization property *iff every formula φ has a uniformizer ψ over $\mathcal{C}$.*

If $\mathcal{C} = \{\mathcal{M}\}$ consists of a single structure, we speak of uniformization in *$\mathcal{M}$ rather than* over *$\mathcal{C}$.*

First, note that, strictly speaking, the question whether φ uniformizes ψ over $\mathcal{C}$ depends not only on formulas φ and ψ, but on the formulas together with a partition of their free-variables into domain variables $\overline{X}$ and image variables $\overline{Y}$. In the few cases where we use

[2] Furthermore, H_k^r is computable from r, k, and there is an algorithm that given τ and φ decides between $\tau \models \varphi$ and $\tau \models \neg\varphi$. We do not use these facts.

letters other than X and Y, we shall state explicitly which variables are to be taken as domain variables and which as image variables and say that φ uniformizes ψ for $(\overline{U};\overline{V})$ over $\mathcal{C}$ if $\overline{U}$ is the set of domain variables and $\overline{V}$ is the set of image variables.

If the cross-sections of a relation R are finite, we say that R is a *finitary* relation.

▶ **Lemma 3.2** (Reducing image variables). *Assume that every finitary relation in $\mathcal{M}$ definable by an* MSO *formula with one image variable has an* MSO *definable uniformizer. Then, every* MSO *definable finitary relation in $\mathcal{M}$ has an* MSO *definable uniformizer.*

4 Finite Cross-sections

▶ **Theorem 4.1** (Finite cross-sections). *If the cross-sections of an MSO definable relation over the full binary tree are finite, then it has an MSO definable uniformizer.*

▶ Remark 4.2 (On computability). It is decidable whether the cross sections of the relation definable by an MSO formula are finite. Our proof of Theorem 4.1 is constructive, and an algorithm which provides an MSO formula that defines a uniformizer can be easily extracted from the proof.

By Lemma 3.2, it is sufficient to prove Theorem 4.1 for the relations definable by formulas with one image variable. To simplify notations we consider formulas with one domain and one image variables. However, everywhere in the proof a domain variable can be replaced by a tuple of domain variables.

▶ **Notations 4.3.** *For a formula $\alpha(X,Y)$ and a subset P of the full binary tree we denote*

1. $\Im\alpha(P) := \{Q \mid \alpha(P,Q)\}$ *- the α-image of P (this is the cross-section of the relation defined by α at P).*
2. $\max_{\subseteq}\Im\alpha(P) := \{Q \in \Im\alpha(P) \mid \neg\exists Y' \in \Im\alpha(P)(Q \subsetneq Y')$ *- $\subseteq$-maximal elements of* $\Im\alpha(P)\}$*. This set is non-empty when $\Im\alpha(P)$ is finite and non-empty.*

It is clear that $\Im\alpha(P)$ and $\max_{\subseteq}\Im\alpha(P)$ are MSO-definable with parameter P.

▶ **Open Question.** *Does there exist an MSO definable linear order on the subsets of the full binary tree?*

If the answer to the question is positive, we can define a uniformizer for α which for every X returns a minimal element in $\Im\alpha(X)$. We have not succeeded to answer the question. However, we still can construct an MSO-definable uniformizer for α. If $\max_{\subseteq}\Im\alpha(P)$ has only one element, the uniformizer chooses it. In Subsection 4.1 we will show how to choose in a definable way between two elements. Relying on this construction, we show in Subsection 4.2 how to choose in a definable way from a finite $\max_{\subseteq}\Im\alpha(P)$, and therefore, from any finite cross-section.

4.1 Choose one Set from two Sets

Here, we are going to show how to choose in a definable way between two elements $Q_1, Q_2 \in \max_{\subseteq}\Im\alpha(P)$.

Let $U_1 := Q_1 \setminus Q_2$ and $U_2 := Q_2 \setminus Q_1$. U_1 and U_2 are non-empty and disjoint. We will use $v\uparrow$ for $\{u \mid v \le u\}$.

Generate a path π as described in **Procedure** Generate π on page 7. We start from the root and at every iteration extend the path by one node. The procedure maintains the invariant: if u is not the last node on the path, then $u\uparrow \cap U_1 \neq \emptyset \neq u\uparrow \cap U_2$ holds. The generated path might be finite (in this case the procedure returns from line 4, 7 or 10) or infinite.

Procedure Generate π.

```
$u \leftarrow root$; $\pi \leftarrow \{u\}$
while $true$ do
    if $u \in U_1 \cup U_2$ then
        return                                                              // (a)
    else if $ul\!\uparrow$ has a non-empty intersection exactly with one of $U_1, U_2$ then
        $\pi \leftarrow \pi \cup \{ul\}$
        return                                                              // (b)
    else if $ur\!\uparrow$ has a non-empty intersection exactly with one of $U_1, U_2$ then
        $\pi \leftarrow \pi \cup \{ur\}$
        return                                                              // (b)
    else if $ul\!\uparrow \cap U_1 \neq \emptyset \neq ul\!\uparrow \cap U_2$ then
        $u \leftarrow ul$
        $\pi \leftarrow \pi \cup \{u\}$
    else
        /* in this case $ur\!\uparrow \cap U_1 \neq \emptyset \neq ur\!\uparrow \cap U_2$                   */
        $u \leftarrow ur$
        $\pi \leftarrow \pi \cup \{u\}$
```

It is clear that π is MSO-definable with parameters U_1 and U_2, i.e., there is an MSO formula $\mu(X_1, X_2, Z)$ such that $\mu(U_1, U_2, \pi)$ holds iff π is generated by this procedure. Moreover, the formula is symmetrical in the parameters, i.e., $\mu(U_1, U_2, \pi) \leftrightarrow \mu(U_2, U_1, \pi)$.

Abbreviations. Below we will abbreviate $\text{type}^n(T, P, Q)$ as $\text{type}^n(P, Q)$ and $\text{type}^n(T, P, Q)_{\geq v}$ as $\text{type}^n(P, Q)_{\geq v}$.

Let us analyse what happens if $\pi := u_1 u_2 \dots$ is infinite. In this case for every i: $u_i \notin (U_1 \cup U_2)$ and $u_i\!\uparrow \cap U_1 \neq \emptyset \neq u_i\!\uparrow \cap U_2$.

Let v_i be a child of u_{i-1} different from u_i. Let n be the quantifier rank of α. Let $\tau_1^i := type^n(P, Q_1)_{\geq v_i}$ and $\tau_2^i := type^n(P, Q_2)_{\geq v_i}$.

▷ Claim 4.4. If π is infinite, then $\exists i(\tau_1^i \neq \tau_2^i)$.

Proof. Toward a contradiction we assume $\forall i(\tau_1^i = \tau_2^i)$, and derive that $\exists^{2^{\aleph_0}} Y \alpha(P, Y)$.

Indeed, let $D := \{i \in \mathbb{N} \mid U_1 \cap (v_i\!\uparrow) \neq \emptyset\}$. First, we observe that D is infinite. Indeed, assume that D is finite and i_0 is its maximal element. Note that $(u_i\!\uparrow \cap U_1) = \{u \in \pi \cap U_1 \mid u \geq u_i\} \cup \bigcup_{j>i}(v_j\!\uparrow \cap U_1)$. Since $\pi \cap U_1 = \emptyset$ and $v_j\!\uparrow \cap U_1 = \emptyset$ for $i > i_0$, we conclude that $u_i\!\uparrow \cap U_1 = \emptyset$ for $i > i_0$. A contradiction.

Now, for every $I \subseteq D$, define $Q_I := \bigcup_{i \in I}(v_i\!\uparrow \cap Q_1) \cup \bigcup_{i \notin I}(v_i\!\uparrow \cap Q_2)$. For every $i \in \mathbb{N}$:

$$\tau_1^i = type^n(P, Q_1)_{\geq v_i} = \tau_2^i = type^n(P, Q_2)_{\geq v_i} = type^n(P, Q_I)_{\geq v_i}.$$

Therefore, by Lemma 2.5

$$type^n(P, Q_1) = type^n(P, Q_I \cup (Q_1 \cap \pi)) \text{ for every } I.$$

Since $\alpha(P, Q_1)$ and the quantifier rank of α is n, we obtain that

$$\alpha(P, Q_I \cup (Q_1 \cap \pi)) \text{ for every } I.$$

For $I_1 \neq I_2 \subseteq D$ we have $Q_{I_1} \neq Q_{I_2}$; moreover, both Q_{I_1} and Q_{I_2} have the empty intersection with π. Therefore, $(Q_{I_1} \cup (Q_1 \cap \pi)) \neq (Q_{I_2} \cup (Q_1 \cap \pi))$, for $I_1 \neq I_2 \subseteq D$. Hence, $|\Im\alpha(P)| \geq |\{I \mid I \subseteq D\}| = 2^{\aleph_0}$. This contradicts our assumption that the cross-sections are finite.

Hence, if π is infinite, then there is $v_i \in \pi$ such that $\tau_1^i \neq \tau_2^i$. ◁

For every P and $Q_1 \neq Q_2 \in \max_{\subseteq} \Im\alpha(P)$ let π be the corresponding path. Define

$$G(P, Q_1, Q_2) = \begin{cases} \text{the last element on } \pi & \text{if } \pi \text{ is finite} \\ \text{the minimal } u_i \in \pi \text{ with } \tau_1^i \neq \tau_2^i & \text{otherwise} \end{cases}$$

From Claim 4.4 and definability of π, we derive that G is total and an MSO definable function.

The next Lemma summarizes properties of a formula which defines G.

▶ **Lemma 4.5.** *There is $\psi(X, Y_1, Y_2, z)$ such that*

1. $\psi(X, Y_1, Y_2, z) \leftrightarrow \psi(X, Y_2, Y_1, z)$
2. $(\psi(X, Y_1, Y_2, z) \wedge Y_1, Y_2 \in \max_{\subseteq} \Im\alpha(X) \wedge Y_1 \neq Y_2) \rightarrow \exists! z \psi(X, Y_1, Y_2, z)$
3. *if $\psi(X, Y_1, Y_2, z) \wedge Y_1, Y_2 \in \max_{\subseteq} \Im\alpha(X) \wedge Y_1 \neq Y_2$, then one of the following conditions holds:*
   a. $z \in Y_1 \Delta Y_2$ *(z in the symmetric difference of Y_1 and Y_2),*
   b. *let $U_1 := Y_1 \setminus Y_2$ and $U_2 := Y_2 \setminus Y_1$, then $U_1 \cap z\!\uparrow \neq \emptyset \wedge U_2 \cap z\!\uparrow = \emptyset$, or $U_1 \cap z\!\uparrow = \emptyset \wedge U_2 \cap z\!\uparrow \neq \emptyset$,*
   c. $\mathit{type}^n(X, Y_1)_{\geq z} \neq \mathit{type}^n(X, Y_2)_{\geq z}$.

Now, we are ready to explain how to choose in a definable way between two elements $Y_1, Y_2 \in \max_{\subseteq} \Im\alpha(X)$. We are going to choose according to the cases (a)-(c) of Lemma 4.5(3). If 3(a) holds, then if $z \in Y_1$ choose Y_1 else choose Y_2. If 3(a) fails, but 3(b) holds, then if $U_1 \cap z\!\uparrow \neq \emptyset$ choose Y_1 else choose Y_2. Let $\leq_n$ be any linear order on a finite set of n-types. If 3(a) and 3(b) fail, then 3(c) holds. In this case, if $\mathit{type}^n(X, Y_1)_{\geq z} <_n \mathit{type}^n(X, Y_2)_{\geq z}$ choose Y_1 else Y_2.

4.2 Choose a Set from Finitely Many Sets

Below we explain how to choose from $\max_{\subseteq} \Im\alpha(P)$ when its cardinality is finite and > 2.

Let $F(X, Y) := \{z \mid (\exists Y' \in \max_{\subseteq} \Im\alpha(X))\psi(X, Y, Y', z)\}$.

F is an MSO-definable function. $F(X, Y)$ maps $\max_{\subseteq} \Im\alpha(P)$ to finite non-empty sets.

Recall that the lexicographical order $<_{lex}$ is an MSO-definable linear order on the nodes of the full binary tree. Define a linear order $\prec_{lex}$ on the finite sets as: $Z' \prec_{lex} Z$ if there is $z \in Z \setminus Z'$ such that $\forall y <_{lex} z (y \in Z \leftrightarrow y \in Z')$. It is clear that $\prec_{lex}$ is MSO-definable.

Assume $Y_1 \neq Y_2 \in \max_{\subseteq} \Im\alpha(X)$ and $F(X, Y_1) = F(X, Y_2) = Z$. Let $z := G(X, Y_1, Y_2)$. Then, $z \in Z$ and one of the conditions 3(a)-3(c) of Lemma 4.5 holds.

Now let us explain how to choose Y from $\max_{\subseteq} \Im\alpha(X)$. It will be easy to see that all the sets and relations described below are MSO-definable. (We will give a verbal description of the relation, leaving it to the reader to check that this verbal description is expressible by an MSO formula.)

1. Let $Z_{\min} := \min_{\prec_{lex}} \{Z = F(X, Y) \mid Y \in \max_{\subseteq} \Im\alpha(X)\}$. We will choose Y from $\Gamma := \{Y \in \max_{\subseteq} \Im\alpha(X) \mid F(X, Y) = Z_{\min}\}$.
2. Define (linear) pre-orders $<_A$, $<_B$ and $<_C$ on Γ (these correspond to items 3(a), 3(b) and 3(c) of Lemma 4.5).

   $Y_1 <_A Y_2$ if $(Y_1 \cap Z_{\min}) \prec_{lex} (Y_2 \cap Z_{\min})$

Let $\Gamma_A :=$ the set of $<_A$ minimal elements of Γ. It is easy to see that if $Y_1, Y_2 \in \Gamma_A$, $Y_1 \neq Y_2$ and $\psi(X, Y_1, Y_2, z)$, then condition 3(a) of Lemma 4.5 fails. Indeed, if 3(a) holds, then $z \in Z_{\min}$ and $z \in Y_1 \Delta Y_2$. However, the minimality of Γ_A implies that $(Y_1 \cap Z_{\min}) = (Y_2 \cap Z_{\min})$ for every $Y_1, Y_2 \in \Gamma_A$. Contradiction.

Now define $<_B$ on Γ_A. Let $U_1 := Y_1 \setminus Y_2$ and $U_2 := Y_2 \setminus Y_1$,

$$Y_1 <_B Y_2 \text{ if } \exists z \in Z_{\min} \text{ such that } U_2 \cap z{\uparrow} \neq \emptyset \wedge U_1 \cap z{\uparrow} = \emptyset \wedge$$
$$\forall z' \in Z_{\min}\ z' <_{lex} z \to (U_1 \cap z'{\uparrow} = \emptyset) \leftrightarrow (U_2 \cap z'{\uparrow} = \emptyset)$$

Let $\Gamma_B :=$ the set of $<_B$ minimal elements of Γ_A. It is easy to see that if $Y_1, Y_2 \in \Gamma_B$, and $Y_1 \neq Y_2$ and $\psi(X, Y_1, Y_2, z)$, then condition 3(b) of Lemma 4.5 fails.

Recall that $\leq_n$ is a linear order on a finite set of n-types. Define $<_C$ on Γ_B.

$$Y_1 <_C Y_2 \text{ if } \exists z \in Z_{\min} \text{ such that}$$
$$\mathit{type}^n(X, Y_1)_{\geq z} <_n \mathit{type}^n(X, Y_2)_{\geq z} \wedge$$
$$\forall z' \in Z_{\min}\ z' <_{lex} z \to (\mathit{type}^n(X, Y_1)_{\geq z'} = \mathit{type}^n(X, Y_2)_{\geq z'})$$

Observe

▷ Claim 4.6. $<_C$ is a linear order on Γ_B.

Proof. It is clear that $<_C$ is irreflexive. We will prove that (1) $<_C$ is transitive and (2) If $Y_1 \neq Y_2$ then $Y_1 <_C Y_2$ or $Y_2 <_C Y_1$. These imply that $<_C$ is a linear order.

(1) $<_C$ is transitive. Indeed, let $Y_1, Y_2, Y_3 \in \Gamma_B$ and $Y_1 <_C Y_2 <_C Y_3$. Assume that $z_{1,2}$ is a witness that $Y_1 <_C Y_2$, i.e., $\mathit{type}^n(X, Y_1)_{\geq z_{1,2}} <_n \mathit{type}^n(X, Y_2)_{\geq z_{1,2}}$ and $\forall z' \in Z_{\min}\ z' <_{lex} z_{1,2} \to (\mathit{type}^n(X, Y_1)_{\geq z'} = \mathit{type}^n(X, Y_2)_{\geq z'}$. Assume that $z_{2,3}$ is a witness that $Y_2 <_C Y_3$.

If $z_{1,2} <_{lex} z_{2,3}$, then $z_{1,2}$ is a witness that $Y_1 <_C Y_3$; otherwise $z_{2,3}$ is a witness that $Y_1 <_C Y_3$. Hence, $<_C$ is transitive.

(2) Now, we prove that If $Y_1 \neq Y_2$ then $Y_1 <_C Y_2$ or $Y_2 <_C Y_1$.

Since, $Y_1, Y_2 \in \Gamma_B$, we know that

if $\psi(X, Y_1, Y_2, z)$, then 3(a) and 3(b) of Lemma 4.5 fail. Hence, 3(c) holds, i.e., $\mathit{type}^n(X, Y_1)_{\geq z} \neq \mathit{type}^n(X, Y_2)_{\geq z}$ Therefore,

$$\exists z \in Z_{\min} \text{ such that } \mathit{type}^n(X, Y_1)_{\geq z} \neq \mathit{type}^n(X, Y_2)_{\geq z} \wedge$$
$$\forall z' \in Z_{\min}\ z' <_{lex} z \to (\mathit{type}^n(X, Y_1)_{\geq z'} = \mathit{type}^n(X, Y_2)_{\geq z'}$$

Hence, $Y_1 <_C Y_2$ or $Y_2 <_C Y_1$. ◁

Now, we choose (a unique) $<_C$-minimal element of Γ_B.

Below is the summary of our choice of $Y \in \max_{\subseteq} \Im\alpha(X)$:

1. $Z_{\min} := \min_{\prec_{lex}}\{Z = F(X, Y) \mid Y \in \max_{\subseteq} \Im\alpha(X)\}$ and $\Gamma := \{Y \in \max_{\subseteq} \Im\alpha(X) \mid F(X, Y) = Z_{\min}\}$.
2. $\Gamma_A :=$ the set of $<_A$ minimal elements of Γ.
3. $\Gamma_B :=$ the set of $<_B$ minimal elements of Γ_A.
4. We choose a unique $<_C$ minimal elements of Γ_B.

It is clear that the above choice can be formalized in MSO.

5 Choice Function and Fooling Sets

A *choice function* is a mapping which assigns to each non-empty set of nodes one element from the set.

Gurevich and Shelah [6] proved:

▶ **Theorem 5.1** (Gurevich and Shelah [6]). *There is no MSO-definable choice function in the full-binary tree.*

A simplified combinatorial proof of Theorem 5.1 was given in [2, 3].

In the rest of this section we introduce some notions and prove lemmas which will be used in the next section to prove uncountable cross-section theorem.

Choice Function on antichains. A choice function on antichains is a mapping which assigns to each non-empty antichain in the full binary tree one element from the antichain.

▶ **Corollary 5.2.** *There is no MSO-definable choice function on antichains.*

Proof. If $\beta(X, y)$ defines a choice function on antichains then $\alpha(X, y) := \exists Z$ "Z is the set of $\leq$ minimal elements of X"$\wedge\beta(Z, y)$ defines a choice function - a contradiction. ◀

▶ **Definition 5.3** (Fooling set). *Assume $\alpha(X, y) \rightarrow y \in X$. A set P is fooling for $\alpha(X, y)$ (wrt choice) if $P \neq \emptyset$ and $\neg\exists!y\alpha(P, y)$. P is a fooling set for a set Φ of formulas if it is a fooling set for each formula in Φ. If, in addition, P is an antichain, we say that P is a fooling antichain.*

It is helpful to use the following convention: If $\alpha(X, y)$ does not imply $y \in X$, then every set is fooling for α. As a consequence: P is fooling for a set Φ of formulas iff P is fooling for $\{\varphi \in \Phi \mid \varphi \rightarrow y \in X\}$. Theorem 5.1 and Corollary 5.2 imply that every formula has a fooling set and a fooling antichain.

▶ **Lemma 5.4.** *The following statements hold in the full binary tree:*

1. *There is no MSO-definable choice function.*
2. *Every $\alpha(X, y)$ has a fooling set.*
3. *Every finite set of formulas has a fooling set.*
4. *There is no MSO-definable choice function on antichains.*
5. *Every formula has a fooling antichain.*
6. *Every finite set of formulas has a fooling antichain.*

Proof. First we prove that implications (1) ⇒ (2) ⇒ (3) ⇒(1) hold in every structure. These and Theorem 5.1 imply that (1)-(3) are true in the full binary tree.

(1)⇒(2). If $\alpha(X, y)$ does not define a choice function and $\alpha(X, y) \rightarrow y \in X$, then there is a fooling set for α.

(2)⇒(3). Let $\alpha_1(X, y), \dots, \alpha_n(X, y)$ be a finite sequence of formulas such that $\alpha_i(X, y) \rightarrow y \in X$.

Define

$$Fool_i(X) := X \neq \emptyset \wedge \neg\exists!y\alpha_i(X, y)$$

$Fool_i(X)$ holds iff X is a fooling set for α_i.

If $\bigwedge_{i=1}^{n} Fool_i(X)$ is satisfiable, then there is a fooling set for $\{\alpha_i \mid i = 1, \dots n\}$.

Toward a contradiction we assume that $\bigwedge_{i=1}^{n} Fool_i(X)$ is unsatisfiable, and construct a formula that has no fooling set.

For $I \subseteq \{1,\dots,n\}$ define $FOOL_I(X) := \bigwedge_{i\in I} Fool_i(X) \wedge \bigwedge_{j\notin I} \neg Fool_j(X)$.

Observe that $FOOL_I(X)$ for $I \subsetneq \{1,\dots,n\}$ defines a partition, i.e., if $I_1 \neq I_2$, then $FOOL_{I_1}(X) \wedge FOOL_{I_2}(X)$ is unsatisfiable, and $\bigvee_{I\subsetneq\{1,\dots,n\}} FOOL_I(X)$ holds for every X.

Let c be a function which assigns to every proper subset of $\{1,\dots,n\}$ an element in its complement. For $I \subsetneq \{1,\dots,n\}$ define

$$\beta_I(X,y) := FOOL_I(X) \wedge \alpha_j(X,y) \text{ for } j = c(I) \notin I$$

and

$$\gamma(X,y) := \bigvee_{I\subsetneq\{1,\dots,n\}} \beta_I$$

For every P there is a unique $I \subsetneq \{1,\dots,n\}$ such that $FOOL_I(P)$. Hence, by the definition of γ: $\gamma(P,y)$ iff $FOOL_I(P) \wedge \alpha_j(P,y)$ for $j = c(I)$. Now, $FOOL_I(P)$ implies $\neg Fool_j(P)$ for $j = c(I)$. Therefore, if $P \neq \emptyset$ then $\exists! y \alpha_j(P,y)$, and therefore, $\exists! y \beta_I(P,y)$, and $\exists! y \gamma(P,y)$. A contradiction to "γ has a fooling set."

Hence, there is a fooling set for $\{\alpha_i \mid i = 1,\dots n\}$.

(3)⇒(1) trivial.

(4) - this is Corollary 5.2.

Finally, the proof of the equivalences between (4),(5) and (6) is obtained by replacing "fooling set" by "fooling antichain" in the proof of the equivalences between (1),(2) and (3). ◀

Recall that there are finitely many (semantically distinct) formulas of qr $\leq n$ with free variables X and y. We say that P is **n-fooling** if it is fooling for the formulas of qr $\leq n$. Hence, by Lemma 5.4,

▶ **Corollary 5.5.** *For every n there is an n-fooling set.*

6 Uncountable Cross-section Theorem

In this section we prove Uncountable Cross-section Theorem which is stated in the Introduction. The idea is to force infinitely many choices from a fooling set. Since every choice leads to at least two possibilities, infinitely many choices lead to uncountable many possibilities. Below are details.

Consider a relation $R^{\infty}_{choice}(X,Y)$ defined as: "Y is a subset of X such that for every n: $r^n l{\uparrow} \cap Y$ is a one element set." It is clear that $R^{\infty}_{choice}(X,Y)$ is MSO definable.

Let P be a subset of $\{l,r\}^*$ and let $r^*lP := \{u \mid u = r^n lv \text{ for } v \in P \text{ and } n \geq 0\}$. Below "$\exists^{2^{\aleph_0}} Y$" stands for "there are at least $2^{\aleph_0}$ different Y."

▶ **Proposition 6.1.**

1. *Assume that $\alpha(X,Y)$ has* qr $\leq n$ *and P is an $n+1$-fooling set. If the cross-section of $\alpha(X,Y)$ at r^*lP is included in the cross-section of $R^{\infty}_{choice}(X,Y)$ at r^*lP, then $T, r^*lP \models \exists Y \alpha(X,Y) \rightarrow \exists^{2^{\aleph_0}} Y \alpha(X,Y)$.*
2. *The relation $R^{\infty}_{choice}(X,Y)$ contains no* MSO*-definable relation with the same domain and only cross-sections of cardinality less than $2^{\aleph_0}$.*

Proof. (2) immediately follows from (1).

(1) First, observe that if P is an $n+1$-fooling set, then no $\beta(X,Y)$ of qr $\leq n$ can choose a unique one element subset of P, i.e., $\neg\big(\exists! Y \beta(P,Y) \wedge \exists Y(\beta(X,Y) \wedge \exists y Y = \{y\})\big)$. Indeed, if β chooses a one element subset of P, then $\exists Y \beta(X,Y) \wedge y \in Y$ chooses a unique element from P. This contradicts that P is $n+1$-fooling.

Second, observe that (T, P) is isomorphic to $(T, r^*lP)_{\geq r^n l}$ for every n, and (T, r^*lP) is obtained from $(T, \emptyset)$ by grafting (T, P) at an antichain $\{r^i l \mid i \in \mathbb{N}\}$.

Let Q be such that $T, r^*lP, Q \models \alpha(X, Y)$. Since $\forall Y(\alpha(r^*lP, Y) \to R^{\infty}_{choice}(r^*lP, Y))$, there are $w_i \in P$ such that $(r^i l{\uparrow} \cap Q) = \{r^i l w_i\}$. Let $\tau_i(X, Y)$ be the n-type of the subtree rooted at $r^i l$ in (T, r^*lP, Q). Then $T, P, \{w_i\} \models \tau_i(X, Y)$. Since P is an $n+1$-fooling set, by the first observation above, there is $W'_i \neq \{w_i\}$ such that $T, P, W'_i \models \tau_i(X, Y)$.

Let $\pi_r := v_1 v_2 \ldots$ be the rightmost branch in the full binary tree T, i.e., $v_i = r^i$ for $i \in \mathbb{N}$. For $I \subseteq \mathbb{N}$ define $Q_I := \{r^i l w_i \mid i \in I\} \cup \bigcup_{i \notin I} r^i l W'_i$. Then the assumptions of Lemma 2.5 hold for $\pi := \pi_r$, $T^1 := (T, r^*lP, Q)$ and $T^2 := (T, r^*lP, Q_I)$. Therefore, by Lemma 2.5, $T, r^*lP, Q_I \models \alpha$ for every $I \subseteq \mathbb{N}$. Since $Q_I \neq Q_{I'}$ for $I \neq I'$, we obtain $T, r^*lP \models \exists^{2^{\aleph_0}} Y \alpha(X, Y)$. ◀

The next proposition describes a property that is more natural than R^{∞}_{choice}, and which also implies Uncountable Cross-section Theorem.

▶ **Proposition 6.2.** *The relation "Y is a branch such that $X \cap Y$ is infinite" contains no* MSO*-definable relation with the same domain and only cross-sections of cardinality* $< 2^{\aleph_0}$.

7 Conclusions and Further Results

Let us introduce some weaker variants of uniformization. Let $R^* \subseteq R$ be two relations with the same domain. If the cross-sections of a relation R^* are at most l for some $l \in \mathbb{N}$ (respectively, finite, countable), we say that R^* is an l-relation (respectively, a finitary relation, an $\aleph_0$-relation).

If R^* is an l-relation for some $l \in \mathbb{N}$ (respectively, a finitary or an $\aleph_0$-relation) we say that R^* is an l-uniformizer of R (respectively, finitary uniformizer, $\aleph_0$-uniformizer).

We say that ψ is an l-uniformizer of φ in a structure $\mathcal{M}$, if the relation definable by ψ in $\mathcal{M}$ is an l-uniformizer of the relation definable by φ in $\mathcal{M}$.

$\mathcal{M}$ has the l-uniformization property if for every MSO formula φ there is an MSO formula ψ that is an l-uniformizer of φ in $\mathcal{M}$. Finitary (and $\aleph_0$) uniformizers and the finitary (and $\aleph_0$) uniformization property are defined similarly.

For $l \in \mathbb{N}$, we say that the **uniformization rank** of R is l if R has an MSO-definable l-uniformizer and either $l = 1$ or R has no MSO-definable $(l-1)$-uniformizer. We say that the uniformization rank of R is finitary if R has an MSO-definable finitary uniformizer and has no MSO-definable l-uniformizer for $l \geq 1$. We say that the uniformization rank of R is $\aleph_0$ if R has an MSO-definable $\aleph_0$-uniformizer and has no finitary uniformizer. We say that the uniformization rank of R is $2^{\aleph_0}$ if R has no uniformizer with the cross-section of cardinality $< 2^{\aleph_0}$.

Our result can be restated as:

▶ **Corollary 7.1.** *The only ranks for* MSO*-definable relations in the full binary tree are one,* $\aleph_0$ *and* $2^{\aleph_0}$.

Indeed, the finite cross-section theorem implies that the rank is one or infinite. The relation $y \in X$ has rank $\aleph_0$. The relations $R^{\infty}_{choice}(X, Y)$ and "Y is a branch such that $X \cap Y$ is infinite" have rank $2^{\aleph_0}$. For the MSO-definable (with parameters) relation the continuum hypothesis holds [1]. Therefore, the infinite cross-sections of MSO-definable relations are either countable or have cardinality $2^{\aleph_0}$. Hence, Corollary 7.1 holds.

7.1 Ranks over Integers

In [4] the uniformization of MSO over the integers with the successor function was investigated. Consider, $\varphi_2(X,Y) := \forall t(Y(t) \leftrightarrow \neg Y(t+1))$. Note that φ_2 does not depend on X and there are exactly two sets Y_{even} := "the set of even integers" and Y_{odd} :="the set of odd integers" that satisfy φ_2. Note also that there is an order preserving automorphism of integers that maps Y_{even} onto Y_{odd}. Therefore, no MSO formula distinguishes between Y_{even} and Y_{odd}. Hence, the uniformization rank of φ_2 is two. Similarly, for every l one can define an MSO formula φ_l which has the uniformization rank l over the integers. It was proved in [4] that the uniformization rank of an MSO formula over integers is computable.

It is open whether the uniformization rank of an MSO formula over the full binary tree is computable.

7.2 Ranks over Ordinals

Lifsches and Shelah [7] considered the uniformization problem over the class of trees and the class of ordinals. Recall that a structure $\mathcal{M}$ is said to have the uniformization property if every MSO definable relation (in $\mathcal{M}$) has an MSO definable uniformizer. Lifsches and Shelah proved that an ordinal α has the uniformization property iff $\alpha < \omega^\omega$.

In the full paper we consider uniformization over ordinals. We have not provided an algorithm to decide the uniformization rank of a formula. However, we prove that if an MSO definable relation over an ordinal has only countable cross-sections, then it has a uniformizer. Moreover, the only ranks for the MSO-definable relations over a countable ordinal are one and $2^{\aleph_0}$.

7.3 Uniformization Degrees

We know that the formula $\psi_1 := y \in X$ has no MSO uniformizer in the full binary tree. Now, let us look at the formula ψ_2 stating that"Y is a branch such that $Y \cap X$ is infinite."

This formula has no MSO uniformizer. But are there any other interesting relations between these two formulas? Can we say, for instance, that ψ_2 is even "harder" to uniformize than ψ_1 (whatever this might mean)? Or, perhaps the other way round? Do we feel that the example of ψ_2 "contains a new idea" when it comes to our discussion of uniformization? To turn these admittedly vague questions into mathematical ones, we require a notion of comparing formulas and perhaps an equivalence relation on them. However, as our example shows, the semantical equivalence seems not to be the right notion. Note, however, the following. For any set X let $X{\uparrow} := \{z \mid \exists x \in X(z \geq x)\}$ be the upward closure of X. X is non-empty iff there is a branch Y such that $|Y \cap X{\uparrow}|$ is infinite. Moreover, if $Y \cap X \neq \emptyset$, then $Y \cap X{\uparrow}$ has a minimal point y which is in X. Hence, by using any uniformizer for ψ_2 we succeeded to define (in MSO) a uniformizer for ψ_1.

This suggests the following definition.

Let $\mathcal{M}$ be a structure in a signature Σ and $\psi(\overline{X},\overline{Y})$ and $\psi_2(\overline{U},\overline{V})$ be formulas in Σ. ψ_1 is easier to uniformize (in $\mathcal{M}$) than ψ_2 if there is an MSO formula $\varphi(\overline{X},\overline{Y})$ (reduction formula) in an expansion of Σ by a relational symbol c such that:

if $\mathcal{M}[c]$ is an expansion of $\mathcal{M}$, where c is interpreted as a uniformizer of ψ_2, then $\varphi(\overline{X},\overline{Y})$ is a uniformizer of ψ_1.

We write $\psi_1 \preceq_{un} \psi_2$ if ψ_1 is easier to uniformize than ψ_2. The relation $\preceq_{un}$ is a preorder relation and its equivalence classes can be called (uniformization) degrees.

We can show that "$y \in X$," $R^{\infty}_{choice}(X,Y)$, "Y is a branch such that $Y \cap X$ is infinite," "Y is a branch such that $Y \cap X$ is finite" and "Y is a finite non-empty subset of an antichain X" have the same degree. It is interesting to study the structure of degrees. In particular, we do not know whether there is a maximal degree.

References

1 Vince Bárány, Łukasz Kaiser, and Alex Rabinovich. Expressing cardinality quantifiers in monadic second-order logic over trees. *Fundamenta Informaticae*, 100(1-4):1–17, 2010.

2 Arnaud Carayol and Christof Löding. MSO on the infinite binary tree: Choice and order. In *International Workshop on Computer Science Logic*, pages 161–176. Springer, 2007.

3 Arnaud Carayol, Christof Löding, Damian Niwinski, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Open Mathematics*, 8(4):662–682, 2010.

4 Grzegorz Fabianski, Michal Skrzypczak, and Szymon Torunczyk. Uniformisations of regular relations over bi-infinite words. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 384–396. ACM, 2020. doi:10.1145/3373718.3394782.

5 Yuri Gurevich. Monadic second-order theories. *Model-theoretic logics*, pages 479–506, 1985.

6 Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem 1. *The Journal of Symbolic Logic*, 48(4):1105–1119, 1983.

7 Shmuel Lifsches and Saharon Shelah. Uniformization and skolem functions in the class of trees. *The Journal of Symbolic Logic*, 63(1):103–127, 1998.

8 Michael O Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the american Mathematical Society*, 141:1–35, 1969.

9 Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, 102(3):379–419, 1975.

10 Wolfgang Thomas. Automata on infinite objects. In *Formal Models and Semantics*, pages 133–191. Elsevier, 1990.

11 Wolfgang Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science*, pages 118–143. Springer, 1997.

12 Boris Trakhtenbrot and Ya Martynovich Barzdin. *Finite automate: behaviour and synthesis.* North-Holland, 1973.

An Exact Algorithm for Knot-Free Vertex Deletion

M. S. Ramanujan ✉
Department of Computer Science, University of Warwick, UK

Abhishek Sahu ✉
Institute of Mathematical Sciences, Chennai, India

Saket Saurabh ✉
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

Shaily Verma ✉
Institute of Mathematical Sciences, Chennai, India

Abstract

The study of the Knot-Free Vertex Deletion problem emerges from its application in the resolution of deadlocks called knots, detected in a classical distributed computation model, that is, the OR-model. A strongly connected subgraph Q of a digraph D with at least two vertices is said to be a knot if there is no arc (u,v) of D with $u \in V(Q)$ and $v \notin V(Q)$ (no-out neighbors of the vertices in Q). Given a directed graph D, the Knot-Free Vertex Deletion (KFVD) problem asks to compute a minimum-size subset $S \subset V(D)$ such that $D[V \setminus S]$ contains no knots. There is no exact algorithm known for the KFVD problem in the literature that is faster than the trivial $\mathcal{O}^\star(2^n)$ brute-force algorithm. In this paper, we obtain the first non-trivial upper bound for KFVD by designing an exact algorithm running in time $\mathcal{O}^\star(1.576^n)$, where n is the size of the vertex set in D.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases exact algorithm, knot-free graphs, branching algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.78

Funding *M. S. Ramanujan*: Supported by Engineering and Physical Sciences Research Council (EPSRC) grants EP/V007793/1 and EP/V044621/1.
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 he also acknowledges the support of Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

In concurrent computing, a deadlock [6] is a state in which each group member waits for another member, including itself, to take action such as sending a message or, more commonly, releasing a lock. Deadlocks are a common problem in multiprocessing systems, parallel computing, and distributed systems. Resolving these deadlocks is a fundamental problem in distributed settings with no efficient (known) algorithms. Note that while distributed systems are dynamic, a deadlock is a stable property. In other words, once a deadlock occurs in the system, it would remain there until it is resolved. Two of the main classic deadlock models are the AND-model and the OR-model [1,2,14,17]. Deadlocks are characterized by the existence of cycles in the AND-model and by the existence of knots in the OR-model. Hence the problem of preventing deadlocks in the AND-model is equivalent to the Directed Feedback Vertex Set (DFVS) problem and in the OR-model, it is equivalent to the Knot-Free Vertex Deletion (KFVD) problem [13].

A *wait-for* graph is useful to analyze deadlock situations. In a wait-for graph, $D = (V; E)$, the vertex set V represents processes, and the set E of directed arcs represents wait-conditions.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 78; pp. 78:1–78:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A knot in a directed graph D is a strongly connected subgraph Q of D with at least two vertices, such that no vertex in $V(Q)$ has an out-neighbor in $V(D) \setminus V(Q)$. The number of processes needed to be preempted to prevent a deadlock in the OR-model is essentially the same as solving the KFVD problem in wait-for graphs. Formally, the problem of finding a vertex subset $S \subseteq V$ of minimum cardinality such that $D[V \setminus S]$ does not contain any knots is known as the KFVD problem.

Recently, KFVD has been studied extensively from algorithmic perspectives. In particular, the problem has been studied on graph classes satisfying some structural properties and from the viewpoint of parameterized complexity [4]. In this paper, we study KFVD from yet another algorithmic paradigm, that is, exact exponential-time algorithms. There has been immense progress in this area in the last two decades, resulting in non-trivial exact exponential algorithms for numerous problems, including Chromatic Number, Hamiltonian Cycle and Satisfiability [3, 9, 12, 16] We refer to the monograph of Fomin and Kratsch for a detailed exposition of the field [11].

1.1 Our contribution

In this paper, using the powerful method of *Measure & Conquer*, pioneered by Fomin, Kratsch, and Woeginger [10], we design the first non-trivial exact algorithm for the KFVD problem. In particular, we obtain the following result.

▶ **Theorem 1.** *There exists an algorithm for* Knot-free vertex deletion *running in* $\mathcal{O}^*(1.576^n)$ *time.*

The starting point of our algorithm is the following simple observation: *a graph is knot-free if and only if every vertex has a path to a sink.* Moreover, finding a minimum size knot-free deletion set is equivalent to finding a subset Z of *sinks (vertices with no out-neighbors)* such that $N^+(Z)$ is exactly the deletion set [2], that is $N^+(Z) = S$. Our algorithm utilizes this observation, and rather than obtaining a minimum deletion set S; it constructs a sink set Z such that every vertex in $D - N^+(Z)$ has a path in $D - N^+(Z)$ to some vertex in Z. Note that Z does not complement S The measure associated with the instance $I = (D, V_1, V_2)$, is given by $\phi(I) = |V_1| + \frac{|V_2|}{2}$. We select a vertex $u \in V_1$ to branch, which means that either u is a sink, or it is not. If u is going to be a sink, then observe that all its out-neighbors, $N^+(u)$, must be in the deletion set and be safely deleted. Further, every vertex that can reach u in $D - N^+(u)$, say in-reachable vertices, must not go to the deletion set. However, we can remove them and work on a smaller graph. Thus removing in-reachable vertices is a prepossessing step that reduces the size of the graph. In the branch where we decide that u is not part of the sink set, we cannot delete this vertex, and thus, to capture our progress, we move the vertex u from V_1 to V_2, resulting in ϕ decreasing by 0.5. To show the desired running time, we first do "potential sensitive" branching and then do an extensive case analysis based on the degree of the vertex in V_1.

1.2 Previous Work

Observe that any directed feedback vertex set (a set of vertices that intersect every cycle in the digraph) is a knot-free deletion set as it removes any strongly connected component of size at least 2. Hence, the size of the smallest directed feedback vertex set gives an upper bound on the size of the smallest knot-free vertex deletion set. One way of eliminating knots in a graph could be to use known algorithms for the Directed Feedback Vertex Set (DFVS) problem. While DFVS has been extensively studied [7, 15], the KFVD problem is yet to receive the same scale of attention. Notice that the optimal solution for KFVD could be significantly smaller than the solution returned by invoking an existing algorithm for DFVS, which motivated a closer look at the KFVD problem itself.

Carneiro, Souza, and Protti [5] first studied the problem and showed that KFVD is NP-complete even when restricted to planar bipartite graphs with the maximum degree 4. They also presented a polynomial-time algorithm for graphs with the maximum degree 3. Recently, KFVD was studied in the parameterized framework. Carneiro et al. [4] showed that KFVD is W[1]-hard when parameterized by the size of the solution, but it can be solved in $2^{k \log \phi} n^{\mathcal{O}(1)}$ time, where ϕ is the size of the largest strongly connected subgraph. KFVD parameterized by the size of the solution k, is W[1]-hard even when the length of a longest directed path of the input graph, as well as its Kenny-width, are bounded by constants [2]. However, the problem is known to be in FPT parameterized by either clique-width or treewidth of the underlying undirected graph. Moreover, the KFVD problem is known to admit FPT algorithms when parameterized by *dfvs+maximum path length* or *dfvs+Kenny-width* where, *dfvs* denotes the size of the smallest directed feedback vertex set in the graph.

Organization of the paper: In Section 3, we start by providing our algorithm, and we verify the correctness of each of its steps in Section 4. The running time is analyzed in Section 5, while Section 6 provides a lower bound on the worst-case performance of our algorithm.

2 Preliminaries and Auxiliary Results

In this section, we first state some notations, definitions, and useful auxiliary results. We also formalize an appropriate potential function based on which we later design our algorithm. A few reduction rules are proved towards the end of this section which are used throughout the paper.

Standard Notation. For a digraph D, $V(D)$ and $E(D)$ denote the set of vertices and arcs, respectively. We denote an arc from u to v by the ordered pair (u, v). For a vertex v of D, the out-neighborhood of v is denoted by $N^+(v) = \{u \mid (v, u) \in E(D)\}$. Similarly, we denote its in-neighborhood by $N^-(v) = \{u \mid (u, v) \in E(D)\}$ and $N(v) = N^+(v) \cup N^-(v)$. A vertex v is called a *source vertex*, if $N^-(v) = \emptyset$. Similarly it is called a *sink vertex*, if $N^+(v) = \emptyset$. We define the *in-reachability set* of a vertex v as the set of vertices that can reach the vertex v via some directed path in $D - N^+(v)$ and we denote it by $R^-(v)$. Notice that $v \in R^-(v)$. We define, $R(v) = N^+(v) \cup R^-(v)$. For a set $S \subseteq V(D)$, $D - S$ denotes the digraph obtained by deleting the vertices S and the edges incident on the vertices of S and $D[S]$ denotes the subgraph of D induced on S. A *path* P of length ℓ is a sequence of distinct vertices $v_1, v_2, \ldots, v_\ell$ such that (v_i, v_{i+1}) is an arc, for each i, $i \in [\ell - 1]$. In our algorithm, branching vector (a, b) means a branching where in the first branch, the potential (measure) drops by a while in the second branch, the potential drops by b. It is a measure of the effectiveness of any branching step. Later we provide a detailed description of such a potential function and how it relates to the hardness of our instance. For graph-theoretic terms and definitions not stated explicitly here, we refer to [8].

2.1 Auxiliary Results

In this subsection, we first state some of the known reduction rules and some new reduction rules for the problem that we use in our branching algorithm.

▶ **Reduction Rule 2** ([2]). *Let $v \in D$ be such that $N^-(v) = \emptyset$, Then (D, k) is a yes-instance if and only if $(D - v, k)$ is a yes-instance.*

▶ **Reduction Rule 3** ([2]). *Let $v \in D$ be such that $N^+(v) = \emptyset$, Then (D, k) is a yes-instance if and only if $(D - R^-(v), k)$ is a yes-instance.*

▶ **Proposition 4** ([2]). *A digraph D is knot-free if and only if for every vertex v of D, v has a path to a sink.*

▶ **Corollary 5** ([2]). *For any optimal solution $S \subseteq V(D)$ with the set of sink vertices Z in $D - S$, we have $N^+(Z) = S$.*

Proposition 4 and Corollary 5 imply that given a digraph D, the problem of finding a set of sink vertices Z such that every vertex in $V(D) \setminus N^+(Z)$ has a directed path to a vertex in Z and $|N^+(Z)|$ is minimum; is equivalent to the Knot-free Vertex Deletion (KFVD) problem. Therefore, our algorithm aims to identify a set of (eventually, to be) sink vertices Z while minimizing $|N^+(Z)|$ instead of directly looking for the deletion set.

Strategy of our Algorithm. We design a branching algorithm for KFVD in general digraphs. At any iteration in our algorithm, we branch on a potential vertex $v \in V(D)$ based on the two possibilities that either v is a sink vertex or a non-sink vertex in an optimal solution. Observe that even if a vertex becomes a non-sink vertex corresponding to an optimal solution, there are two possibilities: it belongs to the deletion set or does not. So we can not simply forget about this vertex, which means its behavior in the final knot-free graph remains inconclusive. To track the possible vertices that become the sink or non-sink, we use a potential function (ϕ) for $V(D)$ defined as follows.

▶ **Definition 6** (Potential function). *Given a digraph $D = (V, E)$, we define a* potential function *on $V(D)$, $\phi : V(D) \to \{0.5, 1\}$ such that $\phi(v) = 1$, if v is a potential vertex to become a sink in an optimal solution and $\phi(v) = 0.5$, if v is a non-sink vertex. For any subset $V' \subseteq V(D)$, $\phi(V') = \sum_{x \in V'} \phi(x)$. We call a vertex v as an* undecided *vertex if $\phi(v) = 1$ and a* semi-decided *vertex if $\phi(v) = 0.5$.*

To solve the Knot-free Vertex Deletion problem on a digraph D, we initialize the potential values of all vertices to 1. As soon as we decide a vertex to be a non-sink vertex, we drop its potential by 0.5. Any vertex whose potential is 0.5 can not become a sink in the final knot-free graph resulting from removing an optimal vertex deletion set from D.

▶ **Definition 7** (Feasible solution). *A set $S \subseteq V(D)$ is called a feasible solution for (D, ϕ) if $D - S$ is knot-free and for any sink vertex s in $D - S$, $\phi(s) = 1$.* KFVD(D, ϕ) *is the size of an optimal solution for (D, ϕ).*

▶ **Reduction Rule 8.** *If all the vertices in D are semi-decided and D has no source or sink vertices, then $V(D)$ is contained inside any feasible solution for (D, ϕ).*

Proof. If S is a solution for (D, ϕ), then $D - S$ is knot-free. Since all vertices of D are semi-decided, $D - S$ has no sink vertices. Therefore, $D - S$ is an empty graph. In other words, $S = V(D)$. ◀

Let S_{opt} and Z_{opt} be the set of deleted vertices and the set of sink vertices with respect to some optimal solution KFVD(D, ϕ).

▷ Claim 9. If $x \in Z_{opt}$, then $N^+(x)$ is in S_{opt} and $S_{opt} \cap R^-(x) = \emptyset$.

Proof. By the definition of a sink vertex, if $x \in Z_{opt}$ then $N^+(x)$ is in S_{opt}. Let $Y = S_{opt} \cap R^-(x)$. We claim $S' = S_{opt} \setminus Y$ is also a solution which will contradict the fact that S_{opt} is an optimal solution. Suppose S' is not a solution, then there exists a vertex v in $G \setminus S'$, that do not reach to a sink s in $G \setminus S'$ but v reaches to some sink s in $G \setminus S_{opt}$. Since every vertex in Y reaches to some sink x in $G \setminus S'$, $v \notin Y$. If s is not a sink vertex in $G \setminus S'$, then some vertex $y \in Y$ is an out-neighbor of s. But then v can reach the sink x in $G \setminus S'$ via y. Hence, S' is a solution of strictly smaller size than S_{opt}, which is a contradiction. ◁

▷ Claim 10. If $x \in Z_{opt}$, then $|S_{opt}| = |N^+(x)| + \mathsf{KFVD}(D - R(x), \phi)$.

Proof. First, we prove that $S' = S_{opt} \setminus N^+(x)$ is a solution to $(D - R(x), \phi)$. If this is not true, then there exists a vertex v in $D - (R(x) \cup S')$ that do not reach any sink. Since $S_{opt} \subseteq R(x) \cup S'$, $v \notin S_{opt}$. But S_{opt} is a solution to (D, ϕ) and v can reach some sink $s \in Z_{opt}$ in $D - S_{opt}$. First we claim that $s \neq x$. Note that v is not in $R(x)$ and it can only reach x via some vertex in $N^+(x)$. So v can not reach x in $D - S_{opt}$ as $N^+(x) \subseteq S_{opt}$. Hence $s \neq x$. Then v has a path to s which is disjoint from $S_{opt} \supseteq N^+(x)$. Moreover this path is also disjoint from the set $R(x) \setminus N^+(x)$, since $v \notin (R(x)$. Hence this path is disjoint from $R(x)$ and S_{opt}. Therefore, v can reach s via the same path in $D - (R(x) \cup S')$. If s is a sink in $D - S_{opt}$, then s is also a sink in $(D - R(x)) \setminus S'$. Hence, v has a path to a sink in D, which is a contradiction. It implies that $S' = S_{opt} \setminus N^+(x)$ is a solution to $(D - R(x))$ and therefore, $|S_{opt}| - |N^+(x)| \geq \mathsf{KFVD}(D - R(x), \phi)$.

To prove the other direction, assume that S'' is an optimal solution to $(D - R(x), \phi)$. We claim that $S' = S'' \cup N^+(x)$ is a solution for (D, ϕ). Suppose not, then there exists a vertex v that does not reach a sink in $D - S'$. Note that $v \notin R(x)$ as all vertices in $R(x)$ can reach sink x in $D - S'$. Then $v \notin R(x) \cup S'$ and hence it is also in $D - (R(x) \cup S'')$ and it reaches a sink s. Since this path is disjoint from $S'' \cup R(x)$, v still can reach s via the same path in $D - S'$. Note that s is not a sink in $D - S'$ only if it has an out-neighbor in $R(x) \setminus N^+(x)$. But then s and v are in $R(x)$ which is not possible. Hence, v can still reach the same sink s and $S' = S'' \cup N^+(x)$ is a solution to D. Thus, $S_{\mathsf{opt}} \leq |N^+(x)| + \mathsf{KFVD}(D - R(x))$. ◁

3 An algorithm to compute minimum knot-free vertex deletion set

In this section, we design an exact algorithm to compute a minimum knot-free vertex deletion set. As mentioned earlier, we start by initializing the potential values of all vertices to 1. As soon as we decide a vertex is a non-sink vertex, we drop its potential by 0.5. In the following algorithm, at any step, if there are vertices with no out-neighbors (sinks) or no in-neighbors (sources), we remove such vertices with the help of Reduction Rules 2 and 3. If all the vertices are semi-decided, we apply Reduction Rule 8 to solve the instance in polynomial time. At any step, if there is an undecided vertex x with $\phi(R(x)) \geq 3.5$), we branch on the possibility of x being a sink or non-sink in the optimal solution. Here we use the large potential drop in the branch where x is chosen to be a sink to our advantage and obtain a $(3.5, 0.5)$ branching. If there are no such vertices, we look for an undecided vertex x such that all its neighbors are semi-decided, and we branch on x. If there are no such vertices, we branch on undecided vertices of degrees 2 and 3. Notice that in these branching steps, even if there is not a large potential drop when x is a sink, in the other branch when x is a non-sink vertex, we still find an undecided vertex s close to x, which is forced to be a sink. Hence the potential drop together in both the branches ensure *good* running time for our algorithm. If all undecided vertices have only one neighbor each, then we remove such vertices with the help of Reduction Rules 2 and 3.

Algorithm 1 KFVD (D, ϕ).

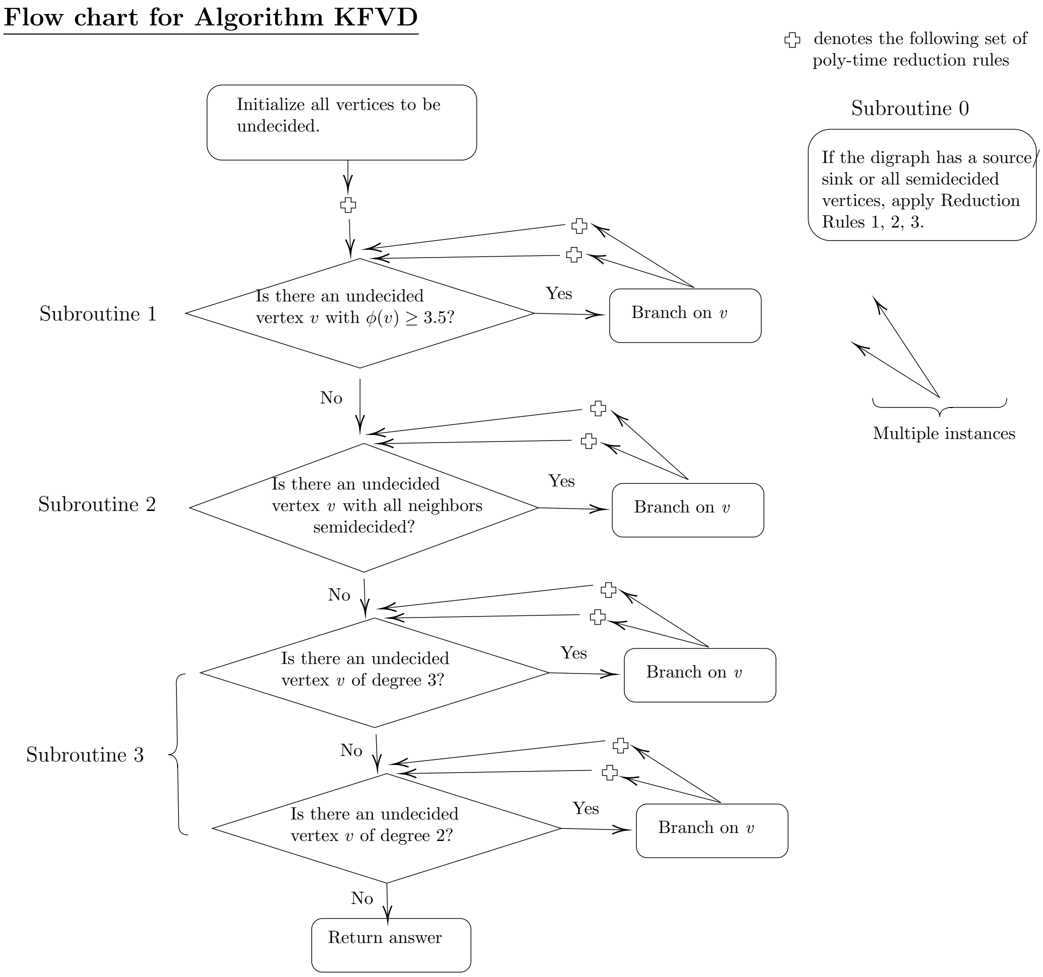
```
    Input: A directed graph $D$ and a potential function $\phi$
    Output: Size of a minimum knot-free vertex deletion set
 1  if $\exists$ $x$ such that $N^-(x) = \emptyset$ then
 2      return KFVD($D - \{x\}, \phi$);
 3  if $\exists$ $x$ such that $N^+(x) = \emptyset$ then
 4      return KFVD($D - R(x), \phi$);
 5  if $\forall v \in D$, $\phi(v) = 0.5$ then
 6      return $|V(D)|$;
 7  if $\exists$ $x \in D$ such that $\phi(x) = 1$ & $\phi(R(x)) \geq 3.5$ then
 8      $D_1 = D - R(x)$;
 9      $D_2 = D$;
10      $\phi_1 = \phi$;
11      $\phi_2 = \phi$;
12      $\phi_2(x) = 0.5$;
13      return min{KFVD($D_1, \phi_1$) + $|N^+(x)|$, KFVD($D_2, \phi_2$)};
14  if $\exists$ $x \in D$ such that $\phi(x) = 1$ & $\forall v \in N(x)$, $\phi(v) = 0.5$ then
15      for $y \in N^-(x)$ & $z \in N^+(x)$ do
16          if $\exists$ $s$ such that $s \in N^-(y) \cap N^+(z)$ & $\phi(s) = 1$ then
17              $D_1 = D - R(x)$;
18              $\phi_1 = \phi$;
19              $D_2 = D - R(s)$;
20              $\phi_2 = \phi$;
21              return min{KFVD($D_1, \phi_1$) + $|N^+(x)|$, KFVD($D_2, \phi_2$) + $|N^+(s)|$};
22          else
23              $D_1 = D - R(x)$;
24              $\phi_1 = \phi$;
25              return KFVD($D_1, \phi_1$) + $|N^+(x)|$;
26  if $\exists x \in D$ such that $|N(x)| \in \{2, 3\}$, $\phi(x) = 1$ & $\exists$ a unique $s \in N(x)$ with $\phi(s) = 1$ then
27      $D_1 = D - R(x)$;
28      $\phi_1 = \phi$;
29      $D_2 = D - R(s)$;
30      $\phi_2 = \phi$;
31      return min{KFVD($D_1, \phi_1$) + $|N^+(x)|$, KFVD($D_2, \phi_2$) + $|N^+(s)|$};
32  if $\exists$ $x \in D$ with $N^-(x) = \{y\}$, $N^+(x) = \{z\}$ and $\phi(x) = \phi(y) = \phi(z) = 1$ then
33      $D_1 = D - R(x)$;
34      $\phi_1 = \phi$;
35      $D_2 = D - R(y)$;
36      $\phi_2 = \phi$;
37      $D_3 = D - R(z)$;
38      $\phi_3 = \phi$;
39      return
          min{KFVD($D_1, \phi_1$) + $|N^+(x)|$, KFVD($D_2, \phi_2$) + $|N^+(y)|$, KFVD($D_3, \phi_3$) + $|N^+(z)|$};
```

4 Correctness of the Algorithm

In this section, we prove that the Algorithm KFVD returns a knot-free vertex deletion set of minimum size. We denote the steps from the lines 1-6 of the algorithm as *Subroutine* 0. The correctness of the lines 1-4 follows from the Reduction Rules 2, and 3 and the correctness of the lines 5-6 follows from the Reduction Rule 8. We call the steps from the lines 7-13, 14-25 and 26-39 of Algorithm KFVD as *Subroutine* 1, *Subroutine* 2, and *Subroutine* 3, respectively.

In the subsequent subsections, we will prove the correctness of Subroutines 1, 2, and 3.

Below we provide a flowchart for Algorithm KFVD.


Flow chart for Algorithm KFVD
✣ denotes the following set of poly-time reduction rules
Subroutine 0
If the digraph has a source/ sink or all semidecided vertices, apply Reduction Rules 1, 2, 3.
Multiple instances
Initialize all vertices to be undecided.
Subroutine 1
Is there an undecided vertex v with $\phi(v) \geq 3.5$?
Yes
Branch on v
No
Subroutine 2
Is there an undecided vertex v with all neighbors semidecided?
Yes
Branch on v
No
Subroutine 3
Is there an undecided vertex v of degree 3?
Yes
Branch on v
No
Is there an undecided vertex v of degree 2?
Yes
Branch on v
No
Return answer

4.1 Correctness of Subroutine 1

Subroutine 1 branches on an undecided vertex $x \in V(D)$, if $\phi(R(x)) \geq 3.5$. Notice that for any solution S in polynomial time we can determine the set of sink vertices, say Z_S, corresponding to set S.

▶ **Lemma 11.** *If $x \in V(D)$ such that $\phi(x) = 1$ and $\phi(R(x)) \geq 3.5$, then* $\mathsf{KFVD}(D, \phi) = \min\{\mathsf{KFVD}(D_1, \phi_1) + |N^+(x)|,\ \mathsf{KFVD}(D_2, \phi_2)\}$ *where $D_1 = D - R(x)$, $D_2 = D$ and $\phi_1 = \phi$, $\phi_2(v) = \phi(v)$, for all $v \in V(D) \setminus \{x\}$ and $\phi_2(x) = 0.5$.*

Proof. We use induction on the total potential $\phi(V(D))$ of the digraph D to prove the lemma. For the base case, assume that $\phi(V(D)) = 1$ and there exists exactly one undecided vertex x. Note that the given recurrence relation holds in this case. Next assume that $\mathsf{KFVD}(D_1, \phi_1)$ and $\mathsf{KFVD}(D_2, \phi_2)$ compute the optimal solution correctly, say S_1 and S_2, respectively. Let S be an optimal solution for $\mathsf{KFVD}(D, \phi)$. We consider the following two cases:

Case 1: $\boldsymbol{x \in Z_S}$.

Here we claim that $S_1 \cup N^+(x)$ is an optimal solution for $\mathsf{KFVD}(D,\phi)$ if and only if S_1 is an optimal solution for $\mathsf{KFVD}(D_1,\phi_1)$. The arguments are exactly the same as that in Claim 6. And by construction, $Z_S \in \phi^{-1}(1)$ if and only if $Z_S \in \phi_1^{-1}(1)$ where $S' = S \setminus N^+(x)$. We also claim that $\mathsf{KFVD}(D_2,\phi_2) \geq \mathsf{KFVD}(D_1,\phi_1) + |N^+(x)|$. For contradiction suppose KFVD $(D_2,\phi_2) < \mathsf{KFVD}(D_1,\phi_1) + |N^+(x)|$. Then any optimal solution S_2 for (D_2,ϕ_2) is also a feasible solution for $\mathsf{KFVD}(D,\phi)$. But S_2 has size strictly smaller than S, which is a contradiction. Hence, $\mathsf{KFVD}(D,\phi) = \min\{\mathsf{KFVD}$ $(D_1,\phi) + |N^+(x)|,$ $\mathsf{KFVD}(D,\phi_2)\}$.

Case 2: $\boldsymbol{x \notin Z_S}$.

In this case, $\mathsf{KFVD}(D_2,\phi_2) = \mathsf{KFVD}(D,\phi)$, by definition. Also $\mathsf{KFVD}(D_2,\phi_2) \leq \mathsf{KFVD}(D_1,\phi_1) + |N^+(x)|$, otherwise we have $S' = S_1 \cup N^+(x)$ as a solution with size strictly smaller than S for $\mathsf{KFVD}(D,\phi)$, where $x \in Z_{S'}$, and that is a contradiction. Hence, $\mathsf{KFVD}(D,\phi) = \min\{\mathsf{KFVD}(D_1,\phi) + |N^+(x)|,$ $\mathsf{KFVD}(D,\phi_2)\}$. ◀

In Subroutine 1, we get a $(3.5, 0.5)$ branching. If Subroutine 1 is no longer applicable on the instance, the digraph has no undecided vertices with $\phi(R(x)) \geq 3.5$. This fact is crucial to obtaining desirable branching vectors for Subroutine 2 and Subroutine 3.

▶ Remark 12. After Subroutine 1 completion, there are no undecided vertices with degree more than 4.

4.2 Correctness of Subroutine 2

Subroutine 2 branches on any undecided vertex x who has semi-decided neighbors only. Note that x has at least one out-neighbor and at least one in-neighbor; otherwise, reduction rules 2 or 3 would have been applied.

▷ Claim 13. Let D be a digraph such that $\phi(R(v)) < 3.5$, for every vertex $v \in V(D)$ and x be an undecided vertex in D such that for all $v \in N(x)$, $\phi(v) = 0.5$. If x is a non-sink vertex in an optimal solution S for (D,ϕ), then there exist vertices $y \in N^-(x)$ and $z \in N^+(x)$ such that there exists a unique $s \in N^-(y) \cap N^+(z)$ with $\phi(s) = 1$ and s is a sink vertex in $D - S$.

Proof. If x is not a sink vertex, then there exists a vertex $z \in N^+(x)$ such that it does not belong to S. Therefore, in digraph $D - S$ the vertex z must reach to a sink s i.e., there exists a path between z and the undecided vertex s. It implies that the vertices $x,\ z,\ s$ belong to $R(s)$. Therefore, $\phi(R(s)) \geq 3.5$ as $\phi(x) = \phi(s) = 1$, which is not possible. It implies that $s \in N^+(z)$ with $\phi(s) = 1$. Since vertex x must have one in-neighbor, say $y \in N^-(x)$. Next, s can not have an out-neighbor outside the set $\{x,y,z\}$, otherwise $\phi(R(x)) \geq 3.5$ which is not possible. Moreover, x or z cannot be an out-neighbor of s as x has no undecided out-neighbor and z does not belong to S_{opt}. Hence, $y \in N^+(s)$ which implies that $s \in R(x)$. Since $\phi(R(x)) < 3.5$, there does not exist any vertex $u \in N^-(y)$ except s such that $\phi(u) = 1$. Hence, s is a unique vertex such that $s \in N^-(y) \cap N^+(z)$ with $\phi(s) = 1$. ◁

Next, we prove the correctness of Subroutine 2.

▶ **Lemma 14.** *Let D be a digraph such that $\phi(R(v)) < 3.5$, for every vertex $v \in V(D)$ and $x \in V(D)$ be such that $\phi(x) = 1$ and for all $v \in N(x)$, $\phi(v) = 0.5$. If $y \in N^-(x)$ and $z \in N^+(x)$, then*

$$\mathsf{KFVD}(D,\phi) = \begin{cases} \min\Big\{\mathsf{KFVD}(D_1,\phi_1) + |N^+(x)|, \\ \qquad \mathsf{KFVD}(D_2,\phi_2) + |N^+(s)|\Big\} & \text{if } \exists\ s \in N^-(y) \cap N^+(z) \text{ with } \phi(s) = 1, \\ \\ \mathsf{KFVD}(D_1,\phi_1) + |N^+(x)| & \text{otherwise,} \end{cases}$$

where $D_1 = D - R(x)$*,* $D_2 = D - R(s)$ *and* $\phi_1 = \phi_2 = \phi$*.*

Proof. Let S, S_1 ans S_2 be the optimal solutions for $\mathsf{KFVD}(D, \phi)$, $\mathsf{KFVD}(D_1, \phi_1)$ and $\mathsf{KFVD}(D_2, \phi_2)$, respectively. We use induction on the total potential $\phi(V(D))$ of the digraph to prove the correctness of the above claim. Let $y \in N^-(x)$ and $z \in N^(x)$. Now, we consider the following two cases:

Case 1: $\boldsymbol{\exists s \in N^-(y) \cap N^+(z)}$ **such that** $\boldsymbol{\phi(s) = 1}$**.**

Assume that there exists $s \in N^-(y) \cap N^+(z)$ with $\phi(s) = 1$. From Claim 13, we know that either x or s will be a sink in an optimal solution. Let $x \in Z_S$, then from Claim 10, $|S| = \mathsf{KFVD}(D_1, \phi_1) + |N^+(x)|$. And $S_2 \cup N^+(s)$ is a feasible solution to $\mathsf{KFVD}(D, \phi)$ and hence $\mathsf{KFVD}(D_2, \phi_2) + |N^+(s)| \geq |S| = \mathsf{KFVD}(D_1, \phi_1) + |N^+(x)|$. Similarly, if $s \in Z(S)$ then from Claim 10, $|S| = \mathsf{KFVD}(D_2, \phi_2) + |N^+(s)|$. Note that $S_1 \cup N^+(x)$ is also a feasible solution to $\mathsf{KFVD}(D, \phi)$ and therefore, $\mathsf{KFVD}(D_1, \phi_1) + |N^+(x)| \geq |S| = \mathsf{KFVD}(D_2, \phi_2) + |N^+(s)|$. Hence, we have $\mathsf{KFVD}(D, \phi) = \min\{\mathsf{KFVD}(D_1, \phi_1) + |N^+(x)|, \mathsf{KFVD}(D_2, \phi_2) + |N^+(s)|\}$.

Case 2: $\boldsymbol{\nexists s \in N^-(y) \cap N^+(z)}$ **such that** $\boldsymbol{\phi(s) = 1}$**.**

Suppose there does not exist any vertex $s \in N^-(y) \cap N^+(z)$ such that $\phi(s) = 1$. Observe that in this case, the vertex x has to be a sink; otherwise, by Claim 13 there exists a vertex $s \in N^-(y) \cap N^+(z)$ with $\phi(s) = 1$, which is a contradiction. Therefore, the vertex x has to be a sink. Hence, $\mathsf{KFVD}(D, \phi) = \mathsf{KFVD}(D_1, \phi_1) + N^+(x)$ by Claim 10, where $D_1 = D - R(x)$. ◀

▶ Remark 15. If Subroutine 2 is no longer applicable, the instance has no undecided vertices with a degree more than 3.

4.3 Correctness of Subroutine 3

Subroutine 3 branches on undecided vertices with degrees 2 and 3. When Subroutine 0, 1, and 2 are no longer applicable, there are no vertices with all semi-decided neighbors. Also there is no undecided vertex x with degree 4 as $\phi(R(x)) \geq 3.5$. So any undecided vertex has degree at most 3. Moreover, any undecided vertex x of degree 3 has exactly one undecided vertex. We will prove the correctness of branching in Subroutine 3 in two parts. First, we analyze the branching vector for degree 3 undecided vertices and then for degree 2 undecided vertices.

Branching on a degree 3 vertex

Given an undecided vertex x of degree 3, there are two cases; either x has one in-neighbor and two out-neighbors or x has two in-neighbors and one out-neighbor.

Case 1: x has one in-neighbor y_1 and two out-neighbors z_1, z_2

Out of three neighbors, x has exactly one neighbor which is undecided. So we have the following 3 three subcases.

Subcase 1: $\boldsymbol{\phi(y_1) = 1}$ **and** $\boldsymbol{\phi(z_1) = \phi(z_2) = 0.5}$**.**

- If x is a sink, there is a potential drop of at least 3 since $\{y_1, z_1, z_2, x\} \in R(x)$.
- If x is not a sink, then x and some $z \in N^+(x)$ are not in the deletion set.Without loss of generality, let $z = z_1$. Now z_1 reaches a sink $s \in Z_{opt}$ in $D - S_{opt}$. If z_1 can reach a sink s ($\neq y_1$), then $\{y_1, x, z_1, s\} \subseteq R(s)$. This is true since s can not have x or z_1 as its out-neighbor and either y_1 is an out-neighbor of s or $y_1 \in R^-(x)$. It implies that

$\phi(R(s)) \geq 3.5$, which is a contradiction. Hence y_1 is the only sink that z_1 must reach. Note that $\phi(R(y_1)) \geq 2.5$.

This gives us a $(3, 2.5)$ branching vector.

Subcase 2: $\phi(y_1) = 0.5$, $\phi(z_1) = 1$ and $\phi(z_2) = 0.5$.

- If x is a sink, the potential drops by at least 3.
- If x is not a sink and z_1 is not in the deletion set, then z_1 reaches a sink say, s in Z_{opt} in $D - S_{opt}$. If z_1 can reach a sink s ($\neq z_1$), then $\{y_1, x, z_1, s\} \subseteq R(s)$ as s can not have x , z_1 or y_1 as its out-neighbors. Therefore, $\phi(R(s)) \geq 3.5$ which is a contradiction. Therefore z_1 has to be a sink when x is not a sink. Note that $\phi(R(z_1)) \geq 2.5$.

This gives us a $(3, 2.5)$ branching vector.

Subcase 3: $\phi(y_1) = \phi(z_1) = 0.5$ and $\phi(z_2) = 1$.

- if x is a sink, again the potential drops by at least 3 in (D_1, ϕ_1).
- If x is not a sink and z_1 is not in the deletion set, then z_1 reaches s in Z_{opt} in $D - S_{opt}$. Suppose z_1 can reach a sink, say s ($\neq z_2$). If z_1 has an out-neighbor outside $\{y_1, x, z_1\}$ then $\phi(R(s)) \geq 3.5$, which is a contradiction. Otherwise y_1 is its only out-neighbor. But then $\{s, y_1, x, z_1, x, z_2\} \subseteq R(x)$ and $\phi(R(s)) \geq 4$, which is not possible. Therefore, the only possibility for z_2 is to be the sink for z_1 when x is not a sink. Note that $\phi(R(z_2)) \geq 2.5$.

Hence we have a $(3, 2.5)$ branching vector.

Case 2: x has two in-neighbors y_1, y_2 and one out-neighbor z_1

Subcase 1: $\phi(y_1) = \phi(y_2) = 0.5$ and $\phi(z) = 1$.

- If x is a sink, there is a potential drop of at least 3.
- If x is not a sink, then z_1 is not in the deletion set. Note that z_1 reaches a sink, say $s \in Z_{opt}$ in $D - S_{opt}$. If z_1 can reach a sink s ($\neq z_1$), then $\{y_1, x, z_1, s\} \subseteq R(s)$. But then $\phi(R(s)) \geq 3.5$ which is a contradiction. The only possibility is for z_1 itself to be a sink when x is not a sink. Note that $\phi(R(z_1)) \geq 2.5$.

Hence we have a $(3, 2.5)$ branching vector.

Subcase 2: $\phi(y_1) = 1$ and $\phi(y_2) = \phi(z_1) = 0.5$.

- If x is a sink, the potential drops by 3.
- If x is not a sink, then z_1 is not in the deletion set. Therefore, z_1 reaches a sink s. If there is a path from z_1 to s not using y_1, y_2, then $\phi(R(s)) \geq 4$, which is not possible. Otherwise if there is a path from z_1 to s via y_1 or y_2, then $\phi(R(s)) \geq 4$, which is again not possible. Hence, the only possibility is that y_1 becomes a sink such that z_1 reaches y_1. Note that in this case, the potential drops by at least 2.5.

Hence we have a $(3, 2.5)$ branching vector in this case.

Branching on a degree 2 vertex

This subsection analyzes the potential drop while branching on a degree 2 undecided vertex x, which has an undecided neighbor. An undecided vertex x of degree 2 has exactly one in-neighbor and one out-neighbor. Let $y \in N^-(x)$ and $z \in N^+(x)$. We consider the following three cases.

Case 1: $\phi(y) = 1$ and $\phi(z) = 0.5$.

- If x is a sink, the potential drops by 2.5.
- If x is not a sink, then z is not in the deletion set and it reaches a sink s. If $s \neq y$, then $\{y, x, z, s\} \subseteq R(s)$ and $\phi(R(s)) \geq 3.5$, which is not possible. Hence, the only possibility is that y is a sink that z reaches. Note that here the potential drops by 2.5.

Hence, we get a $(2.5, 2.5)$ branching vector.

Case 2: $\phi(y) = 0.5$ and $\phi(z) = 1$.

- If x is a sink, the potential drops by 2.5.
- If x is not a sink, then z is not in the deletion set. Therefore, z reaches a sink, say s. If $s \neq y$ then $\{y, x, z, s\} \subseteq R(s)$ and $\phi(R(s)) \geq 3.5$, which is not possible. Hence, the only possibility is for z to be a sink itself. Note that in this case, the potential drops by 2.5

Hence we have a $(2.5, 2.5)$ branching vector.

Case 3: $\phi(y) = 1$ and $\phi(z) = 1$.

- If x is a sink, potential drops by 3.
- If x is not a sink, then z is not in the deletion set. Therefore, z reaches a sink, say s. If $s \neq y$ and $s \neq z$, then $\{y, x, z, s\} \subseteq R(s)$ and $\phi(R(s)) \geq 3.5$, which is not possible. Hence, the only possibility is for z to be a sink itself or to reach sink y. Note both in these cases, the potential drops by at least 3.

Hence we have a (3, 3, 3) branching vector.

Next, we give a formal proof of the correctness for Subroutine 3.

▶ **Lemma 16.** *Let D be a digraph such that $\phi(R(v)) < 3.5$, for every vertex $v \in V(D)$ and $x \in V(D)$ be such that $d(x) \in \{2, 3\}$, $\phi(x) = 1$ and there exists some vertex $s \in N(x)$ such that $\phi(s) = 1$. Then* $\mathsf{KFVD}(D, \phi) = \min\{\mathsf{KFVD}(D_1, \phi) + |N^+(x)|,\ \mathsf{KFVD}(D_2, \phi) + N^+(s)\}$ *where $D_1 = D - R(x)$, $D_2 = D - R(s)$ and $\phi_1 = \phi_2 = \phi$.*

Proof. Notice that there is no vertex v with $\phi(v) \geq 3.5$ and no vertex v that has all its neighbors with potential 0.5. We use induction on the potential function to prove the correctness of the claim. Let $\mathsf{KFVD}(D_1, \phi_1)$ and $\mathsf{KFVD}(D_2, \phi_2)$ correctly compute optimal solution (S_1, Z_1) and (S_2, Z_2), where $Z_i \subseteq \phi^{-1}(1)$. Let (S, Z) be an optimal solution for $\mathsf{KFVD}(D, \phi)$.

Case 1: $x \in Z$.

In this case, we know that $S_1 \cup N^+(x)$ is an optimal solution for $\mathsf{KFVD}(D, \phi)$ if and only if S_1 is an optimal solution for $\mathsf{KFVD}(D_1, \phi_1)$ and $|S_2 \cup N^+(s)| \geq |S_1 \cup N^+(x)|$, since it is also a solution to $\mathsf{KFVD}(D, \phi)$ that does not contain $x \in z$. Hence $\mathsf{KFVD}(D, \phi)$=$\min\{\mathsf{KFVD}(D_2, \phi_2) + |N^+(x)|,\ \mathsf{KFVD}(D_1, \phi_1) + |N^+(s)|\}$.

Case 2: $x \notin Z$.

By arguments made in the branching steps for degree 3 and 2 vertices, we can always find an s that must be a sink in Z. And hence $S_2 \cup N^+(s)$ is an optimal solution for for $\mathsf{KFVD}(D, \phi)$ if and only if S_1 is an optimal solution for $\mathsf{KFVD}(D_2, \phi_2)$. $|S_1 \cup N^+(x)| \geq |S_2 \cup N^+(s)|$ since it is also a solution to $\mathsf{KFVD}(D, \phi)$ that does contain $x \in z$. Hence $\mathsf{KFVD}(D, \phi)$=$\min\{\mathsf{KFVD}(D_2, \phi_2) + |N^+(x)|,\ \mathsf{KFVD}(D_1, \phi_1) + |N^+(s)|\}$. ◀

5 Running time analysis

Reduction Rules 1, 2 and 3 are applied on the instance (D, ϕ) in $n^{\mathcal{O}(1)}$ time. In Subroutine 1, while branching we get a potential drop of at least 3.5 in one branch, while in the other branch the potential drop is at least 0.5. Hence, we get the recurrence $f(\mu) \leq f(\mu - 3.5) + f(\mu - .5)$, which solves to $f(\mu) = \mathcal{O}(1.576^\mu)$. Similarly for Subroutine 2, we have a branching vector $(2, 3)$. Therefore, we have the recurrence $f(\mu) \leq f(\mu - 2) + f(\mu - 3)$, which solves to $f(\mu) = \mathcal{O}(1.324^\mu)$. For Subroutine 3, we have branching vectors $(3, 2.5), (2.5, 2.5)$, and $(3, 3, 3)$, out of which $(3, 3, 3)$ gives the worst running time. This solves to $f(\mu) = \mathcal{O}(1.442^\mu)$.

To solve the Knot-free Vertex Deletion, when we call KFVD(D, ϕ), potentials of all the vertices of D are initialized to 1 and $\phi(V(D)) = |V(D)| = n$. Moreover, in the recursive calls to the algorithm, the potential of the input graph never drops below 0. Hence we can upper bound the running time of Algorithm KFVD by the worst-case running subroutine (Subroutine 1). Thus we obtain the following theorem.

▶ **Theorem 17.** *Algorithm* KFVD *solves* Knot-free vertex deletion *in* $\mathcal{O}^*(1.576^n)$ *time.*

6 A lower bound on the worst case running time of our algorithm

In this section, we give a lower bound on the worst-case running time of our algorithm.

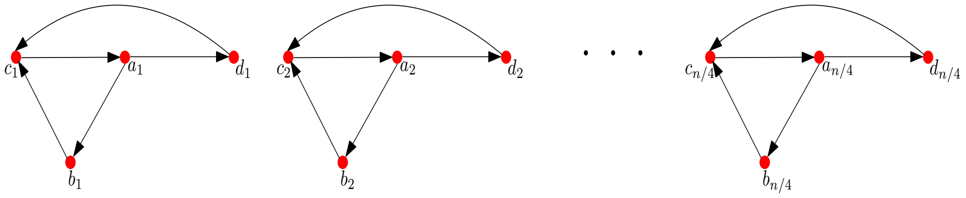


Figure 1 Illustration of a worst-case instance for our algorithm.

We run our algorithm KFVD on the graph D (shown in Figure 1), where $V(D) = \cup_{i=1}^{n/4}\{a_i, b_i, c_i, d_i\}$ and $E(D) = \cup_{i=1}^{n/4}\{(a_i, b_i), (b_i, c_i), (c_i, a_i), (a_i, d_i), (d_i, c_i)\}$. We claim that in the worst case, our algorithm takes $\mathcal{O}*(2^{n/2})$ time to solve KFVD on D. We will give this lower bound via adversarial arguments.

Let $V_i = \{a_i, b_i, c_i, d_i\}$ where $i \in [1, n/4]$. Since the potentials of all the vertices are initialized to 1, we have $\phi(R(a_i)) = 4$, $\phi(R(b_i)) = 3$, $\phi(R(c_i)) = 4$, $\phi(R(d_i)) = 3$. The adversary chooses the vertex a_i for KFVD to branch on. If a_i is a sink, all the four vertices are deleted. If a_i is a non-sink vertex, its potential drops by 0.5 while all other vertices' potentials remain unchanged. In the next iteration, $\phi(R(c_i)) = 3.5$, so we branch on c_i. Again, if c_i is a sink then all four vertices get deleted. If c_i is not a sink, then its potential drops by 0.5. In the next step, b_i and d_i are vertices where all their neighbors are semi-decided. The adversary chooses b_i to branch on. If b_i becomes a sink all the vertices are again removed. Else, when b_i is a non-sink d_i has to be a sink and all vertices are removed. There are four leaves of the branching tree while branching on the set of vertices $\{a_i, b_i, c_i, d_i\}$. This gives us a recurrence equation: $T(n) = 4T(n-4)$ which solves to $4^{n/4} = 1.414^n$.

▶ **Theorem 18.** *Algorithm* KFVD *runs in time* $\Omega(1.414^n)$.

7 Conclusion

We obtain a $\mathcal{O}(1.576^n)$ time and polynomial space algorithm for KFVD problem. Notice that our algorithm is not optimal and its improvement is a suggested direction for future work. Also exact algorithms for KFVD with dependency on number of edges instead of vertices can be an interesting research topic.

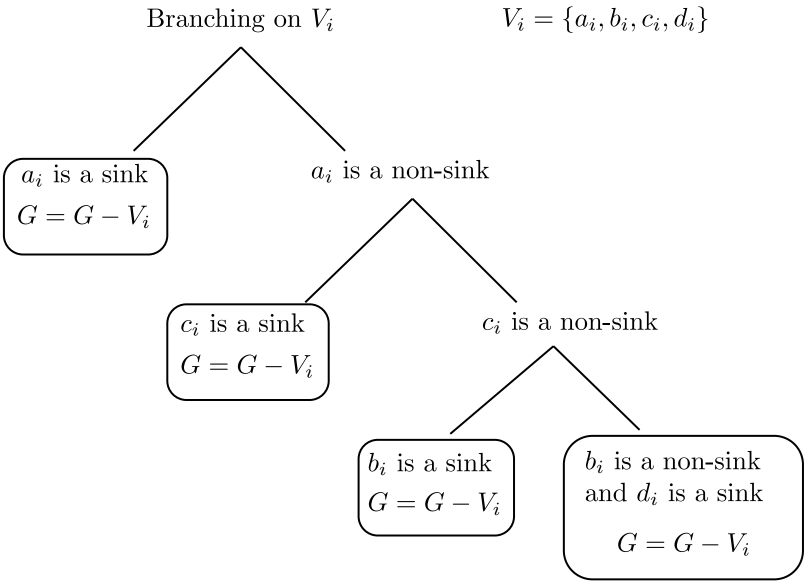


Figure 2 Branching steps on V_i.

References

1 Valmir Carneiro Barbosa, Alan Diêgo A. Carneiro, Fábio Protti, and Uéverton S. Souza. Deadlock models in distributed computation: foundations, design, and computational complexity. In Sascha Ossowski, editor, *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, pages 538–541. ACM, 2016. doi:10.1145/2851613.2851880.

2 Stéphane Bessy, Marin Bougeret, Alan Diêgo A. Carneiro, Fábio Protti, and Uéverton S. Souza. Width parameterizations for knot-free vertex deletion on digraphs. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 2:1–2:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.2.

3 Andreas Björklund. Determinant sums for hamiltonicity (invited talk). In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 1:1–1:1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.1.

4 Alan Diêgo Aurélio Carneiro, Fábio Protti, and Uéverton dos Santos Souza. On knot-free vertex deletion: Fine-grained parameterized complexity analysis of a deadlock resolution graph problem. *Theoretical Computer Science*, 909:97–109, 2022.

5 Alan Diêgo Aurélio Carneiro, Fábio Protti, and Uéverton S. Souza. Deletion graph problems based on deadlock resolution. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics – 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings*, volume 10392 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 2017. doi:10.1007/978-3-319-62389-4_7.

6 Alan Diêgo Aurélio Carneiro, Fábio Protti, and Uéverton S. Souza. Deadlock resolution in wait-for graphs by vertex/arc deletion. *J. Comb. Optim.*, 37(2):546–562, 2019. doi:10.1007/s10878-018-0279-5.

7 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. 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 764–775. ACM, 2016. doi:10.1145/2897518.2897551.

10 Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *Proceedings of the 30th International Conference on Graph-Theoretic Concepts in Computer Science*, WG'04, pages 245–256, Berlin, Heidelberg, 2004. Springer-Verlag. doi:10.1007/978-3-540-30559-0_21.

11 F.V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010. URL: https://books.google.co.in/books?id=LXDe1XHJCYIC.

12 Gordon Hoi. An improved exact algorithm for the exact satisfiability problem. *CoRR*, abs/2010.03850, 2020. arXiv:2010.03850.

13 Kamal Jain, Mohammad Taghi Hajiaghayi, and Kunal Talwar. The generalized deadlock resolution problem. 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 853–865. Springer, 2005. doi:10.1007/11523468_69.

14 Carlos VGC Lima, Fábio Protti, Dieter Rautenbach, Uéverton S Souza, and Jayme L Szwarcfiter. And/or-convexity: a graph convexity based on processes and deadlock models. *Annals of Operations Research*, 264(1):267–286, 2018.

15 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. 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 1916–1933. SIAM, 2018. doi:10.1137/1.9781611975031.125.

16 Alane Marie de Lima and Renato Carmo. Exact algorithms for the graph coloring problem. *Revista de Informática Teórica e Aplicada*, 25:57, November 2018. doi:10.22456/2175-2745.80721.

17 Fabiano de S Oliveira and Valmir C Barbosa. Revisiting deadlock prevention: A probabilistic approach. *Networks*, 63(2):203–210, 2014.

On Extended Boundary Sequences of Morphic and Sturmian Words

Michel Rigo
Department of Mathematics, University of Liège, Belgium

Manon Stipulanti
Department of Mathematics, University of Liège, Belgium

Markus A. Whiteland
Department of Mathematics, University of Liège, Belgium

Abstract

Generalizing the notion of the boundary sequence introduced by Chen and Wen, the nth term of the ℓ-boundary sequence of an infinite word is the finite set of pairs (u, v) of prefixes and suffixes of length ℓ appearing in factors uyv of length $n + \ell$ ($n \geq \ell \geq 1$). Otherwise stated, for increasing values of n, one looks for all pairs of factors of length ℓ separated by $n - \ell$ symbols.

For the large class of addable numeration systems U, we show that if an infinite word is U-automatic, then the same holds for its ℓ-boundary sequence. In particular, they are both morphic (or generated by an HD0L system). We also provide examples of numeration systems and U-automatic words with a boundary sequence that is not U-automatic. In the second part of the paper, we study the ℓ-boundary sequence of a Sturmian word. We show that it is obtained through a sliding block code from the characteristic Sturmian word of the same slope. We also show that it is the image under a morphism of some other characteristic Sturmian word.

2012 ACM Subject Classification Theory of computation → Regular languages; Mathematics of computing → Combinatorics on words

Keywords and phrases Boundary sequences, Sturmian words, Numeration systems, Automata, Graph of addition

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.79

Related Version *Full Version*: `https://arxiv.org/abs/2206.15319`

Funding *Manon Stipulanti*: Supported by the FNRS Research grant 1.B.397.20F.
Markus A. Whiteland: Supported by the FNRS Research grant 1.B.466.21F.

Acknowledgements We thank Jean-Paul Allouche for references [10, 30, 31], and Jeffrey Shallit for discussions about the "logical approach". The anonymous referees are warmly thanked for providing useful feedback improving the quality of the text.

1 Introduction

Let $\mathbf{x}$ be an infinite word, i.e., a sequence of letters belonging to a finite alphabet. Imagine a window of size n moving along $\mathbf{x}$. Such a reading frame permits to detect all factors of length n occurring in $\mathbf{x}$. For instance, the factor complexity function of $\mathbf{x}$ mapping $n \in \mathbb{N}$ to the number of distinct factors of length n is extensively studied in combinatorics on words. Now let n, ℓ be such that $n \geq \ell$. Assume that within the sliding window, we only focus on its first and last ℓ symbols. Otherwise stated, for a factor uyv of length n, we only consider its borders u and v of length ℓ.

For any given window length n, we would like to determine what are the pairs of length-ℓ borders that may occur. This leads to the following definition, where, to simplify notation, we consider borders of factors of length $n + \ell$ rather than n.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 79; pp. 79:1–79:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

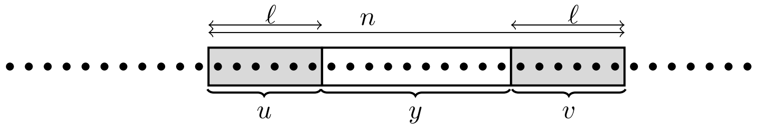


Figure 1 A sliding window where we focus on two regions of a fixed length.

▶ **Definition 1.1.** *Let $\ell \in \mathbb{N}_{>0}$ and $\mathbf{x} \in A^{\mathbb{N}}$. For $n \geq \ell$, we define the n*th boundary set *by*

$$\partial_{\mathbf{x},\ell}[n] := \{(u,v) \in A^\ell \times A^\ell \mid uyv \text{ is a factor of } \mathbf{x} \text{ for some } y \in A^{n-\ell}\}$$

*and call the sequence $\partial_{\mathbf{x},\ell} := (\partial_{\mathbf{x},\ell}[n])_{n\geq\ell}$ the ℓ-*boundary sequence *of $\mathbf{x}$. When $\ell = 1$, we write $\partial_{\mathbf{x},1} = \partial_{\mathbf{x}}$ and simply talk about the* boundary sequence.

The ℓ-boundary sequence takes values in $2^{A^\ell \times A^\ell}$, and hence itself can be seen as an infinite word over a finite alphabet. We give an introductory example.

▶ **Example 1.2.** Consider the Fibonacci word $\mathbf{f} = 0100101001\cdots$; the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 0$. We have $\partial_{\mathbf{f}} = a\,b\,b\,a\,b\,b\,b\,b\,a\,b\,b\,a\,b\,b\,b\,b\,a\,b\,b\,b\,b\,a\,b\,b\,a\,b\,b\,b\,b\cdots$, where $a := \{(0,0),(0,1),(1,0)\}$ and $b := \{0,1\} \times \{0,1\}$. For instance, $\partial_{\mathbf{f}}[1] = a$ because the length-2 factors of $\mathbf{f}$ are $00, 01, 10$, while $\partial_{\mathbf{f}}[2] = b$ because its length-3 factors are of the form 0_0, 0_1, 1_0, 1_1 (they are in fact $010, 001, 100, 101$). The 2-boundary sequence starts with

$$\partial_{\mathbf{f},2} = a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_1\ a_2\ a_3\ a_1\ a_2\ a_3\ a_4\ a_5\ a_1\ a_2\ a_3\ a_4\ a_5\ a_1\ a_2\ a_3\ a_1\ a_2\ a_3\cdots$$

where

$$\begin{aligned}
a_0 &:= \{(00,10),(01,00),(01,01),(10,01),(10,10)\},\\
a_1 &:= \{(00,00),(00,01),(01,01),(01,10),(10,00),(10,10)\},\\
a_2 &:= \{(00,01),(00,10),(01,00),(01,10),(10,00),(10,01)\},\\
a_3 &:= \{(00,00),(00,10),(01,00),(01,01),(10,01),(10,10)\},\\
a_4 &:= \{(00,01),(01,01),(01,10),(10,00),(10,01),(10,10)\},\\
a_5 &:= \{(00,10),(01,00),(01,01),(01,10),(10,01),(10,10)\}.
\end{aligned}$$

The first element $\partial_{\mathbf{f},2}[2] = a_0$ is peculiar; it corresponds exactly to the five length-4 factors occurring in $\mathbf{f}$. Our Proposition 4.8 shows that a_0 appears only once in $\partial_{\mathbf{f}}$. Then, e.g., $\partial_{\mathbf{f},2}[3] = a_1$ because the length-5 factors of $\mathbf{f}$ are of the form 00_00, 00_01, 01_01, 01_10, 10_00 and 10_10 (the factors are 00100, 00101, 01001, 10100, 10010, and 01010). For length-6 factors, note that two are of the form $10u01$ for some $u \in \{0,1\} \times \{0,1\}$. For $i \geq 1$, the letter a_i appears infinitely often in $\partial_{\mathbf{f},2}$: see Theorem 4.1.

1.1 Motivation and related work

In combinatorics on words, borders and boundary sets are related to important concepts. For instance, a word v is *bordered* if there exist u, x, y such that $v = ux = yu$ and $0 < |u| < |v|$. One reason to study bordered words is Duval's theorem: for a sufficiently long word v, the maximum length of unbordered factors of v is equal to the period of v [14]. In formal language theory, a language L is *locally ℓ-testable* (LT) if the membership of a word w in L only depends on the prefix, suffix and factors of length ℓ of w. In [35], the authors consider the so-called *separating problem* of languages by LT languages; they utilize *ℓ-profiles* of a word, which can again be related to boundary sets. Let us also mention that, in bioinformatics and computational biology, one of the aims is to reconstruct sequences from subsequences [26]. To determine DNA segments by bottom-up analysis, *paired-end* sequencing is used. In this case both ends of DNA fragments of known length are sequenced. See, for instance, [18]. This is quite similar to the theoretical concept we discuss here.

The notion of a (1-)boundary sequence was introduced by Chen and Wen in [8] and was further studied in [19], where it is shown that the boundary sequence of a *k-automatic* word (in the sense of Allouche and Shallit [1]: see Definition 2.2) is k-automatic. It is well-known that a k-automatic word $\mathbf{x}$ is *morphic*, i.e., there exist morphisms $f\colon A \to A^*$ and $g\colon A \to B$ and a letter $a \in A$ such that $\mathbf{x} = g(f^\omega(a))$, where $f^\omega(a) = \lim_{n\to\infty} f^n(a)$. However, k-automatic words (with k ranging over the integers) do not capture all morphic words: a well-known characterization of k-automatic words is given by Cobham [9] (the generating morphism f maps each letter to a length-k word). This paper is driven by the natural question whether, in general, the ℓ-boundary sequence of a morphic word is morphic. In case such generating morphisms can be constructed, we have at our disposal a simple algorithm providing the set of length-ℓ borders in factors of all lengths.

We briefly present several situations in which the notion of boundary sets is explicitly or implicitly used. In [12, Thm. 4], the authors study the boundary sequence to exhibit a squarefree word for which each subsequence arising from an arithmetic progression contains a square. Boundary sets play an important role in the study of so-called *k-abelian* and *k-binomial complexities* of infinite words (for definitions, see [37]). For instance, computing the 2-binomial complexity of generalized Thue–Morse words [25] requires inspecting pairs of prefixes and suffixes of factors, which is again related to the boundary sequence when these prefixes and suffixes have equal length. The k-binomial complexities of images of binary words under powers of the Thue–Morse morphism are studied in [39]; there some general properties of boundary sequences of binary words are required. Moreover, if $\partial_{\mathbf{x}}$ is automatic, then the abelian complexity of the image of $\mathbf{x}$ under a so-called Parikh-constant morphism is automatic [8]. Guo, Lü, and Wen combine this result with theirs in [19] to establish a large family of infinite words with automatic abelian complexity.

Let $k \geq 1$. We let $\equiv_k$ denote the k-abelian equivalence, i.e., $u \equiv_k v$ if the words u and v share the same set of factors of length at most k with the same multiplicities [22]. For u and v equal length factors of a Sturmian word $\mathbf{s}$, we have $u \equiv_k v$ if and only if they share a common prefix and a common suffix of length $\min\{|u|, k-1\}$ and $u \equiv_1 v$ [22, Prop. 2.8]. Under the assumption that the largest power of a letter appearing in $\mathbf{s}$ is less than $2k-2$, the requirement $u \equiv_1 v$ in the previous result may be omitted [33, Thm. 3.6] (compare to Proposition 4.8). Thus the quotient of the set of factors of length n occurring in a Sturmian word by the relation $\equiv_k$ is completely determined by $\partial_{\mathbf{s},k-1}[n-k+1]$ for large enough k (depending on $\mathbf{s}$). Other families of words with k-abelian equivalence determined by the boundary sets are given in [33, Prop. 4.2].

1.2 Our contributions

Up to our knowledge, we are the first to propose a systematic study of the ℓ-boundary sequences of infinite words. It is therefore natural to consider the notion on well-known classes of words. In this paper, we consider morphic words and Sturmian words.

Any morphic word is S-automatic for some abstract numeration system S [38]. With Theorem 3.1, we prove that for a large class of numeration systems U, if $\mathbf{x}$ is a U-automatic word, then the boundary sequence $\partial_{\mathbf{x}}$ is again U-automatic. Our approach generalizes the arguments provided by [19]. Considering exotic numeration systems allows a better understanding of underlying mechanisms, which do not arise in the ordinary integer base systems. In particular, we deal with addition within the numeration system; in integer base systems, the carry propagation is easy to handle (by a two-state finite automaton). Our arguments apply to so-called *addable* numeration systems (see Definition 2.1).

As an alternative, we observe that the Büchi–Bruyère theorem [5] can be extended to addable positional numeration systems U (Theorem 2.6). The U-automaticity of the ℓ-boundary sequence then follows from the fact that it is definable by a first-order formula of the structure $\langle \mathbb{N}, + \rangle$ extended with a unary function relating an integer with the least element of U properly appearing in its representation.

This alternative proof however hides the important details that might help identifying the technical limits of the result: not all morphic words allow an addable system to work with. However, the framework we consider captures all morphic words (see Theorem 2.4). Also, one practical difficulty when one wants to use automatic provers (such as Walnut [28]) is to be able to provide the relevant automaton for addition. To identify the contours of our result, we also discuss the case where $\mathbf{x}$ is U-automatic and $\partial_{\mathbf{x}}$ is not U-automatic. To construct such examples, we have to consider non-addable numeration systems in Section 3.2.

We then turn to the other class of words under study. Letting $\mathbf{s}$ be a Sturmian word with slope α, with Theorem 4.1 we show that the ℓ-boundary sequence of $\mathbf{s}$ is obtained through a sliding block code from the *characteristic Sturmian word of slope* α (see Section 4 for a definition) up to the first letter. This result holds even for non-morphic Sturmian words, so for an arbitrary irrational α. Where the techniques used in the first part of the paper have an automata-theoretic flavor, the second part relies on the geometric characterization of Sturmian words as codings of rotations. We provide another description of the ℓ-boundary sequence of a Sturmian word as the morphic image of some characteristic Sturmian word in Proposition 4.10. We remark that it is unclear to us whether some of the results in Section 4 can be proved automatically using the very recent tool developed in [21].

2 Preliminaries

Throughout this paper we let A denote a finite alphabet. Then A^n denotes the set of length-n words. For an infinite word $\mathbf{x}$, we let $\mathbf{x}[n]$ denote its nth letter, for all $n \geq 0$. For general references on numeration systems, see [16] and [4, Chap. 1–3]. We assume that the reader has some knowledge in automata theory. For a reference see [40] or [36, Chap. 1].

2.1 Numeration systems and automatic words

Let $U = (U_n)_{n \geq 0}$ be an increasing sequence of integers such that $U_0 = 1$. Any integer n can be decomposed (not necessarily uniquely) as $n = \sum_{i=0}^{t} c_i\, U_i$ with non-negative integer coefficients c_i. The finite word $c_t \cdots c_0 \in \mathbb{N}^*$ is a *U-representation* of n. If this representation is computed greedily [16, 36], then for all $j \leq t$ we have $\sum_{i=0}^{j} c_i\, U_i < U_{j+1}$ and $\operatorname{rep}_U(n) = c_t \cdots c_0$ is said to be the *greedy* (or *normal*) U-representation of n. By convention, the greedy representation of 0 is the empty word ε and the greedy representation of $n > 0$ starts with a non-zero digit. An extra condition on the boundedness of $\sup_{i \geq 0}(U_{i+1}/U_i)$ implies that the digit-set for greedy representations is finite. For any $c_t \cdots c_0 \in \mathbb{N}^*$, we let $\operatorname{val}_U(c_t \cdots c_0)$ denote the integer $\sum_{i=0}^{t} c_i\, U_i$. A sequence U satisfying all the above conditions is said to define a *positional numeration system*. For the following, we refer to the terminology in [32] (addable systems are called regular in [41]).

▶ **Definition 2.1.** *A positional numeration system U with digit-set A is* addable *if the following* graph of addition*, denoted by $\mathcal{L}_+$, is regular:*

$$\left\{ \begin{pmatrix} u \\ v \\ w \end{pmatrix} \in (0^* \operatorname{rep}_U(\mathbb{N}))^3 \cap (A \times A \times A)^* \mid \operatorname{val}_U(u) + \operatorname{val}_U(v) = \operatorname{val}_U(w) \right\} \setminus \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} (A \times A \times A)^*.$$

Notice that words in $\operatorname{rep}_U(\mathbb{N})$ do not start with 0; however, when dealing with tuples of such words, shorter U-representations are padded with leading zeroes to get words of equal length (so they can be processed by an automaton over tuples of letters).

For general references about automatic words and abstract numeration systems, see [1] and [38] or [4, Chap. 3]. An *abstract numeration system* is a triple $S = (L, A, <)$ with L an infinite regular language over the totally ordered alphabet A (with $<$). Genealogically (i.e., radix or length-lexicographic) ordering L gives a one-to-one correspondence rep_S between $\mathbb{N}$ and L; the S-representation of n is the $(n+1)$st word of L, and the inverse map is denoted by val_S. A *deterministic finite automaton with output* (DFAO) $\mathcal{A}$ is a DFA (with state set Q) equipped with a mapping $\tau \colon Q \to A$ (with A an alphabet). The output $\mathcal{A}(w)$ of $\mathcal{A}$ on a word w is $\tau(q)$, where q is the state reached by reading w from the initial state.

▶ **Definition 2.2.** *An infinite word $\mathbf{x}$ is S-*automatic *if there exists a DFAO $\mathcal{A}$ such that $\mathbf{x}[n] = \mathcal{A}(\operatorname{rep}_S(n))$. In particular, for $k \geq 2$ an integer, if $\mathcal{A}$ is fed with the genealogically ordered language $L = \{\varepsilon\} \cup \{1, \ldots, k-1\}\{0, \ldots, k-1\}^*$, then $\mathbf{x}$ is said to be k-*automatic*.*

We introduce abstract numeration systems due to the following theorem:

▶ **Theorem 2.3** ([38])**.** *A word $\mathbf{x}$ is morphic if and only if it is S-automatic for some abstract numeration system S.*

Fix $s \in A^*$. For a word $\mathbf{x}$, define the subsequence $\mathbf{x} \circ s$ by $(\mathbf{x} \circ s)[n] := \mathbf{x}[\operatorname{val}_S(p_{s,n}\, s)]$, where $p_{s,n}$ is the nth word in the genealogically ordered language $Ls^{-1} = \{u \in A^* \mid us \in L\}$. The *$S$-kernel* of the word $\mathbf{x}$ is defined as the set of words $\{\mathbf{x} \circ s \mid s \in A^*\}$. The following theorem is critical to our arguments. Details are given in [4, Prop. 3.4.12–16].

▶ **Theorem 2.4** ([38])**.** *A word $\mathbf{x}$ is S-automatic if and only if its S-kernel is finite.*

▶ **Example 2.5.** Consider the Fibonacci numeration system based on the sequence $(F_n)_{n \geq 0}$ with $F_0 = 1$, $F_1 = 2$, and $F_{n+1} = F_n + F_{n-1}$ for $n \geq 1$. The first few terms of the associated subsequences $\mu_s : \mathbb{N} \to \mathbb{N}$, such that $(\mathbf{x} \circ s)[n] = \mathbf{x}[\mu_s(n)]$, are given in Table 1. One simply computes the numerical value of all the Fibonacci representations with the suffix s.

Table 1 The first few terms of some subsequences μ_s for the Fibonacci numeration system.

s	$(\mu_s(n))_{n\geq 0}$	s	$(\mu_s(n))_{n\geq 0}$
ε	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...	01	4, 6, 9, 12, 14, 17, 19, 22, 25, ...
0	2, 3, 5, 7, 8, 10, 11, 13, 15, ...	00	3, 5, 8, 11, 13, 16, 18, 21, 24, ...
1	1, 4, 6, 9, 12, 14, 17, 19, 22, 25, ...	10	2, 7, 10, 15, 20, 23, 28, 31, 36, 41, ...

Notice that some kernel elements $\mathbf{x} \circ s$ may be finite; more precisely, this occurs exactly when the language Ls^{-1} is finite. Our reasoning will not be affected by such particular cases, and we let the reader adapt it to such situations.

We now set our **general assumptions**. From now on we let U be a positional numeration system with digit-set A such that $\operatorname{rep}_U(\mathbb{N})$ is regular, and thus it is an abstract numeration system $(\operatorname{rep}_U(\mathbb{N}), A, <)$ for the natural ordering of the digits; this is because for all $m, n \in \mathbb{N}$ we have $m < n$ if and only if $\operatorname{rep}_U(m) <_{\text{gen}} \operatorname{rep}_U(n)$ for the genealogical order.

In Section 3.1, we further require U to be addable. All these assumptions are shared by many systems. For instance, classical integer base systems or the Fibonacci numeration system have all the assumed properties including being addable. For the latter system, the minimal automaton (reading most significant digits first) of $\mathcal{L}_+$ has 17 states (its transition table is given in [29]). The one reading least significant digits first has 22 states. The largest known family of positional systems with all these properties (addable with $\operatorname{rep}_U(\mathbb{N})$ being regular) is the one of those based on a linear recurrence sequence whose characteristic polynomial is the minimal polynomial of a Pisot number [5, 36].

2.2 Link with first-order logic

The result stated below is nothing else but the Büchi–Bruyère theorem [5, Thm. 16], originally given in the context of Pisot numeration systems. Its proof can be straightforwardly adapted to the case where U is addable, as the key ingredients are that $\operatorname{rep}_U(\mathbb{N})$ and the graph of addition $\mathcal{L}_+$ are regular languages. For a positional numeration system U, we define the map V_U by $V_U(n) = U_j$ whenever $n \in \mathbb{N}$ and $\operatorname{rep}_U(n) = c_t \cdots c_j 0^j$ and $c_j \neq 0$.

▶ **Theorem 2.6.** *Let U be an addable positional numeration system. A word $\mathbf{x}$ over B is U-automatic if and only if for each symbol $b \in B$, the set $\{n \geq 0 \mid \mathbf{x}[n] = b\}$ can be defined by a first-order formula $\varphi_b(n)$ of the structure $\langle \mathbb{N}, +, V_U \rangle$.*

As already mentioned in [19], the boundary sequence of a k-automatic word $\mathbf{x}$ may be defined by means of a first-order formula and therefore automaticity readily follows. This extends to addable systems: let $\mathbf{x} \in B^{\mathbb{N}}$ be U-automatic for an addable system U. The above theorem implies that, for all $b \in B$, we have a formula $\varphi_b(n)$ which is true if and only if $\mathbf{x}[n] = b$. We have $(u_1 \cdots u_\ell, v_1 \cdots v_\ell) \in \partial_{\mathbf{x},\ell}[m]$ if and only if

$$(\exists i) \bigwedge_{j=1}^{\ell} \varphi_{u_j}(i + j - 1) \wedge \bigwedge_{j=1}^{\ell} \varphi_{v_j}(i + m + j - 1).$$

For each subset R of $A^\ell \times A^\ell$ there is thus a formula $\psi_R(m)$ which is true if and only if $\partial_{\mathbf{x},\ell}[m] = R$. We may now apply Theorem 2.6 to conclude that $\partial_{\mathbf{x},\ell}$ is U-automatic. We remark that a similar proof of our Theorem 3.1 as sketched above is given in a forthcoming book of Shallit [41]. Also, this proof works for U not necessarily positional using [7, Lem. 37 and Thm. 55]. In particular, they show the following: Let $\mathbf{x}$ be an S-automatic sequence, where S is an addable abstract numeration system. Any first-order formula involving the predicates defined by the letters of $\mathbf{x}$ leads to an automaton accepting the S-representations of integers for which the formula holds.

3 On the boundary sequences of automatic words

In this section we provide the first of our main contributions, an alternative proof (not relying on Theorem 2.6) to the fact that a U-automatic word has a U-automatic boundary sequence whenever U is addable and satisfies the assumptions laid down in Section 2.1. We then show that this result does not necessarily hold for non-addable U.

3.1 Addable systems: automatic boundary sequences

For the sake of presentation, we only consider the case of the 1-boundary sequence. Our proof provides a precise description of a set containing the U-kernel of $\partial_{\mathbf{x}}$ in terms of three equivalence relations based on the kernel of $\mathbf{x}$, the graph of addition, and the numeration language; see (2). This set is finite, and so Theorem 2.3 gives the claim. In particular, one is the Myhill–Nerode congruence associated with the graph of addition since we have to consider the elements $\mathbf{x}[i]$ and $\mathbf{x}[i+m]$ for some $m > 0$. For $\ell > 1$, the only technical difference is that we have to consider longer factors $\mathbf{x}[i] \cdots \mathbf{x}[i+\ell-1]$ and $\mathbf{x}[i+m] \cdots \mathbf{x}[i+m+\ell-1]$.

▶ **Theorem 3.1.** *Let U be an addable numeration system with digit-set A and $\mathbf{x}$ be a U-automatic word. The boundary sequence $\partial_{\mathbf{x}}$ is U-automatic.*

Proof. Thanks to Theorem 2.4, the U-kernel of $\mathbf{x}$ is finite, say of cardinality m. Moreover, since $L = \mathrm{rep}_U(\mathbb{N})$ and $\mathcal{L}_+$ are regular, the following two sets of languages are finite by the Myhill–Nerode theorem [40, Sec. 3.9], say of cardinality k and ℓ, respectively:

$$\{Ls^{-1} \mid s \in A^*\} \quad \text{and} \quad \left\{\mathcal{L}_+ \begin{pmatrix} s \\ t \\ r \end{pmatrix}^{-1} \,\middle|\, \begin{pmatrix} s \\ t \\ r \end{pmatrix} \in (A \times A \times A)^*\right\}.$$

Let $\partial_{\mathbf{x}}$ be the boundary sequence of $\mathbf{x}$. An element of the U-kernel of $\partial_{\mathbf{x}}$ is given by $\partial_{\mathbf{x}} \circ s = \partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,0}s)]\ \partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,1}s)]\ \partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,2}s)] \cdots$ where $p_{s,n}$ is the nth word in the language Ls^{-1}, $n \geq 0$. Let us inspect the nth term of such an element of the kernel: it is precisely the set

$$\partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,n}s)] = \{(\mathbf{x}[i],\, \mathbf{x}[i + \mathrm{val}_U(p_{s,n}s)]) \mid i \geq 0\} \tag{1}$$

of pairs of letters. Let t, r be length-$|s|$ suffixes of words in L for which $\mathcal{L}_+(s,t,r)^{-1}$ is non-empty. There exist words w, x, y such that ws, xt, $yr \in 0^*L$ and $\mathrm{val}_U(ws) + \mathrm{val}_U(xt) = \mathrm{val}_U(yr)$. We let $\mathcal{P}(s)$ denote the set of such pairs $(t,r) \in (A \times A)^{|s|}$. Now partition (1) depending on the suffixes of length $|s|$ of $\mathrm{rep}_U(i)$ and $\mathrm{rep}_U(i + \mathrm{val}_U(p_{s,n}s))$: we may write

$$\partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,n}s)] = \bigcup_{(t,r)\in\mathcal{P}(s)} \left\{(\mathbf{x}[\mathrm{val}_U(xt)],\, \mathbf{x}[\mathrm{val}_U(yr)]) \,\middle|\, \begin{pmatrix} w \\ x \\ y \end{pmatrix} \in \mathcal{L}_+ \begin{pmatrix} s \\ t \\ r \end{pmatrix}^{-1} \wedge w \in 0^* p_{s,n}\right\}.$$

Roughly speaking, we look at all pairs of positions such that the first one is represented by a word ending with t, the second position is a shift of the first one by $\mathrm{val}_U(p_{s,n}s)$ and is represented by a word ending with r.

For convenience, we set $L(s,t,r,n) := \mathcal{L}_+ \begin{pmatrix} s \\ t \\ r \end{pmatrix}^{-1} \cap (0^*p_{s,n} \times A^* \times A^*)$ for all $n \geq 0$. Note that if $\mathcal{L}_+(s,t,r)^{-1} = \mathcal{L}_+(s',t',r')^{-1}$ and $Ls^{-1} = Ls'^{-1}$ then, for all n, $L(s,t,r,n) = L(s',t',r',n)$. Indeed, the second condition means that $p_{s,n} = p_{s',n}$ for all n.

Ordering $L(s,t,r,n)$. For each w, x of the same length, there is at most one y not starting with 0 such that (w,x,y) belongs to $\mathcal{L}_+(s,t,r)^{-1}$. Similarly if y does not start with 0, for each $w \in A^{|y|}$ (resp., $x \in A^{|y|}$) there is at most one x (resp., w) such that (w,x,y) belongs to $\mathcal{L}_+(s,t,r)^{-1}$.

Now let (w,x,y) and (w',x',y') in $L(s,t,r,n)$. We will always assume (this is not a restriction) that triplets do not start with $(0,0,0)$ – otherwise, different triplets may have the same numerical value. Note that $\mathrm{val}_U(x) < \mathrm{val}_U(x')$ if and only if $\mathrm{val}_U(y) < \mathrm{val}_U(y')$. Indeed, w, w' both belong to $0^*p_{s,n}$ thus $\mathrm{val}_U(ws) = \mathrm{val}_U(p_{s,n}s) = \mathrm{val}_U(w's)$. We have

$$\mathrm{val}_U(yr) - \mathrm{val}_U(xt) = \mathrm{val}_U(ws) = \mathrm{val}_U(w's) = \mathrm{val}_U(y'r) - \mathrm{val}_U(x't),$$

so, since U is positional, $\mathrm{val}_U(y0^{|r|}) - \mathrm{val}_U(y'0^{|r|}) = \mathrm{val}_U(x0^{|r|}) - \mathrm{val}_U(x'0^{|r|})$. Without loss of generality, we may assume that x and x' (resp., y and y') have the same length (indeed one can pad shorter representations with leading 0's). Then the above equation means that x is lexicographically less than x' if and only if the same holds for y and y'. We can thus order $L(s,t,r,n)$ by the numerical value of the second component of an element, and therefore the jth element of $L(s,t,r,n)$ is well-defined.

Defining two subsequences by the maps $\lambda_{s,t,r,n} : \mathbb{N} \to \mathbb{N}$ and $\mu_{s,t,r,n} : \mathbb{N} \to \mathbb{N}$. Let (w_j, x_j, y_j) be the jth element in $L(s,t,r,n)$ with $j \geq 0$. After removing the leading 0's, the word x_j belongs to $Lt^{-1} \cup \{\varepsilon\}$, which can also be ordered by genealogical order. We let $\lambda_{s,t,r,n}(j)$ denote the index (i.e., position counting from 0) of x_j within this language. Similarly, the word y_j belongs to $Lr^{-1} \cup \{\varepsilon\}$ and has an index $\mu_{s,t,r,n}(j)$ within this language.

Note that if $\mathcal{L}_+(s,t,r)^{-1} = \mathcal{L}_+(s',t',r')^{-1}$, $Ls^{-1} = Ls'^{-1}$, and $Lt^{-1} = Lt'^{-1}$ then, for all n, the maps $\lambda_{s,t,r,n}$ and $\lambda_{s',t',r',n}$ are the same. Indeed, the first two conditions imply that $L(s,t,r,n) = L(s',t',r',n)$. Similarly, if $\mathcal{L}_+(s,t,r)^{-1} = \mathcal{L}_+(s',t',r')^{-1}$, $Ls^{-1} = Ls'^{-1}$, and $Lr^{-1} = Lr'^{-1}$ then, for all n, the maps $\mu_{s,t,r,n}$ and $\mu_{s',t',r',n}$ are the same.

We now obtain

$$\begin{aligned}
\partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,n}s)] &= \bigcup_{(t,r)\in\mathcal{P}(s)} \left\{ (\mathbf{x}[\mathrm{val}_U(xt)],\, \mathbf{x}[\mathrm{val}_U(yr)]) \;\middle|\; \left(\begin{smallmatrix} w \\ x \\ y \end{smallmatrix}\right) \in \mathcal{L}_+ \left(\begin{smallmatrix} s \\ t \\ r \end{smallmatrix}\right)^{-1} \wedge w \in 0^* p_{s,n} \right\} \\
&= \bigcup_{(t,r)\in\mathcal{P}(s)} \left\{ (\mathbf{x}[\mathrm{val}_U(x_j t)],\, \mathbf{x}[\mathrm{val}_U(y_j r)]) \;\middle|\; \left(\begin{smallmatrix} w_j \\ x_j \\ y_j \end{smallmatrix}\right) \in L(s,t,r,n), j \geq 0 \right\} \\
&= \bigcup_{(t,r)\in\mathcal{P}(s)} \{ ((\mathbf{x}\circ t)[\lambda_{s,t,r,n}(j)], (\mathbf{x}\circ r)[\mu_{s,t,r,n}(j)]) \mid j \geq 0 \}\,.
\end{aligned}$$

Let us define an equivalence relation on triplets by $(s,t,r) \sim (s',t',r')$ if and only if all the following hold:

$$\begin{gathered}
\mathcal{L}_+ \left(\begin{smallmatrix} s \\ t \\ r \end{smallmatrix}\right)^{-1} = \mathcal{L}_+ \left(\begin{smallmatrix} s' \\ t' \\ r' \end{smallmatrix}\right)^{-1}, \quad Ls^{-1} = Ls'^{-1}, \quad Lt^{-1} = Lt'^{-1}, \quad Lr^{-1} = Lr'^{-1}, \\
\mathbf{x}\circ t = \mathbf{x}\circ t', \quad \text{and} \quad \mathbf{x}\circ r = \mathbf{x}\circ r'.
\end{gathered} \tag{2}$$

Since we have regular languages and the kernel of $\mathbf{x}$ is finite by assumption, this relation has a finite index (bounded by $\ell k^3 m^2$). Given s, the set $\{(s,t,r) \mid (t,r) \in \mathcal{P}(s)\}$ can be replaced by a set $\Lambda(s)$ of representatives of the equivalence classes for $\sim$. Since $\sim$ has a finite index, there are finitely many possible subsets of the form $\Lambda(s)$. So, we can write

$$\partial_{\mathbf{x}}[\mathrm{val}_U(p_{s,n}s)] = \bigcup_{(b,c,a)\in\Lambda(s)} \{ ((\mathbf{x}\circ c)[\lambda_{b,c,a,n}(j)], (\mathbf{x}\circ a)[\mu_{b,c,a,n}(j)]) \mid j \geq 0 \}\,.$$

Now if s and s' are such that $Ls^{-1} = Ls'^{-1}$ and $\Lambda(s) = \Lambda(s')$, then $\partial_{\mathbf{x}} \circ s = \partial_{\mathbf{x}} \circ s'$. This proves that the kernel of $\partial_{\mathbf{x}}$ is finite (of size bounded by $k \cdot 2^{\ell k^3 m^2}$). ◀

3.2 Non-addable systems: counterexamples

Our aim is to show that the boundary sequence of a U-automatic word is not always U-automatic. We give two such examples. The numeration system defined first is a variant of the base-2 system.

▶ **Example 3.2.** Take the numeration system $(U_n)_{n\geq 0}$ defined by $U_n = 2^{n+1} - 1$ for all $n \geq 0$. We have $0^* \operatorname{rep}_U(\mathbb{N}) = (0+1)^*(\varepsilon + 20^*)$. Consider the characteristic word $\mathbf{u}$ of U, i.e., $\mathbf{u}[n] = 1$ if and only if $n \in \{U_j \mid j \geq 0\}$. The boundary sequence $\partial_{\mathbf{u}}$ starts with

$$a\,b\,a\,b\,a\,b\,a\,b\,a\,a\,a\,b\,a\,b\,a\,b\,a\,a\,a\,a\,a\,a\,a\,b\,a\,a\,a\,b\,a\,b\,a\,b\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,b\,a\,a\,a\cdots$$

where $a := \{(0,0),(0,1),(1,0)\}$ and $b := \{0,1\} \times \{0,1\}$.

One can show that the language $\{\operatorname{rep}_U(n) \colon \partial_{\mathbf{u}}[n] = b\}$ is not regular, hence:

▶ **Proposition 3.3.** *Let $U = (2^{n+1} - 1)_{n\geq 0}$. The word $\mathbf{u}$ from Example 3.2 is U-automatic but its boundary sequence $\partial_{\mathbf{u}}$ is not U-automatic.*

As a consequence of the previous proposition and Theorem 3.1, U is non-addable.

▶ Remark 3.4. One may notice that both $\mathbf{u}$ and $\partial_{\mathbf{u}}$ are 2-automatic: this follows by Theorem 2.6 from the set $X := \{U_{m+r} - U_m \mid m \geq 0, r > 0\}$ (which equals $\{n \in \mathbb{N}\colon \partial_{\mathbf{u}}[n] = b\}$) being 2-definable by the formula $\varphi(n) := (\exists x)\,(\exists y)\,(x < y \wedge V_2(x) = x \wedge V_2(y) = y \wedge n = y - x)$, where $V_2(y)$ is the smallest power of 2 occurring with a non-zero coefficient in the binary expansion of y.

In view of the above remark, Example 3.2 could be considered as unsatisfactory. We now make use of a similar strategy but with a more complicated numeration system, for which we do not know any analogue of Remark 3.4. To this end, consider the non-addable numeration system from [17, Ex. 3] or [27, Ex. 2] defined by

$$V_0 = 1,\ V_1 = 4,\ V_2 = 15,\ V_3 = 54 \quad \text{and} \quad V_n = 3V_{n-1} + 2V_{n-2} + 3V_{n-4}, \quad \forall\, n \geq 4. \tag{3}$$

▶ **Example 3.5.** Consider the characteristic word $\mathbf{v}$ of V, i.e., $\mathbf{v}[n] = 1$ if and only if $n \in \{V_j \mid j \geq 0\}$. This word is trivially V-automatic. The boundary sequence $\partial_{\mathbf{v}}$ starts with

$$a\,a\,b\,a\,a\,a\,a\,a\,a\,a\,b\,a\,a\,b\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,b\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,b\cdots$$

where again $a := \{(0,0),(0,1),(1,0)\}$ and $b := \{0,1\} \times \{0,1\}$.

Similar to the above, $\{\mathrm{rep}_V(n)\colon \partial_{\mathbf{v}}[n] = b\}$ is not regular, whence

▶ **Proposition 3.6.** *Let V be the numeration system given by (3). The word $\mathbf{v}$ from Example 3.5 is V-automatic but its boundary sequence $\partial_{\mathbf{v}}$ is not V-automatic.*

▶ Remark 3.7. We do not know whether $\mathbf{v}$ and $\partial_{\mathbf{v}}$ are both V'-automatic for some numeration system V'.

4 The extended boundary sequences of Sturmian words

We give two descriptions of the ℓ-boundary sequences of Sturmian words (Theorem 4.1 and Proposition 4.10) and discuss some of their word combinatorial properties. We first recap minimal background on Sturmian words seen as codings of rotations. For a general reference, see [24, §2]. Let α, $\rho \in \mathbb{T} := [0,1)$ with α irrational. Define the *rotation* of the 1-dimensional torus $R_\alpha\colon \mathbb{T} \to \mathbb{T}$ by $R_\alpha(x) = \{x + \alpha\}$, where $\{\,\cdot\,\}$ denotes the fractional part. Let $I_0 = [0, 1-\alpha)$ (or $I_0 = (0, 1-\alpha]$) and $I_1 = \mathbb{T} \setminus I_0$. (The endpoints of I_0 will not matter in the forthcoming arguments.) Define the coding $\nu\colon \mathbb{T} \to \{0,1\}$ by $\nu(x) = 0$ if $x \in I_0$, otherwise $\nu(x) = 1$. We define the word $\mathbf{s}_{\alpha,\rho}$ by $\mathbf{s}_{\alpha,\rho}[n] = \nu(R_\alpha^n(\rho))$, for all $n \geq 0$. We call α the *slope* and ρ the *intercept* of $\mathbf{s}_{\alpha,\rho}$. The *characteristic Sturmian word of slope* α is $\mathbf{s}_{\alpha,\alpha}$.

4.1 A description of the extended boundary sequence

In the following, a *sliding block code of length* r is a mapping $\mathfrak{B}\colon A^{\mathbb{N}} \to B^{\mathbb{N}}$ defined by $\mathfrak{B}(\mathbf{x})[n] = \mathcal{B}(\mathbf{x}[n] \cdots \mathbf{x}[n + r - 1])$ for all $n \geq 0$ and some $\mathcal{B}\colon A^r \to B$. Let $T\colon A^{\mathbb{N}} \to A^{\mathbb{N}}$ denote the shift map $Tx_0x_1x_2\cdots = x_1x_2\cdots$.

▶ **Theorem 4.1.** *For a Sturmian word $\mathbf{s}$ of slope α (and intercept ρ) and $\ell \geq 1$, the (shifted) ℓ-boundary sequence $T\partial_{\mathbf{s},\ell}$ is obtained by a sliding block code of length 2ℓ applied to the characteristic Sturmian word of slope α.*

To prove the theorem we develop the required machinery. For a word $u = u_0 \cdots u_{\ell-1}$, we let $I_u = \bigcap_{i=0}^{\ell-1} R_\alpha^{-i}(I_{u_i})$. It is well-known that u occurs at position i in $\mathbf{s}_{\alpha,\rho}$ if and only if $R_\alpha^i(\rho) \in I_u$. These intervals of factors of length ℓ can also be described as follows: order the set $\big\{\{-j\alpha\}\big\}_{j=0}^{\ell}$ as $0 = i_0 < i_1 < i_2 < \cdots < i_\ell$. For convenience, we set $i_{\ell+1} = 1$. If the $\ell+1$ factors of length ℓ of the Sturmian word $\mathbf{s}_{\alpha,\rho}$ are lexicographically ordered as $w_0 < w_1 < \cdots < w_\ell$, then $I_{w_j} = [i_j, i_{j+1})$ for each $j \in \{0, \dots, \ell\}$. From the following claim it is evident that the intercept ρ plays no further role in our considerations. (This also follows from the fact that two Sturmian words have the same set of factors if and only if they have the same slope.)

▷ Claim 4.2. Let $n \geq \ell$ and u, v be length-ℓ factors of $\mathbf{s}_{\alpha,\rho}$. Then $(u, v) \in \partial_{\mathbf{x},\ell}[n]$ if and only if $R_\alpha^n(I_u) \cap I_v \neq \emptyset$.

The endpoints of I_u are of the form i_j and i_{j+1} for some $j \in \{0, \dots, \ell\}$. Hence, for $n \geq \ell$, the set of pairs belonging to $\partial_{\mathbf{x},\ell}[n]$ is determined by the positions of the rotated endpoints $R_\alpha^n(i_j)$ within the intervals I_{w_k}. Notice that each rotated endpoint $R_\alpha^n(i_j)$ always lies in the interior of some I_{w_k} whenever $n > \ell$. When $n = \ell$, we have $R_\alpha^n(\{-\ell\alpha\}) = 0$, which is an endpoint of one of the intervals I_{w_k}. For the time being we assume $n > \ell$, and return to the case $n = \ell$ in Proposition 4.8. Now, for example, if $R_\alpha^n(i_j) \in I_{w_k}$ then we have (w_j, w_k), $(w_{j-1}, w_k) \in \partial_{\mathbf{x},\ell}[n]$ (if $j = 0$, w_{j-1} is replaced with w_ℓ). Determining the boundary sets can be quite an intricate exercise; see Example 4.4.

An alternative to considering the positions of the points $R_\alpha^n(i_j)$ within the intervals I_{w_k} is to consider the positions of the points $R_\alpha^n(\{-j\alpha\})$ within the intervals I_{w_k} – the only difference is the order of enumeration. For each $n > \ell$, there is a map $\sigma = \sigma_n \in T_\ell$, where T_ℓ is the set of mappings from $\{0, \dots, \ell\}$ to itself, such that

$$R_\alpha^n(\{-j\alpha\}) \in I_{w_{\sigma(j)}} \quad \forall\, j \in \{0, \dots, \ell\}. \tag{4}$$

The realizable such configurations in (4) are called *constellations*. These points, when ordered according to the i_j's, determine the boundary set $\partial_{\mathbf{s},\ell}[n]$ as described above. See Example 4.4 for an illustration of the construction.

▶ **Definition 4.3.** *Let $\sigma \in T_\ell$ be such that* (4) *holds for some $n \in \mathbb{N}$. We define $\partial_\sigma \in 2^{A^\ell \times A^\ell}$ as the boundary set corresponding to any constellation inducing σ.*

It is now evident that if $\sigma_n = \sigma_m =: \sigma$, then $\partial_{\mathbf{s},\ell}[n] = \partial_\sigma = \partial_{\mathbf{s},\ell}[m]$.

▶ **Example 4.4.** The Fibonacci word $\mathbf{f}$ is $\mathbf{s}_{\alpha,\alpha}$ for $\alpha = (3 - \sqrt{5})/2 \simeq 0.382$. In Figure 2, the outer circle shows the partition with the interval $I_{w_0}, \dots, I_{w_\ell}$ and the inner circle shows the positions of the points $R_\alpha^n(\{-j\alpha\})$ for $\ell = 4$ and $n = 17$. The corresponding words $w_0, \dots, w_\ell$ are written next to their interval. Here σ_n is defined by $(0, 1, 2, 3, 4) \mapsto (2, 0, 3, 1, 4)$. For any constellation inducing σ_n, we see the pairs belonging to $\partial_{\sigma_n} = \partial_{\mathbf{f},4}[17]$ from Figure 2: the inner intervals (obtained from the outer intervals by applying R_α^{17}) give the prefix matching the suffix of the overlapping outer intervals, in clockwise order:

$$\binom{0010}{0101}, \binom{0010}{1001}, \binom{0100}{1001}, \binom{0100}{1010}, \binom{0101}{1010}, \binom{0101}{0010}, \binom{1001}{0010}, \binom{1001}{0100}, \binom{1010}{0100}, \binom{1010}{0101}.$$

Coming back to the introductory Example 1.2, the five sets $a_1, \dots, a_5$ correspond to the situations depicted from left to right in Figure 3. For instance, in the fourth picture, we understand why 10 is a prefix belonging to three pairs in a_4: the red inner interval intersects the three outer intervals of the partition. The situation is similar in the fifth picture where 01 is the prefix of three pairs in a_5. It is however not the case with the first three sets/pictures.

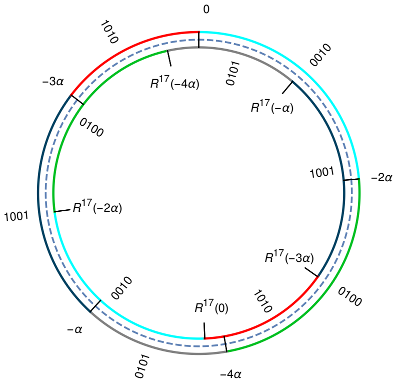


Figure 2 A constellation for $\alpha = (3 - \sqrt{5})/2$, $\ell = 4$ and $n = 17$.

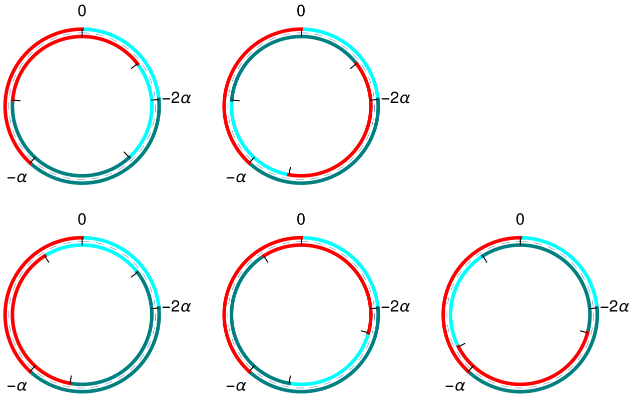


Figure 3 Some constellations for $\alpha = (3-\sqrt{5})/2$ and $\ell = 2$ inducing the five maps σ_n sending $(0, 1, 2)$, resp., to $(0, 2, 1)$, $(1, 0, 2)$, $(2, 1, 0)$, $(1, 2, 1)$, $(2, 1, 2)$.

▶ Remark 4.5. It is possible that $\partial_\sigma = \partial_{\sigma'}$ for distinct maps σ, $\sigma' \in T_\ell$. Indeed, for the Fibonacci word and $\ell = 1$, we have equality for the identity mapping id and $\sigma\colon (0, 1) \mapsto (1, 0)$; in this case $\partial_{\text{id}} = \partial_\sigma = \{0, 1\} \times \{0, 1\}$. So two constellations inducing different maps in T_ℓ lead to the same set of boundary pairs. (See however Lemma 4.15.)

▶ **Definition 4.6.** *Let* $\mathbf{r}$ *be the rotation word defined by* $\mathbf{r}[n] = \eta(R^n_\alpha(\alpha))$ *for all* $n \geq 0$*, where* $\eta\colon \mathbb{T} \to \{0, \ldots, \ell\}$ *is defined by* $\eta(x) = j$ *when* $x \in I_{w_j}$ *(recall* I_{w_i} *corresponds to the ith factor of length* ℓ*).*

We have that $\mathbf{r}[n] = j$ if and only if the characteristic Sturmian word $\mathbf{s}_{\alpha,\alpha}$ has the length-ℓ factor w_j occurring at position n.

Proof sketch of Theorem 4.1. It can be shown that the boundary sequence of $\mathbf{s}$ can be obtained from $\mathbf{r}$ constructed above using a sliding block code of length $\ell + 1$. Notice that by definition $\mathbf{r}$ is obtained by a sliding block code of length ℓ of the characteristic Sturmian word $\mathbf{s}_{\alpha,\alpha}$. The claim follows as composing sliding block codes of length r and r', respectively, yields a sliding block code of length $r + r' - 1$. ◀

▶ **Example 4.7.** We apply Theorem 4.1 to the Fibonacci word $\mathbf{f}$. Take $\alpha = (3-\sqrt{5})/2$, $\ell = 1$, $I_0 = [0, 1 - \alpha)$ and $I_1 = [1 - \alpha, 1)$. Then the rotation word $\mathbf{r}$ associated with the partition $\{I_0, I_1\}$, slope α, and intercept α is $\mathbf{s}_{\alpha,\alpha}$ by definition, which happens to be the Fibonacci word $\mathbf{f}$. We have $\mathbf{f} = 0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,0\,0\,1 \cdots$. Recall from the construction that the length-2 factors of the rotation word determine the boundary sets. The three length-2 factors of $\mathbf{f}$ are 01, 10, and 00 occurring at positions $m = 0$, 1, and 2, respectively. We get the three maps $\sigma_{m+2} \in T_1$ defined by $(0, 1) \mapsto (1, 0)$, $(0, 1) \mapsto (0, 1)$, and $(0, 1) \mapsto (0, 0)$, respectively. We deduce that an occurrence of 01 or 10 corresponds to the boundary set $b := \{0, 1\} \times \{0, 1\}$, and 00 to $a := \{(0, 0), (0, 1), (1, 0)\}$. We may therefore define $\mathcal{B}\colon 01, 10 \mapsto b$, $00 \mapsto a$ and the associated sliding block code $\mathfrak{B}$ of length 2; applying $\mathfrak{B}$ to $\mathbf{f}$, we get

$$\begin{aligned}&\mathfrak{B}((01)(10)(00)(01)(10)(01)(10)(00)(01)(10)(00)(01)(10)(01)(10)(00)(01)(10)(01)\cdots)\\ &= \;b \quad b \quad a \quad b \quad b \quad b \quad b \quad a \quad b \quad b \quad a \quad b \quad b \quad b \quad b \quad a \quad b \quad b \quad b\cdots,\end{aligned}$$

which indeed gives back Example 1.2 after prepending the letter a.

We next discuss the first element $\partial_{\mathbf{s},\ell}[\ell]$ of the (extended) boundary sequence. Notice that the set is in one-to-one correspondence with the factors of length 2ℓ, and thus has cardinality $2\ell+1$. The points $\{-j\alpha\}$ and $R_\alpha^\ell(\{-j\alpha\})$, $j \in \{0,\ldots,\ell\}$, on the torus still determine the boundary set, but notice that there are only $2\ell+1$ distinct pairs. The following proposition describes rather precisely how the first element appears in the boundary sequence.

▶ **Proposition 4.8.** *For a Sturmian word* $\mathbf{s}$*, the boundary set* $\partial_{\mathbf{s},\ell}[\ell]$ *appears infinitely often in* $\partial_{\mathbf{s},\ell}$ *if and only if* $0^{2\ell}$ *or* $1^{2\ell}$ *appears in* $\mathbf{s}$*. Otherwise it appears exactly once.*

Notice that either 00 or 11 appears in a Sturmian word $\mathbf{s}$, so the above implies that the first letter of the (1-)boundary sequence $\partial_{\mathbf{s}}$ always appears infinitely often in the sequence. Returning to Example 1.2, since 0^4 does not appear in the Fibonacci word, the letter a_0 appears only once in $\partial_{\mathbf{f},2}$.

We conclude with the immediate corollary of Theorem 4.1 and Proposition 4.8; here we say that a word $\mathbf{w}$ is *uniformly recurrent* if each of its factors occurs infinitely often within bounded gaps (the distance between two consecutive occurrences depends on the factor). It is known that, e.g., Sturmian words are uniformly recurrent.

▶ **Corollary 4.9.** *For any Sturmian word* $\mathbf{s}$*, the shifted sequence* $T\partial_{\mathbf{s},\ell}[n]$ *is uniformly recurrent. The sequence* $\partial_{\mathbf{s},\ell}$ *is uniformly recurrent if and only if* $0^{2\ell}$ *or* $1^{2\ell}$ *appears in* $\mathbf{s}$*.*

4.2 Another description of the extended boundary sequence

We give another description of the ℓ-boundary sequences of Sturmian words when $\ell \geq 2$. For any irrational number $\alpha \in (0,1)$ there is a unique infinite continued fraction expansion (CFE)

$$\alpha = [0; a_1, a_2, a_3, \ldots] := \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}},$$

where $a_n \geq 1$ are integers for all $n \geq 1$. Then the characteristic Sturmian word $\mathbf{s}_{\alpha,\alpha}$ of slope α equals $\lim_{k\to\infty} S_k$, where $S_{-1} = 1$, $S_0 = 0$, $S_1 = S_0^{a_1-1}S_{-1}$, and $S_{k+1} = S_k^{a_{k+1}}S_{k-1}$ for all $k \geq 1$ [1, Chap. 9]. The main result of this part is the following.

▶ **Proposition 4.10.** *Let* $\mathbf{s}$ *be a Sturmian word of slope* $\alpha = [0; a_1+1, a_2, \ldots]$*. For each* $\ell \geq 2$*, there exists* $k_\ell \in \mathbb{N}$ *such that for any* $k \geq k_\ell$ *there is a morphism* $h_{k,\ell}$ *such that* $T\partial_{\mathbf{s},\ell} = h_{k,\ell}(\mathbf{s}_{\beta_k,\beta_k})$*, where* $\beta_k = [0; a_{k+1}+1, a_{k+2}, \ldots]$*.*

▶ **Example 4.11.** Take the slope $\alpha = (3-\sqrt{5})/2$; its CFE is $[0; 2, 1, 1, 1, \ldots]$. Using the previous notation, $S_{-1} = 1$, $S_0 = 0$, and $S_{k+1} = S_kS_{k-1}$ for all $k \geq 0$. Then the sequence $(S_k)_{k\geq 0}$ converges to the Fibonacci word; the first few words in the sequence $(S_k)_{k\geq 0}$ are $0, 01, 010, 01001, 01001010$.

Now for any $\ell \geq 2$, the above proposition thus gives that $\partial_{\mathbf{f},\ell}$ is the morphic image of the characteristic Sturmian word of slope $\beta = \alpha$. In other words, the ℓ-boundary sequence is always a morphic image of $\mathbf{f}$.

We generalize the last observation made in the above example.

▶ **Corollary 4.12.** *Let* $\mathbf{s}$ *be a Sturmian word with quadratic slope. Then* $\partial_{\mathbf{s},\ell}$ *is morphic. In particular, the* ℓ*-boundary sequence of a Sturmian word fixed by a non-trivial morphism is morphic.*

Proof. A remarkable result of Yasutomi [43] (see also [3]), characterizing those Sturmian words that are fixed by some non-trivial morphism, implies that if a Sturmian word is fixed by a non-trivial morphism, then so is the characteristic Sturmian word of the same slope. Furthermore, the slope is characterized by the property that its CFE is of the form $[0;1,a_2,\overline{a_3,\ldots,a_r}]$ with $a_r \geq a_2$ or $[0;1+a_1,\overline{a_2,\ldots,a_r}]$ with $a_r \geq a_1 \geq 1$ [11, 30] (see also [24, Thm. 2.3.25]). Here $\overline{x_1,\ldots,x_t}$ indicates the periodic tail of the infinite CFE. Let $\mathbf{s}$ have slope α; since α is quadratic, it has an eventually periodic CFE. There thus exist arbitrarily large k for which the characteristic Sturmian word of slope $\beta_k := [0;a_k+1,a_{k+1},\ldots]$ is a fixed point of a non-trivial morphism (it is of the latter form). Proposition 4.10 then posits that $T\partial_{\mathbf{s},\ell}$ is the morphic image of such a word, and the claim follows (because prepending the letter $\partial_{\mathbf{s},\ell}[\ell]$ preserves morphicity [1, Thm. 7.6.3]). ◀

Notice that given the morphism fixing a Sturmian word $\mathbf{s}$, one can compute (the CFE of) its quadratic slope (and intercept) [42, 34, 23].

The above corollary has an alternative proof via the logical approach as well. For the definitions of notions that follow, we refer to the cited papers. From the work of Hieronymi and Terry [20], it is known that addition in the *Ostrowski-numeration system* based on an irrational quadratic number α is recognizable by a finite automaton. This motivated Baranwal, Schaeffer, and Shallit to introduce *Ostrowski-automatic sequences* in [2]. For example, they showed that the characteristic Sturmian word of slope α is Ostrowski α-automatic. Since the numeration system is addable, the above corollary follows by the same arguments as in Section 2.2.

4.3 Factor complexities of the extended boundary sequences

▶ **Definition 4.13.** *An infinite word over an alphabet A is of* minimal complexity *if its factor complexity is $n+|A|-1$ for all $n \geq 1$.*

Minimal complexity words can be seen as a generalization of Sturmian words to larger alphabets: if a word (containing all letters of A) has less than $n+|A|-1$ factors of length n for some n, then it is ultimately periodic. Otherwise it is aperiodic (a consequence of the Morse–Hedlund theorem). See [10, 31, 15, 6, 13] for characterizations and generalizations.

The following proposition is almost immediate after the key Lemma 4.15.

▶ **Proposition 4.14.** *Let $\ell \geq 2$. The ℓ-boundary sequence of a Sturmian word is a minimal complexity word (of complexity $n \mapsto n+2\ell$, $n \geq 1$).*

▶ **Lemma 4.15.** *Let σ and $\sigma' \in T_\ell$, $\ell \geq 2$, be distinct mappings both satisfying* (4) *(for different n). Then $\partial_\sigma \neq \partial_{\sigma'}$.*

We conclude with a formula for the factor complexity of the 1-boundary sequence of Sturmian words.

▶ **Proposition 4.16.** *Let r be the maximal integer such that $(01)^r$ appears in the Sturmian word $\mathbf{s}$. The boundary sequence $\partial_{\mathbf{s}}$ has factor complexity* $n \mapsto \begin{cases} n+1, & \text{if } n < 2r; \\ n+2, & \text{otherwise.} \end{cases}$

As an immediate corollary, we see that the ℓ-boundary sequence is aperiodic for all $\ell \geq 1$.

References

1 Jean-Paul Allouche and Jeffrey Shallit. *Automatic sequences: Theory, applications, generalizations*. Cambridge University Press, Cambridge, 2003.

2 Aseem Baranwal, Luke Schaeffer, and Jeffrey Shallit. Ostrowski-automatic sequences: Theory and applications. *Theoretical Computer Science*, 858:122–142, 2021. doi:10.1016/j.tcs.2021.01.018.

3 Valérie Berthé, Hiromi Ei, Shunji Ito, and Hui Rao. On substitution invariant Sturmian words: an application of Rauzy fractals. *RAIRO Theoretical Informatics and Applications*, 41(3):329–349, 2007. doi:10.1051/ita:2007026.

4 Valérie Berthé and Michel Rigo, editors. *Combinatorics, Automata, and Number Theory*, volume 135 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2010. doi:10.1017/CBO9780511777653.

5 Véronique Bruyère and Georges Hansel. Bertrand numeration systems and recognizability. *Theoretical Computer Science*, 181(1):17–43, 1997. doi:10.1016/S0304-3975(96)00260-5.

6 Julien Cassaigne. Sequences with grouped factors. In Symeon Bozapalidis, editor, *Proceedings of the 3rd International Conference Developments in Language Theory*, pages 211–222. Aristotle University of Thessaloniki, 1997.

7 Émilie Charlier, Célia Cisternino, and Manon Stipulanti. Regular sequences and synchronized sequences in abstract numeration systems. *European Journal of Combinatorics*, 101:103475, 2022. doi:10.1016/j.ejc.2021.103475.

8 Jin Chen and Zhi-Xiong Wen. On the abelian complexity of generalized Thue–Morse sequences. *Theoretical Computer Science*, 780:66–73, 2019. doi:10.1016/j.tcs.2019.02.014.

9 Alan Cobham. Uniform tag seqences. *Mathematical Systems Theory*, 6(3):164–192, 1972. doi:10.1007/BF01706087.

10 Ethan M. Coven. Sequences with minimal block growth ii. *Mathematical systems theory*, 8:376–382, 1974. doi:10.1007/BF01780584.

11 David Crisp, William Moran, Andrew Pollington, and Peter Shiue. Substitution invariant cutting sequences. *Journal de Théorie des Nombres de Bordeaux*, 5(1):123–137, 1993. doi:10.2307/26273915.

12 James Currie, Tero Harju, Pascal Ochem, and Narad Rampersad. Some further results on squarefree arithmetic progressions in infinite words. *Theoretical Computer Science*, 799:140–148, 2019. doi:10.1016/j.tcs.2019.10.006.

13 Gilles Didier. Caractérisation des N-écritures et application à l'étude des suites de complexité ultimement $n+c^{ste}$. *Theoretical Computer Science*, 215(1–2):31–49, 1999. doi:10.1016/S0304-3975(97)00122-9.

14 Jean-Pierre Duval. Relationship between the period of a finite word and the length of its unbordered segments. *Discrete Mathematics*, 40:31–44, 1982. doi:10.1016/0012-365X(82)90186-8.

15 Sébastien Ferenczi and Christian Mauduit. Transcendence of numbers with a low complexity expansion. *Journal of Number Theory*, 67(2):146–161, 1997. doi:10.1006/jnth.1997.2175.

16 Aviezri S. Fraenkel. Systems of numeration. *The American Mathematical Monthly*, 92:105–114, 1985. doi:10.2307/2322638.

17 Christiane Frougny. On the sequentiality of the successor function. *Information and Computation*, 139(1):17–38, 1997. doi:10.1006/inco.1997.2650.

18 Melissa J. Fullwood, Chia-Lin Wei, Edison T. Liu, and Yijun Ruan. Next-generation DNA sequencing of paired-end tags (PET) for transcriptome and genome analyses. *Genome research*, 19(4):521–532, 2009. doi:10.1101/gr.074906.107.

19 Ying-Jun Guo, Xiao-Tao Lü, and Zhi-Xiong Wen. On the boundary sequence of an automatic sequence. *Discrete Mathematics*, 345(1):9, 2022. Id/No 112632. doi:10.1016/j.disc.2021.112632.

20 Philipp Hieronymi and Alonza Terry Jr. Ostrowski Numeration Systems, Addition, and Finite Automata. *Notre Dame Journal of Formal Logic*, 59(2):215–232, 2018. doi:10.1215/00294527-2017-0027.

21 Philipp Hieronymi, Dun Ma, Reed Oei, Luke Schaeffer, Christian Schulz, and Jeffrey Shallit. Decidability for Sturmian Words. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2022.24.

22 Juhani Karhumäki, Aleksi Saarela, and Luca Q. Zamboni. On a generalization of abelian equivalence and complexity of infinite words. *Journal of Combinatorial Theory, Series A*, 120(8):2189–2206, 2013. doi:10.1016/j.jcta.2013.08.008.

23 Jana Lepšová, Edita Pelantová, and Štěpán Starosta. On a faithful representation of Sturmian morphisms, 2022. Preprint. doi:10.48550/ARXIV.2203.00373.

24 M. Lothaire. *Algebraic combinatorics on words*, volume 90 of *Encyclopedia of Mathematics and its Applications*. Cambridge: Cambridge University Press, 2002.

25 Xiao-Tao Lü, Jin Chen, Zhi-Xiong Wen, and Wen Wu. On the 2-binomial complexity of the generalized Thue-Morse words, 2021. Preprint. doi:10.48550/ARXIV.2112.05347.

26 Dimitris Margaritis and Steven S. Skiena. Reconstructing strings from substrings in rounds. In *36th Annual symposium on Foundations of computer science. Held in Milwaukee, WI, USA, October 23–25, 1995*, pages 613–620. Los Alamitos, CA: IEEE Computer Society Press, 1995.

27 Adeline Massuir, Jarkko Peltomäki, and Michel Rigo. Automatic sequences based on Parry or Bertrand numeration systems. *Advances in Applied Mathematics*, 108:11–30, 2019. doi:10.1016/j.aam.2019.03.003.

28 Hamoon Mousavi. Walnut prover, 2016. , https://cs.uwaterloo.ca/~shallit/walnut.html. URL: https://github.com/hamousavi/Walnut.

29 Hamoon Mousavi, Luke Schaeffer, and Jeffrey Shallit. Decision algorithms for Fibonacci-automatic words. I: Basic results. *RAIRO Theoretical Informatics and Applications*, 50(1):39–66, 2016. doi:10.1051/ita/2016010.

30 Bruno Parvaix. Propriétés d'invariance des mots sturmiens. *Journal de Théorie des Nombres de Bordeaux*, 9(2):351–369, 1997. doi:10.5802/jtnb.207.

31 Michael E. Paul. Minimal symbolic flows having minimal block growth. *Mathematical systems theory*, 8:309–315, 1974. doi:10.1007/BF01780578.

32 Jarkko Peltomäki and Ville Salo. Automatic winning shifts. *Information and Computation*, 285:104883, 2022. doi:10.1016/j.ic.2022.104883.

33 Jarkko Peltomäki and Markus A. Whiteland. On k-abelian equivalence and generalized Lagrange spectra. *Acta Arithmetica*, 194(2):135–154, 2020. doi:10.4064/aa180927-10-9.

34 Li Peng and Bo Tan. Sturmian Sequences and Invertible Substitutions. *Discrete Mathematics & Theoretical Computer Science*, 13(2), 2011. doi:10.46298/dmtcs.554.

35 Thomas Place, Lorijn Van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *33nd international conference on foundations of software technology and theoretical computer science, FSTTCS 2013, Guwahati, India, December 12–14, 2013. Proceedings*, pages 363–375. Wadern: Schloss Dagstuhl – Leibniz Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.363.

36 Michel Rigo. *Formal languages, automata and numeration systems. 2*. Networks and Telecommunications Series. ISTE, London; John Wiley & Sons, Inc., Hoboken, NJ, 2014. Applications to recognizability and decidability, With a foreword by Valérie Berthé.

37 Michel Rigo. Relations on words. *Indagationes Mathematicae*, 28(1):183–204, 2017. doi:10.1016/j.indag.2016.11.018.

38 Michel Rigo and Arnaud Maes. More on generalized automatic sequences. *Journal of Automata, Languages, and Combinatorics*, 7(3):351–376, 2002. doi:10.25596/jalc-2002-351.

39 Michel Rigo, Manon Stipulanti, and Markus A. Whiteland. Binomial complexities and Parikh-collinear morphisms. In Volker Diekert and Mikhail Volkov, editors, *Developments in Language Theory*, pages 251–262, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-05578-2_20.

40 Jeffrey Shallit. *A second course in formal languages and automata theory.* Cambridge: Cambridge University Press, 2009. doi:10.1017/CBO9780511808876.

41 Jeffrey Shallit. *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with Walnut.* London Mathematical Society Lecture Note Series. Cambridge University Press, 2022. To appear.

42 Bo Tan and Zhi-Ying Wen. Invertible substitutions and Sturmian sequences. *European Journal of Combinatorics*, 24(8):983–1002, 2003. doi:10.1016/S0195-6698(03)00105-7.

43 Shin-Ichi Yasutomi. On Sturmian sequences which are invariant under some substitution. In *Number Theory and Its Applications (Kyoto, 1997)*, volume 2 of *Dev. Math.*, pages 347–373. Kluwer Academic Publishers, Dordrecht, 1999.

Higher-Order Causal Theories Are Models of BV-Logic

Will Simmons
Department of Computer Science, University of Oxford, Oxford, UK
Cambridge Quantum, Terrington House, 13–15 Hills Road, Cambridge, UK

Aleks Kissinger
Department of Computer Science, University of Oxford, Oxford, UK

Abstract

The Caus[−] construction takes a compact closed category of basic processes and yields a *-autonomous category of higher-order processes obeying certain signalling/causality constraints, as dictated by the type system in the resulting category. This paper looks at instances where the base category C satisfies additional properties yielding an affine-linear structure on Caus[$\mathcal{C}$] and a substantially richer internal logic. While the original construction only gave multiplicative linear logic, here we additionally obtain additives and a non-commutative, self-dual sequential product yielding a model of Guglielmi's BV logic. Furthermore, we obtain a natural interpretation for the sequential product as "A can signal to B, but not vice-versa", which sits as expected between the non-signalling tensor and the fully-signalling (i.e. unconstrained) par. Fixing matrices of positive numbers for $\mathcal{C}$ recovers the BV category structure of probabilistic coherence spaces identified by Blute, Panangaden, and Slavnov, restricted to normalised maps. On the other hand, fixing the category of completely positive maps gives an entirely new model of BV consisting of higher order quantum channels, encompassing recent work in the study of quantum and indefinite causal structures.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Linear logic; Theory of computation → Categorical semantics

Keywords and phrases Causality, linear logic, categorical logic, probabilistic coherence spaces, quantum channels

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.80

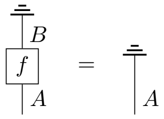
Related Version *Full Version*: `https://arxiv.org/abs/2205.11219` [26]

Funding *Will Simmons*: was supported by Cambridge Quantum (Quantinuum Ltd).
Aleks Kissinger: acknowledges support from Grant No. 61466 from the John Templeton Foundation as part of the QISS project. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the John Templeton Foundation.

Acknowledgements The authors would like to thank Alessio Guglielmi for posing the question of modelling BV-logic within Caus[$\mathcal{C}$], as well as Chris Barrett, Lutz Straßburger, and members of the quantum group at University of Oxford for useful discussions.

1 Introduction

The causality condition [8] is a simple equation one can impose on a family of processes that states essentially that "discarding the output of f is the same as discarding its input." Pictorially:[1]



(1)

[1] We will use *string diagram* notation for monoidal categories throughout, depicting morphisms as boxes, composition as "plugging" boxes from bottom-to-top and ⊗ as placing boxes side-by-side. See e.g. [25].

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 80; pp. 80:1–80:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for some fixed family of “discarding” processes $\bar{\bar{\top}}_A$. While this seems to be a simple equation, and reasonable to impose on most sane collections of physical processes, it deserves the somewhat lofty name of *causality* because it really seems to capture the essence of influences moving in a single direction: forward in time. In particular, it guarantees that distant agents cannot send messages to one another without their existing a forward-directed path in the diagram of their processes. Without referring explicitly to spacetime, this serves as an abstract stand-in for what a relativity theorist would also call causality, namely the limitation that influences cannot propagate faster than the speed of light [17].

In concrete categories of interest, namely those of probabilistic and quantum maps, equation (1) simply imposes that processes should preserve normalised states. Iterating this, we might wish to consider second-order processes that preserve processes that preserve normalised states, and so on. This turns out to provide a rich landscape for modelling causal relationships between events, enabling for example chains [6, 13] (or directed acyclic graphs [19]) of events in a definite causal order, probabilistic mixtures of causal orderings, or even exotic indefinite causal structures [1, 9, 22].

There are several routes to constructing a framework for characterising and composing such higher-order processes, including describing acyclic graphs of interactions between discrete first-order channels [4, 7] or building an infinite hierarchy of types recursively out of constructors describing a kind of causal relationship [2].

The Caus[−] construction of Kissinger and Uijlen [19] fits the latter style through a category where morphisms map normalised states of the source type to normalised states of the target type. For example, Caus[CP*] describes higher-order generalisations of CPTP-maps, containing quantum combs [7], process matrices [22], and bipartite second-order maps [18] which include the quantum switch [9], which have collectively been critical in the formulation and study of indefinite causal structures. The typing of a morphism is a judgement about externally-observable properties of normalisation and information flow as opposed to any assumed internal structure or spacetime geometry. The type constructors model the operators of Multiplicative Linear Logic (MLL), i.e. it forms a $*$-autonomous category. MLL features two logical connectives, tensor and par, which serve as two extremes in the Caus[−] construction. Tensor yields a *non-signalling* composition of processes, where causal influences are not allowed to pass from one side to the other, whereas par gives a *fully-signalling* composition, i.e. one that imposes no signalling constraints.

In between this lies the *one-way signalling* processes, where causal influences can flow from one agent (say, Alice in the past) to another (say, Bob in the future). While special cases of such processes were treated in an *ad hoc* way in [19], here we will show that one-way signalling can in fact be treated as a fully-fledged connective in its own right, yielding a substantially richer logical structure.

BV-logic [11] adds a third logical connective that is non-commutative to capture sequentiality in a similar way to Retoré’s pomset logic [24]. It admits a categorical characterisation via BV-categories, and has previously been applied to the study of (probabilistic) coherence spaces [3] and a certain graph-based model of quantum causal structures called discrete quantum causal dynamics [4].

In this paper, we will adapt the Caus[−] construction by modifying some of the assumptions on the base category, requiring it to be *additive precausal.* The extra structure allows us to consider new ways of combining processes and types to describe operational constructions such as binary tests, probability distributions, and one-way signalling processes. These correspond to extending the logical structure of Caus[$\mathcal{C}$] with the additive connectives (sometimes called “with” and “plus”) of linear logic, as well as the sequential connective of BV. We show that the main classical and quantum examples of precausal categories are furthermore additive precausal, enabling Caus[$\mathcal{C}$] to model higher-order theories and classical and quantum causal structures, respectively. We also note the existence of non-standard models, such as higher-order affine (a.k.a. quasi-probabilistic) processes.

During the development of this paper, independent work by Hoffreumon and Oreshkov [14] investigated the same spaces of higher-order one-way signalling processes specifically in quantum theory and identified the same self-duality along with some additional results on decompositions into intersections and unions of spaces. Additionally, Cavalcanti et al. [5] obtained the same characterisation of non-signalling spaces as the affine closure of separable processes for multi-partite first-order channels in any generalised probabilistic theory with local tomography.

We refer the reader to the arXiv version of this paper [26] for full proofs of all the results presented here.

2 Higher-Order Causal Structure

We use "process theory" style terminology (see e.g. [10]) for monoidal categories throughout. Namely, we use the terms *type* or *system* interchangeably to refer to objects of a symmetric monoidal category, *process* to refer to a generic morphism $f : A \to B$, *state* to refer to a morphism $\rho : I \to A$ from the tensor unit, *effect* to refer to a morphism $\pi : A \to I$ to the tensor unit, and *scalar* to refer to a morphism $\lambda : I \to I$. In categories admitting higher-order structure, we think of first-order types as state spaces and first-order processes as maps that transform first-order states to first-order states. Higher-order processes are transformations where the input or output system is itself a map. The main example used in causality literature is a quantum 2-comb which is a process taking a channel over first-order types as an input and transforms it into a new channel, typically by composing with some pre- and post-processing which may share some memory channel. One can extend this to an infinite hierarchy of higher-order processes where an $(n+1)$-comb transforms an n-comb into a 1-comb (a channel) [7]. A higher-order process theory is one which describes processes in such an infinite hierarchy uniformly.

Higher-order theories are commonly achieved by providing a mechanism to encode transformations into states of some function type. In category theory, this corresponds to an internal hom in monoidal closed categories, i.e. a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ such that $\mathcal{C}(C \otimes A, B) \simeq \mathcal{C}(C, A \multimap B)$ and this is natural in all arguments. Wilson and Chiribella [27] demonstrate that adding basic manipulations (morphisms that capture sequential and parallel composition of such encoded functions) is sufficient to permit the inductive generation of comb types. However, with higher-order theories, there are more ways of composing processes that need not be obtainable in this way.

To solve this, we may further ask that $\mathcal{C}$ be a *compact closed category*, i.e. we require every object A has a "cup" state $\eta_A : I \to A^* \otimes A$ and a "cap" effect satisfying the so-called "yanking equations": $(\mathrm{id}_A \otimes \eta_A) \fatsemi (\epsilon_A \otimes \mathrm{id}_A) = \mathrm{id}_A$ and $(\eta_A \otimes \mathrm{id}_{A^*}) \fatsemi (\mathrm{id}_{A^*} \otimes \epsilon_A) = \mathrm{id}_{A^*}$. These enable us to convert inputs to outputs at will, which in turn lets us "wire up" processes in arbitrary ways to each other. Namely, we can treat all higher order processes as states, and use "caps" to connect them to each other in arbitrary ways. See Figure 1 for an example and [19, Section 2.1] for more details.

The causality condition in equation (1) specialises to states as the requirement that $\rho \fatsemi \bar{\bar{\top}}_A = \mathrm{id}_I =: 1$ for any state ρ, i.e. it imposes that states be normalised with respect to the discarding effect. Discarding is unique in the sense that it is the only effect normalised for all states. However, for the higher-order analogue of state spaces, there might be more such effects, motivating the following definition.

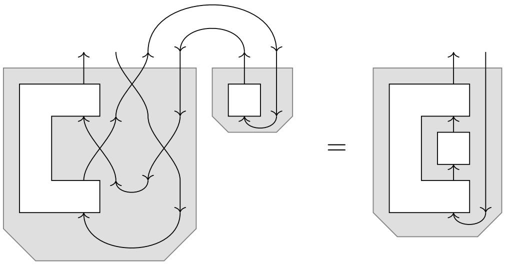


Figure 1 Example of embedding higher-order processes and composition via compact closure.

▶ **Definition 1.** *For $c \subseteq \mathcal{C}(I, A)$, the dual set is $c^* := \{\pi \in \mathcal{C}(A, I) \,|\, \forall \rho \in c.\rho \,⨾\, \pi = \mathrm{id}_I\}$.*

▶ Remark 2. In this paper, we will freely mix between treating c/c^* as a set of states in $\mathcal{C}(I, A)/\mathcal{C}(I, A^*)$ and a set of effects in $\mathcal{C}(A^*, I)/\mathcal{C}(A, I)$, depending on the context. They are equivalent through the transpose isomorphism induced by compact closure.

From this we can recover the space of first-order/*causal* states for a system A as $\left\{\overset{=}{\top}_A\right\}^*$. For example, in quantum theory we take $\overset{=}{\top}_A \in \mathcal{C}(A, I)$ to represent the partial trace over the system A, so this set describes the space of trace-1 density matrices. First-order/causal processes from A to B are those that map every state in $\left\{\overset{=}{\top}_A\right\}^*$ to a state in $\left\{\overset{=}{\top}_B\right\}^*$ (equivalent to mapping the effect $\overset{=}{\top}_B$ to $\overset{=}{\top}_A$ when our category has enough states/well-pointedness), which we can encode as $\left\{\left\{\overset{=}{\top}_A\right\}^* \otimes \overset{=}{\top}_B\right\}^* \subseteq \mathcal{C}(I, A^* \otimes B)$. These concepts are used to define a precausal category, which specifies one possible set of conditions to enable discussions of higher-order causal structures.

▶ **Definition 3.** *A compact closed category $\mathcal{C}$ is a precausal category if:*

PC1. *$\mathcal{C}$ has a discarding process $\overset{=}{\top}_A \in \mathcal{C}(A, I)$ for every system A, compatible with the monoidal structure as below;*

$$\overset{=}{\top}_{A\otimes B} = \overset{=}{\top}_A \otimes \overset{=}{\top}_B \tag{2}$$

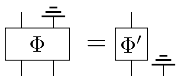
$$\overset{=}{\top}_I = \mathrm{id}_I \tag{3}$$

PC2. *The dimension $d_A := \underset{=}{\bot}_A \,⨾\, \overset{=}{\top}_A$ is invertible for all non-zero A;*

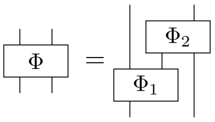
PC3. *$\mathcal{C}$ has enough causal states: $\forall f, g : A \to B. \left(\forall \rho \in \left\{\overset{=}{\top}_A\right\}^* .\rho \,⨾\, f = \rho \,⨾\, g\right) \Rightarrow f = g$;*

PC4. *Causal one-way signalling processes on first-order types factorise: for any causal $\Phi : A \otimes B \to C \otimes D$,*

$\left(\exists \Phi' : A \to C \text{ causal.} \right.$



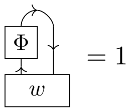
$\left. \right) \Rightarrow \left(\exists Z, \Phi_1 : A \to C \otimes Z \text{ causal}, \quad \Phi_2 : Z \otimes B \to D \text{ causal.} \right.$



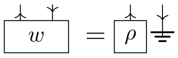
$\left. \right) \tag{4}$

PC5. *For all $w : I \to A \otimes B^*$:*

$\left(\forall \Phi : A \to B \text{ causal.} \right.$



$\left. \right) \Rightarrow \left(\exists \rho : I \to A \text{ causal.} \right.$



$\left. \right) \tag{5}$

Note that in [19], PC4 and PC5 are rolled into a single axiom (C4), which is then proven equivalent to PC4 and PC5.

▶ **Example 4.** Mat$[\mathbb{R}^+]$ is a precausal category, whose objects are natural numbers and whose morphisms $M : m \to n$ are $n \times m$ matrices. Then, $\otimes$ is given by tensor product (a.k.a. Kronecker product) of matrices and consequently the tensor unit is the natural number 1. Hence states are column vectors, discarding maps $\bar{\bar{\top}}_m : m \to 1$ are given by row vectors of all 1's, and the causality condition for states $\rho \mathbin{⨾} \bar{\bar{\top}} = \mathrm{id}_1$ imposes the condition that the entries of ρ sum to 1. The conditions PC1, PC2, and PC3 are easily checked, whereas PC4 and PC5 follow from the product rule for conditional probability distributions (see [19]).

▶ **Example 5.** CP is a precausal category, whose objects are algebras $\mathcal{L}(H)$ of linear operators from a finite-dimensional Hilbert space to itself, and whose morphisms are completely positive maps (CP-maps). $\otimes$ is given by tensor product and consequently the tensor unit is the 1D algebra $\mathcal{L}(\mathbb{C}) \cong \mathbb{C}$. States are CP-maps $\mathbb{C} \to \mathcal{L}(H)$, which correspond to positive semidefinite operators $\rho \in \mathcal{L}(H)$. Discarding is given by the trace, hence causal states are the trace-1 positive semidefinite operators, a.k.a. quantum (mixed) states. Again the conditions PC1, PC2, and PC3 are easily checked, whereas PC4 and PC5 follow from the essential uniqueness of purification for CP-maps (see [19]).

Given a precausal category $\mathcal{C}$, we can refine to a category Caus$[\mathcal{C}]$ which equips each object with a nice set of states that should be considered "normalised" (or "causal") and restricts to the morphisms that preserve them. First, we'll define what it means for a set of states to be suitably nice.

▶ **Definition 6.** *A set $c \subseteq \mathcal{C}(I, A)$ is closed if $c = c^{**}$ and flat if either there exist invertible scalars λ, μ such that $\lambda \cdot \underline{\underline{\perp}}_A \in c$ and $\mu \cdot \bar{\bar{\top}}_A \in c^*$ or A is a zero system.*

▶ **Definition 7.** *Given a precausal category $\mathcal{C}$, the category* Caus$[\mathcal{C}]$ *has as objects pairs $\mathbf{A} = (A, c_{\mathbf{A}} \subseteq \mathcal{C}(I, A))$ where $c_{\mathbf{A}}$ is closed and flat. A morphism $f : \mathbf{A} \to \mathbf{B}$ is a morphism $f : A \to B$ in $\mathcal{C}$ such that $\forall \rho \in c_{\mathbf{A}}. \rho \mathbin{⨾} f \in c_{\mathbf{B}}$.*

Caus$[\mathcal{C}]$ inherits a monoidal structure and monoidal closure from $\mathcal{C}$, from which one can show that it is $*$-autonomous [19]. This is in fact a full subcategory of a particular double gluing construction (specifically the tight orthogonality subcategory of the double glueing construction using the $\{1\}$-focussed orthogonality [16]). Aside from the flatness restriction, this is therefore a relatively well-known means of constructing models of linear logic.

$$\begin{array}{lrl}
\text{First-order states} & \mathbf{A^1} := & \left(A, \left\{\bar{\bar{\top}}_A\right\}^*\right) \\
\text{Dual} & \mathbf{A}^* := & (A^*, c_{\mathbf{A}}^*) \\
\text{Tensor product} & \mathbf{A} \otimes \mathbf{B} := & (A \otimes B, \{\rho_A \otimes \rho_B | \rho_A \in c_{\mathbf{A}}, \rho_B \in c_{\mathbf{B}}\}^{**}) \\
\text{Par} & \mathbf{A} \parr \mathbf{B} := & (\mathbf{A}^* \otimes \mathbf{B}^*)^* \\
\text{Internal hom} & \mathbf{A} \multimap \mathbf{B} := & \mathbf{A}^* \parr \mathbf{B} \\
\text{Monoidal unit} & \mathbf{I} := & (I, \{1\})
\end{array}$$

The intuition between the two monoidal products is that $\mathbf{A} \otimes \mathbf{B}$ is the closure of the space of local processes and hence we can compose the $\mathbf{A}$ and $\mathbf{B}$ components however we choose, whereas $\mathbf{A} \parr \mathbf{B}$ is the space of bipartite processes that are normalised in local contexts so we generally cannot compose the components, just act locally on each side. Both share the same unit $\mathbf{I} = \mathbf{I}^*$ and there is a canonical inclusion $\mathbf{A} \otimes \mathbf{B} \hookrightarrow \mathbf{A} \parr \mathbf{B}$ making Caus$[\mathcal{C}]$ into an isomix category.

From these operators, we can build objects capturing the set of processes compatible with some common causal structures. For example, $(\mathbf{A^1} \multimap \mathbf{B^1}) \parr (\mathbf{C^1} \multimap \mathbf{D^1})$ includes all causal bipartite first-order processes, $(\mathbf{A^1} \multimap \mathbf{B^1}) \otimes (\mathbf{C^1} \multimap \mathbf{D^1})$ is the subset of those that are non-signalling [19, Theorem 6.2], and $(\mathbf{A^1} \multimap \mathbf{B^1}) \multimap (\mathbf{C^1} \multimap \mathbf{D^1})$ includes all 2-combs that map causal channels $\mathbf{A^1} \multimap \mathbf{B^1}$ to causal channels $\mathbf{C^1} \multimap \mathbf{D^1}$.

3 Additive Precausal Categories

When investigating higher order causal theories, it is useful to strengthen the definition of a precausal category in a handful of ways. For the remainder of this paper, we will adapt the definition of $\mathrm{Caus}[\mathcal{C}]$ to be built from an additive precausal category $\mathcal{C}$.

▶ **Definition 8.** *Let $\mathcal{C}$ be a compact closed category with products (and hence biproducts and additive enrichment [15]). $\mathcal{C}$ is an additive precausal category if:*

APC1. *$\mathcal{C}$ has a discarding process $\overset{=}{\top}_A \in \mathcal{C}(A, I)$ for every system A, compatible with the monoidal and biproduct structures as below;*

$$\overset{=}{\top}_{A\otimes B} = \overset{=}{\top}_A \otimes \overset{=}{\top}_B \tag{6}$$

$$\overset{=}{\top}_I = \mathrm{id}_I \tag{7}$$

$$\overset{=}{\top}_{A\oplus B} = \left[\overset{=}{\top}_A, \overset{=}{\top}_B\right] \tag{8}$$

APC2. *The dimension $d_A := \underset{=}{\bot}_A \fatsemi \overset{=}{\top}_A$ is invertible for all non-zero A;*

APC3. *Each object $A \in \mathrm{Ob}(\mathcal{C})$ has a finite causal basis: some $\{\rho_i\}_i \subseteq \left\{\overset{=}{\top}_A\right\}^*$ such that $\forall B.\forall f, g \in \mathcal{C}(A, B).(\forall i.\ \rho_i \fatsemi f = \rho_i \fatsemi g) \Rightarrow f = g$.*

APC4. *Addition of scalars is cancellative ($\forall x, y, z.\ x+z = y+z \Rightarrow x = y$), totally pre-ordered ($\forall x, y.\exists z.\ x + z = y \vee x = y + z$), and all non-zero scalars have a multiplicative inverse.*

APC5. *All effects have a complement with respect to discarding: for any $\pi \in \mathcal{C}(A, I)$, there exists some $\pi' \in \mathcal{C}(A, I)$ and scalar λ such that $\pi + \pi' = \lambda \cdot \overset{=}{\top}_A$.*

The first 3 axioms relate closely to the corresponding ones in Definition 3, whereas the last two are quite different in flavour, and are in some sense more elementary, as their proofs don't rely on any particularly deep facts about our main classical and quantum examples (see Examples 13 and 15 below).

The axioms APC1 and APC2 above are essentially identical to the corresponding axioms in Definition 3 of precausal categories, with the additional requirement that discarding be compatible with biproducts as well as tensor products.

APC3 is a strengthening of condition PC3. Rather than requiring us to check a pair of processes agree on *all* causal states to be equal, we only require agreement on some fixed finite set of states. In other other words, each system has a set of states that behaves like a basis spanning all of the others, for the purposes of distinguishing maps. In the quantum foundations literature, this is sometimes called a *fiducial set of states*.

APC4 says that the semiring of scalars $\mathcal{C}(I, I)$ behaves somewhat like the set of non-negative real numbers $\mathbb{R}^+$. While the scalars need not be a field (indeed our main example $\mathbb{R}^+$ is not), any field satisfies this axiom as well. We do however exclude categories of non-determinstic processes such as Rel, since the scalars are boolean values with addition given by the (non-cancellative) operation of disjunction.

Note that, with the help of bases (APC3), we can promote additive cancellativity of scalars to additive cancellativity for all processes.

▶ **Proposition 9.** *In an additive precausal category:*

$$\forall f, g, h \in \mathcal{C}(A, B).\ f + h = g + h \Rightarrow f = g \tag{APC5a}$$

This condition allows us to define the free *subtractive closure* $\mathrm{Sub}(\mathcal{C})$ which extends $\mathcal{C}$ with all negatives and prove that there exists a faithful embedding $[-] : \mathcal{C} \to \mathrm{Sub}(\mathcal{C})$. More details on the explicit construction of $\mathrm{Sub}(\mathcal{C})$, its properties, and the embedding functor are given in the full version of this paper [26].

The utility of freely introducing negatives is summarised in the following proposition.

▶ **Proposition 10.** *For a precausal category $\mathcal{C}$, the scalars $K := \mathrm{Sub}(\mathcal{C})(I, I)$ are a field, and hence $\mathrm{Sub}(\mathcal{C})$ is enriched over K-vector spaces.*

In particular, we can therefore treat processes in $\mathcal{C}(A, B)$ as being embedded in an ambient vector space $\mathrm{Sub}(\mathcal{C})(A, B)$. So, while we might not be able to make all linear algebraic constructions directly in $\mathcal{C}$, we can do so in $\mathrm{Sub}(\mathcal{C})$. An important example of this is the ability to extend any independent set of states to a basis of states in $\mathcal{C}$ as well as its corresponding dual basis of effects in $\mathrm{Sub}(\mathcal{C})$.

▶ **Lemma 11.** *Given any set of morphisms in $\mathcal{C}(A, B)$ that are linearly independent in $\mathrm{Sub}(\mathcal{C})$, they can be extended to a basis in $\mathcal{C}$ with a dual basis in $\mathrm{Sub}(\mathcal{C})$.*

While the dual basis in $\mathrm{Sub}(\mathcal{C})$ may not be physically meaningful (e.g. in the classical case it will contain vectors with negative probabilities), it will be a useful mathematical tool for working with the morphisms in $\mathcal{C}$.

APC5 allows us to interpret effects (up to some renormalisation) as testing some predicate. To see how this works, first assume for simplicity that $\lambda = \mathrm{id}_I$. For a type A, we can think of $\pi : A \to I$ as some predicate over A, and π' as its negation. For some causal state ρ, we can think of the composition $p_1 := \rho \mathbin{⨟} \pi$ as the probability that π is true for ρ and $p_2 := \rho \mathbin{⨟} \pi'$ as the probability that π is false. The fact that $\pi + \pi' = \overset{=}{\top}$ lets us conclude that those probabilities sum to 1:

$$p_1 + p_2 = \rho \mathbin{⨟} \pi + \rho \mathbin{⨟} \pi' = \rho \mathbin{⨟} (\pi + \pi') = \rho \mathbin{⨟} \overset{=}{\top} = \mathrm{id}_I$$

If $\lambda \neq \mathrm{id}_I$, the previous reasoning holds after re-normalising, i.e. replacing π and π' with $\lambda^{-1} \cdot \pi$ and $\lambda^{-1} \cdot \pi'$.

Thanks to compact closure, we can promote APC5 to a property about all processes. Namely, any process $f : A \to B$ has a complement f' where, up to re-normalisation, $f + f'$ gives the uniform noise process.

▶ **Proposition 12.** *For any $f : A \to B$ in an additive precausal category, there exists $f' : A \to B$ and a scalar λ such that:*

$$f + f' = \lambda \cdot \overset{=}{\top}_A \mathbin{⨟} \underset{=}{\bot}_B \tag{APC5a}$$

▶ **Example 13.** $\mathrm{Mat}[\mathbb{R}^+]$ defined as in Example 4 is also an additive precausal category, where $\oplus$ is given by the direct sum of matrices. The standard basis of unit vectors gives a basis for APC3, the semiring of scalars $\mathrm{Mat}[\mathbb{R}^+](I, I) \cong \mathbb{R}^+$ satisfies APC4, and for APC5, we just need to choose a suitably large λ such that $\pi' := \lambda \cdot \overset{=}{\top}_A - \pi$ contains all positive numbers.

▶ **Example 14.** In addition to $\mathbb{R}^+$, we can construct an additive precausal category $\mathrm{Mat}[K]$ for any field of characteristic 0. In particular, $\mathrm{Mat}[\mathbb{R}]$ is an additive precausal category that is identical to $\mathrm{Mat}[\mathbb{R}^+]$ but without any positivity constraint, describing affine or "quasi-probabilistic" maps where negative probabilities are permitted. In this case, the subtractive closure gives an equivalent category to $\mathrm{Mat}[\mathbb{R}]$ itself.

▶ **Example 15.** The quantum example is very nearly the category CP, as defined in Example 5, but CP doesn't have biproducts. If we freely add biproducts, we obtain a category CP^* whose objects are all finite-dimensional C*-algebras (or equivalently, algebras of the form $\mathcal{L}(H_1) \oplus \ldots \oplus \mathcal{L}(H_k)$) and completely positive maps. Discarding is again given by the trace operator, so APC1 and APC2 are straightforward to verify. For APC3, we can fix a (non-orthogonal) basis of states for each type. As in the classical case, the scalars are $\mathbb{R}^+$, so APC4 is immediate and since $\bar{\bar{\top}}_A$ is an interior point in the cone of positive effects, π' can defined as $\lambda \cdot \bar{\bar{\top}}_A - \pi$ for suitably large λ.

Given an additive precausal category, we can apply the Caus construction in the same way as before. However, it may now be easier to devise interesting closed sets or interpret the impact of the closure operator since it just corresponds to taking affine combinations of states. For this to make sense, we should say precisely what we mean to take affine combinations of states in $\mathcal{C}$.

▶ **Definition 16.** *For a set of states $c \subseteq \mathcal{C}(I, A)$, we define sets $\mathsf{aff}(c) \subseteq \mathrm{Sub}(\mathcal{C})(I, A)$ and $\mathsf{aff}^+(c) \subseteq \mathcal{C}(I, A)$ as follows, for $K := \mathrm{Sub}(\mathcal{C})(I, I)$:*

$$\mathsf{aff}(c) := \left\{ \rho \,\middle|\, \exists \{\rho_i\}_i \subseteq \mathcal{C}(I, A), \{\lambda_i\}_i \subseteq K.\ \sum_i \lambda_i = \mathrm{id}_I, \rho = \sum_i \lambda_i \cdot [\rho_i] \right\}$$

$$\mathsf{aff}^+(c) := \{\rho \,|\, \exists \rho' \in \mathsf{aff}(c).\ \rho' = [\rho]\}$$

If we identify the set $\mathcal{C}(I, A)$ with its image under $[-]$, we can think if $\mathsf{aff}^+(c)$ as the intersection of the affine closure of c with the set $\mathcal{C}(I, A) \subseteq \mathrm{Sub}(\mathcal{C})(I, A)$ of “positive” states embedded in the subtractive closure. In the classical and quantum case, $\mathsf{aff}^+(-)$ arises from taking all of the affine combinations of elements of c, then intersecting the resulting set with the positive cone of (unnormalised) probability distributions or quantum states, respectively.

▶ **Theorem 17.** *Given any flat set $c \subseteq \mathcal{C}(I, A)$ for a non-zero A, $c^{**} = \mathsf{aff}^+(c)$.*

This characterises the tensor space $c_{\mathbf{A} \otimes \mathbf{B}} = \{\rho_A \otimes \rho_B | \rho_A \in c_{\mathbf{A}}, \rho_B \in c_{\mathbf{B}}\}^{**}$ in $\mathrm{Caus}[\mathcal{C}]$ as the affine closure of the separable states, from which we can prove the following property.

▶ **Theorem 18.** *If $\mathbf{A} = (A, c_{\mathbf{A}})$ with $c_{\mathbf{A}} = \{\mu \cdot \text{⏚}_A\}$ for any non-zero A, then every $h \in c_{\mathbf{A} \otimes \mathbf{B}}$ is a product morphism of the form $\mu \cdot \text{⏚}_A \otimes g$ for some $g \in c_{\mathbf{B}}$.*

This captures what [27] refers to as the principle of “no interaction with trivial degrees of freedom”. In particular, it recovers the precausal category axiom PC5 by showing that every state of $(\mathbf{A}^\mathbf{1} \multimap \mathbf{B}^\mathbf{1})^* = (\mathbf{A}^{\mathbf{1}^*} \parr \mathbf{B}^\mathbf{1})^* = \mathbf{A}^\mathbf{1} \otimes \mathbf{B}^{\mathbf{1}^*}$ decomposes into a product of a state of $\mathbf{A}^\mathbf{1}$ (i.e. a causal state) and ⏚_{B^*}.

It should be noted that we have completely dropped condition PC4 on our underlying category of basic processes. In Section 5 we will recover a slightly weaker version of this condition by the equivalence of one-way signalling and the affine closure of semi-localisability (Theorem 30). Fortunately, we can still reuse the same proof to show that $\mathrm{Caus}[\mathcal{C}]$ is $*$-autonomous for additive precausal $\mathcal{C}$ since it doesn’t rely on PC4.

4 Additive Types

We may also lean on the relation to the double glueing construction to add type constructors corresponding to the additives of linear logic. In categorical models of linear logic, additive conjunction of types is captured by cartesian product and additive disjunction by coproduct, satisfying a De Morgan duality with one another [20]. In terms of resources, these represent a classical choice: an instance of $A \times B$ is a single resource unit that we can choose to be used either as an instance of A or an instance of B, whereas an instance of $A + B$ is a single resource unit which is fixed on creation as either a unit of A or a unit of B.

The concept of classical choice is often built into operational theories in the form of probability distributions, classical outcomes of tests, and conditional tests. This is typically deemed essential for any experimentalist who is bound to classical data to interact with and make inference from an experiment. On the other hand, the Caus construction concerns the underlying systems present in the physical theory without consideration of any classical agent. We can recover finite-outcome random variables by incorporating binary classical choice through new additive type constructors, i.e. finding constructions for products and coproducts in $\mathrm{Caus}[\mathcal{C}]$.

▶ **Definition 19.** *Given types* $\mathbf{A} = (A, c_{\mathbf{A}}), \mathbf{B} = (B, c_{\mathbf{B}})$ *in* $\mathrm{Caus}[\mathcal{C}]$*, we define* $\mathbf{A} \times \mathbf{B} := (A \oplus B, c_{\mathbf{A}\times\mathbf{B}})$ *and* $\mathbf{A} \oplus \mathbf{B} := (A \oplus B, c_{\mathbf{A}\oplus\mathbf{B}})$ *where:*

$$\begin{aligned} c_{\mathbf{A}\times\mathbf{B}} &= \left(\{p_A \mathbin{⨾} \pi_A | \pi_A \in c_{\mathbf{A}}^* \subseteq \mathcal{C}(A, I)\} \cup \{p_B \mathbin{⨾} \pi_B | \pi_B \in c_{\mathbf{B}}^* \subseteq \mathcal{C}(B, I)\}\right)^* \\ &= \{\langle \rho_A, \rho_B \rangle | \rho_A \in c_{\mathbf{A}}, \rho_B \in c_{\mathbf{B}}\} \end{aligned} \tag{9}$$

$$\begin{aligned} c_{\mathbf{A}\oplus\mathbf{B}} &= \left(\{\rho_A \mathbin{⨾} \iota_A | \rho_A \in c_{\mathbf{A}}\} \cup \{\rho_B \mathbin{⨾} \iota_B | \rho_B \in c_{\mathbf{B}}\}\right)^{**} \\ &= \{[\pi_A, \pi_B] \,|\pi_A \in c_{\mathbf{A}}^* \subseteq \mathcal{C}(A, I), \pi_B \in c_{\mathbf{B}}^* \subseteq \mathcal{C}(B, I)\}^* \end{aligned} \tag{10}$$

▶ **Lemma 20.** *The alternative definitions of* $c_{\mathbf{A}\times\mathbf{B}}$ *and* $c_{\mathbf{A}\oplus\mathbf{B}}$ *are equivalent; that is, Equations 9 and 10 hold.*

▶ **Corollary 21.** *The operators* $\times$ *and* $\oplus$ *are De Morgan duals under* $(-)^*$*.*

▶ **Proposition 22.** $\mathbf{A} \times \mathbf{B}$ *is a categorical product in* $\mathrm{Caus}[\mathcal{C}]$*.*

▶ **Proposition 23.** $\mathbf{A} \oplus \mathbf{B}$ *is a categorical coproduct in* $\mathrm{Caus}[\mathcal{C}]$*.*

The zero object 0 has a unique state $0_{I,0} \in \mathcal{C}(I, 0)$ by terminality and a unique effect $0_{0,I} \in \mathcal{C}(0, I)$ by initiality. There are only two candidates for causal sets: the empty set $\emptyset$ and the singleton $\{0_{I,0}\}$. These have roles in our causal category as the additive units.

▶ **Proposition 24.** *The initial object in* $\mathrm{Caus}[\mathcal{C}]$ *is* $\mathbf{0} := (0, \emptyset)$ *and the terminal object is* $\mathbf{1} := (0, \{0_{I,0}\})$*. Furthermore, they are duals of each other and are units for* $\oplus$ *and* $\times$ *respectively.*

▶ Remark 25. The product and coproduct constructions are only partially-defined as they may not always yield flat sets of states when incorporating the initial or terminal. Specifically, given any $\mathbf{A}$ on a non-zero system, we have

$$\begin{array}{lclclcl} c_{\mathbf{A}\times\mathbf{0}} & = & \emptyset & & c_{\mathbf{A}\times\mathbf{0}}^* & \simeq & \mathcal{C}(A, I) \\ c_{\mathbf{A}\oplus\mathbf{1}} & \simeq & \mathcal{C}(I, A) & & c_{\mathbf{A}\oplus\mathbf{1}}^* & = & \emptyset \end{array} \tag{11}$$

though it may be possible that a careful weakening of the flatness condition may permit this more generally without sacrificing some of the other results of this paper.

Thinking of first-order types as describing systems with no input (i.e. no choice in how to consume them), both the product and coproduct have interesting interactions with first-order types because of where the classical choice happens. For coproducts, the choice is already fixed in the creation of a state so we expect it to preserve the first-order property. However, products introduce freedom of choice in effects allowing us to view the projectors as inputs to the system dictating whether it should prepare the left or the right state.

▶ **Proposition 26.** *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are both first-order types, then so is* $\mathbf{A} \oplus \mathbf{B}$*.*

▶ **Proposition 27.** *If* A *and* B *are non-zero, then* $\mathbf{A} \times \mathbf{B}$ *is never a first-order type.*

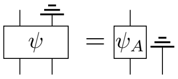
We have both $*$-autonomy and all finite products and coproducts, which is all the evidence required to show that Caus[$\mathcal{C}$] is a model of linear logic with additives. For both the multiplicative and additive structures, we found that the Caus construction took a degenerative categorical structure and generated non-degenerate structures from them. For example, the compact closure of $\mathcal{C}$ means that it is a $*$-autonomous category in which the two monoidal products are the same, but Caus[$\mathcal{C}$] is $*$-autonomous with distinct $\otimes$ and $\parr$. Similarly, the construction for the additives takes biproducts/the zero object and yields distinct products and coproducts/initial and terminal objects. The degenerate exception here is that $\mathbf{I}$ is still the unit for both $\otimes$ and $\parr$.

5 One-Way Signalling Types

When examining multi-partite systems, causal structures can be investigated from the perspective of information signalling (which parties can observe changes to another party's inputs) or decompositions (does the channel admit a decomposition into local channels compatible with some configuration of time- and space-like separations between the parties). In the bipartite case, we can compare these perspectives with the examples of one-way signalling and semi-localisable channels.

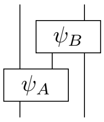
The one-way signalling causal structure refers to a bipartite system (say, between Alice and Bob) in which causal influence may only exist in one direction. There is a standard definition given in causality literature for one-way signalling for quantum channels.

▶ **Definition 28.** *A bipartite process is one-way signalling (Alice to Bob) if discarding Bob's output admits a decomposition into local processes.*



Semi-localisability is an alternative to one-way signalling for expressing compatibility with a time-like separation of parties, giving a constructive example of how the combined channel can be decomposed into local operations.

▶ **Definition 29.** *A bipartite channel between Alice and Bob is semi-localisable (with Alice before Bob) if it decomposes into local processes with a channel from Alice to Bob.*

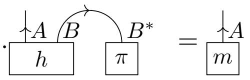


Both of these properties state that the channel is compatible with the setting where Bob is in Alice's future light cone. It is important to note that both are judgements of compatibility with a causal structure rather than inference of any necessary causal relationship between Alice and Bob - completely local processes trivially satisfy these properties but obviously have no causal influence between the parties.

These definitions are specific to first-order channels, but we can consider situations where Alice and Bob have higher-order systems $\mathbf{A}, \mathbf{B} \in \text{Ob}(\text{Caus}[\mathcal{C}])$ representing some more complex interaction with their environments. We will describe these properties as constructions on the causal sets.

The essence of one-way signalling is that Alice's local effect is independent of any input given to Bob by his context:

$$c_{\mathbf{A}} < c_{\mathbf{B}} := \left\{ h \in c_{\mathbf{A} \parr \mathbf{B}} \,\middle|\, \exists m \in c_{\mathbf{A}} . \forall \pi \in c_{\mathbf{B}^*} . \right.$$

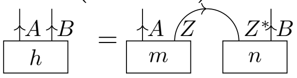


$$\left. \vphantom{\int} \right\} \qquad (12)$$

When this property holds, we refer to m as the *(left-)residual* of h (with *right-residual* for the corresponding component in the symmetrically-defined $c_{\mathbf{A}} > c_{\mathbf{B}}$). The residual represents the constant local effect Alice observes regardless of inputs provided to Bob. In the case where $\mathbf{A}$ and $\mathbf{B}$ both describe first-order channels, this exactly reduces to Definition 28, since the only causal contexts for a causal channel is to provide an arbitrary state at the input and discard the output.

For semi-localisability, it is not enough to consider a factorisation by any system, but specifically by a first-order system:

$$c_{\mathbf{A}} \lhd c_{\mathbf{B}} := \left\{ h \in c_{\mathbf{A} \parr \mathbf{B}} \,\middle|\, \exists \mathbf{Z} = \left(Z, \left\{ \bar{\top}_Z \right\}^* \right), m \in c_{\mathbf{A} \parr \mathbf{Z}}, n \in c_{\mathbf{Z}^* \parr \mathbf{B}} . \right.$$



$$\left. \vphantom{\int} \right\} \qquad (13)$$

This is because using a higher-order system (e.g. the dual of a first-order system) may allow information to flow in the opposite direction. Again, we recover the original definition of semi-localisability if we fix $\mathbf{A}$ and $\mathbf{B}$ to types of first-order channels (up to our encoding of channels via the Choi-Jamiołkowski isomorphism).

Intuitively, based on the interpretation of the sequence operator in a BV-category as representing an ordered combination of systems, both of these would be good candidates for sequence operator in Caus$[\mathcal{C}]$. It is also interesting to compare them to sequence operators from other BV-categories. The following construction is inspired by that of the BV-category of probabilistic coherence spaces [3]:

$$c_{\mathbf{A}} \obslash c_{\mathbf{B}} := \left\{ h \in c_{\mathbf{A} \parr \mathbf{B}} \,\middle|\, \begin{array}{l} \exists \mathcal{I}, \{f_i\}_{i \in \mathcal{I}} \subseteq \mathrm{Sub}(\mathcal{C})(I, A \otimes B), \{g_i\}_{i \in \mathcal{I}} \subseteq c_{\mathbf{B}}, f \in c_{\mathbf{A}} . \\ {[h]} \sim \sum_{i \in \mathcal{I}} f_i \otimes [g_i] \wedge [f] \sim \sum_{i \in \mathcal{I}} f_i \end{array} \right\} \qquad (14)$$

Note this definition refers to maps in the subtractive closure Sub$(\mathcal{C})$ to make use of the affine-linear structure.

A core result in the field of causal structures is the equivalence of one-way signalling and semi-localisability for first-order channels [8]. In the setting provided by the Caus construction, we find a higher-order generalisation of this, where all three of these definitions coincide up to affine closure, as well as a proof that they are self-dual properties.

▶ **Theorem 30.** $c_{\mathbf{A}} < c_{\mathbf{B}} = \left(c_{\mathbf{A}}^* < c_{\mathbf{B}}^*\right)^* = \left(c_{\mathbf{A}} \lhd c_{\mathbf{B}}\right)^{**} = c_{\mathbf{A}} \obslash c_{\mathbf{B}}$

$c_{\mathbf{A}} < c_{\mathbf{B}}$ is closed, since it the dual of another set, and flat, since $c_{\mathbf{A} \otimes \mathbf{B}} \subseteq c_{\mathbf{A}} < c_{\mathbf{B}}$ gives the uniform state and $c_{\mathbf{A} \parr \mathbf{B}}^* \subseteq (c_{\mathbf{A}} < c_{\mathbf{B}})^*$ gives discarding. We can therefore elevate it to a genuine object $\mathbf{A} < \mathbf{B} := (A \otimes B, c_{\mathbf{A}} < c_{\mathbf{B}})$ in Caus$[\mathcal{C}]$.

We can go further into the categorical structure induced by this type constructor and show that it adds another monoidal structure with a weak interchange with both $\otimes$ and $\parr$, turning Caus$[\mathcal{C}]$ into a model of BV-logic.

▶ **Definition 31.** *Given a symmetric, linearly distributive category $\mathcal{D}$, a weak interchange is an additional monoidal structure $(\mathcal{D}, \obslash, I_{\obslash})$ with natural transformations*

$$\begin{array}{llll} w_{\otimes} & : (R \obslash U) \otimes (T \obslash V) \to (R \otimes T) \obslash (U \otimes V) & w_{I_{\otimes}} & : I_{\otimes} \to I_{\otimes} \obslash I_{\otimes} \\ w_{\parr} & : (C \parr E) \obslash (D \parr F) \to (C \obslash D) \parr (E \obslash F) & w_{I_{\parr}} & : I_{\parr} \obslash I_{\parr} \to I_{\parr} \end{array}$$

which are compatible with the structure isomorphisms of $(\mathcal{D}, \otimes, I_\otimes)$ *and* $(\mathcal{D}, \parr, I_\parr)$ *and the distributive structure (we refer the reader to Blute, Panangaden, and Slavnov [3, Definition 4.1] for the full set of conditions).*

A BV*-category is a symmetric linearly distributive category with a weak interchange and an isomorphism* $m : I_\obslash \to I_\otimes$ *such that* m *is an isomix map and* m^{-1} *is a counit for* $w_{I_\otimes}$*:*

$$(\mathrm{id}_{I_\obslash} \otimes m) \fatsemi \rho^\otimes_{I_\obslash} = (m \otimes \mathrm{id}_{I_\obslash}) \fatsemi \lambda^\otimes_{I_\obslash} : I_\obslash \otimes I_\obslash \to I_\obslash$$

$$w_{I_\otimes} \fatsemi (m^{-1} \obslash \mathrm{id}_{I_\otimes}) \fatsemi \lambda^\obslash_{I_\otimes} = \mathrm{id}_{I_\otimes} = w_{I_\otimes} \fatsemi (\mathrm{id}_{I_\otimes} \obslash m^{-1}) \fatsemi \rho^\obslash_{I_\otimes} : I_\otimes \to I_\otimes$$

▶ **Theorem 32.** $\mathrm{Caus}[\mathcal{C}]$ *is a* BV*-category.*

The equivalence of one-way signalling and (the affine closure of) semi-localisability show that any directed information signalling will always exhibit an equivalent construction where the information transfer is mediated by first-order types. Moreover, first-order types (or scaled versions thereof) are *exactly* those that can carry information in one direction.

▶ **Theorem 33.** $\mathbf{A}^* \parr \mathbf{A} = \mathbf{A}^* < \mathbf{A} \Leftrightarrow |c^*_{\mathbf{A}}| = 1.$

This result presents a characterisation of first-order system types that can be interpreted in any BV-category, which may be a useful lemma in characterising which categories arise as $\mathrm{Caus}[\mathcal{C}]$ for some additive precausal $\mathcal{C}$.

6 Non-Signalling Systems

The one-way signalling structure is not the only interesting causal structure on a bipartite system. For example, the non-signalling causal structure is a symmetric property about the statistical independence of the different parties.

▶ **Definition 34.** *A bipartite process is non-signalling if it satisfies the one-way signalling condition in both directions.*

This is a necessary condition for compatibility with the setting where Alice and Bob are space-like separated. However, this is not a sufficient condition since there exist non-signalling processes, such as a quantum implementation of a PR box, that cannot be factorised into local processes with a shared history [23].

In [19], the tensor product of first-order channel types $(\mathbf{A^1} \multimap \mathbf{B^1}) \otimes (\mathbf{C^1} \multimap \mathbf{D^1})$ was shown to exactly contain those bipartite channels that are non-signalling [19]. This proof relied on some properties that do not generalise beyond first-order channels, such as the ability to apply PC5 to decompose causal contexts for channels into a causal input state and discarding the output. Between Gutoski [12] and Chiribella et al. [9], it was shown that the space of non-signalling channels in quantum theory can be characterised as the affine closure of product channels. By translating this proof into categorical terms, we can generalise the result to hold for arbitrary higher-order systems in $\mathrm{Caus}[\mathcal{C}]$ for any additive precausal $\mathcal{C}$.

▶ **Theorem 35.** $(c_{\mathbf{A}} < c_{\mathbf{B}}) \cap (c_{\mathbf{A}} > c_{\mathbf{B}}) = c_{\mathbf{A} \otimes \mathbf{B}}.$

7 Conclusion

By extending the assumptions on the base category with additive structure, the Caus construction yields a BV-category with additives within which the characterisation of first-order types resembles the causality condition expressed as an equation of types. We also obtain general characterisations of $(-)^{**}$ as affine closure and the tensor product as the space of non-signalling bipartite processes. A number of the proofs are similar to those used to show that probabilistic coherence spaces form a BV-category [3], and the characterisation of non-signalling processes as affine closure of local processes for first-order channels in quantum theory [12]. As BV-logic is known to prove a strict subset of the theorems of pomset logic [21], it remains for future work to investigate whether the Caus construction yields models of the latter or if it is limited to BV-logic.

We recover almost all of the precausal conditions from an additive precausal category. Instead of the equivalence of one-way signalling and semi-localisability for first-order channels (Condition PC4), we obtained equivalence with the affine closure of semi-localisability for arbitrary higher-order systems (Theorem 30). Since existing proofs of PC4 for precausal categories rely on the essential uniqueness of purification, it may be possible to strengthen this result in future work by incorporating notions of purity and purification into this framework.

There are still other interesting process theories that cannot be considered as either the base category or the result of the Caus construction in which it would be interesting to analyse constructions for one-way signalling and their logical role. For example, settings with infinite-dimensional systems are rarely compact closed, real quantum mechanics doesn't have enough causal states (since it is compact closed but doesn't admit local discrimination), and Rel fails Condition PC5. We needn't expect the results of this paper to generalise to other theories since (non-deterministic) coherence spaces form a BV-category using a similar construction to $\mathbf{A} \obslash \mathbf{B}$ for the non-commutative operator [3], but this is distinct from the naive adaptation of $\mathbf{A} < \mathbf{B}$ (instead of a constant residual, every local effect must be a subset of some constant clique) which is not self-dual in this category.

References

1 Ämin Baumeler, Adrien Feix, and Stefan Wolf. Maximal incompatibility of locally classical behavior and global causal order in multiparty scenarios. *Physical Review A - Atomic, Molecular, and Optical Physics*, 90(4):42106, October 2014. doi:10.1103/PhysRevA.90.042106.

2 Alessandro Bisio and Paolo Perinotti. Theoretical framework for higher-order quantum theory. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 475(2225), May 2019. doi:10.1098/rspa.2018.0706.

3 Richard Blute, Prakash Panangaden, and Sergey Slavnov. Deep inference and probabilistic coherence spaces. *Applied Categorical Structures*, 2012. doi:10.1007/s10485-010-9241-0.

4 Richard F. Blute, Ivan T. Ivanov, and Prakash Panangaden. Discrete quantum causal dynamics. *International Journal of Theoretical Physics*, 42(9):2025–2041, September 2003. doi:10.1023/A:1027335119549.

5 Paulo J. Cavalcanti, John H. Selby, Jamie Sikora, and Ana Belén Sainz. Simulating all multipartite non-signalling channels via quasiprobabilistic mixtures of local channels in generalised probabilistic theories. *arxiv.org*, 2022. arXiv:2204.10639.

6 G Chiribella, G. M. D'Ariano, and P. Perinotti. Quantum circuit architecture. *Physical Review Letters*, 101(6), August 2008. doi:10.1103/PhysRevLett.101.060401.

7 Giulio Chiribella, Giacomo Mauro D'Ariano, and Paolo Perinotti. Theoretical framework for quantum networks. *Physical Review A*, 80(2):022339, August 2009. doi:10.1103/PhysRevA.80.022339.

8 Giulio Chiribella, Giacomo Mauro D'Ariano, and Paolo Perinotti. Probabilistic theories with purification. *Physical Review A*, 81(6):062348, June 2010. doi:10.1103/PhysRevA.81.062348.

9 Giulio Chiribella, Giacomo Mauro D'Ariano, Paolo Perinotti, and Benoit Valiron. Quantum computations without definite causal structure. *Physical Review A*, 88(2):022318, August 2013. doi:10.1103/PhysRevA.88.022318.

10 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes*. Cambridge University Press, March 2017. doi:10.1017/9781316219317.

11 Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 2007. doi:10.1145/1182613.1182614.

12 Gus Gutoski. Properties of Local Quantum Operations with Shared Entanglement. *Quantum Information and Computation*, 9(9-10):0739–0764, May 2008. arXiv:0805.2209.

13 Gus Gutoski and John Watrous. Toward a general theory of quantum games. *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 565–574, 2007. doi:10.1145/1250790.1250873.

14 Timothée Hoffreumon and Ognyan Oreshkov. Projective characterization of higher-order quantum transformations. *arxiv.org*, 2022. arXiv:2206.06206.

15 Robin Houston. Finite products are biproducts in a compact closed category. *Journal of Pure and Applied Algebra*, 2008. doi:10.1016/j.jpaa.2007.05.021.

16 Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. In *Theoretical Computer Science*, volume 294, pages 183–231, 2003. doi:10.1016/S0304-3975(01)00241-9.

17 Aleks Kissinger, Matty Hoban, and Bob Coecke. Equivalence of relativistic causal structure and process terminality. *CoRR*, August 2017. arXiv:1708.04118.

18 Aleks Kissinger and Sander Uijlen. Picturing indefinite causal structure. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*, volume 236, pages 87–94, January 2017. doi:10.4204/EPTCS.236.6.

19 Aleks Kissinger and Sander Uijlen. A categorical semantics for causal structure. *Logical Methods in Computer Science*, 2019. doi:10.23638/LMCS-15(3:15)2019.

20 Paul-André Melliès. Categorical Semantics of Linear Logic, 2009. URL: https://www.irif.fr/~mellies/mpri/mpri-ens/biblio/categorical-semantics-of-linear-logic.pdf.

21 Lê Thành Dũng Nguyên and Lutz Straßburger. BV and Pomset Logic Are Not the Same. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 216, pages 1–32, 2022. doi:10.4230/LIPIcs.CSL.2022.32.

22 Ognyan Oreshkov, Fabio Costa, and Časlav Brukner. Quantum correlations with no causal order. *Nature Communications*, 3(1):1–8, October 2012. doi:10.1038/ncomms2076.

23 Sandu Popescu and Daniel Rohrlich. Quantum nonlocality as an axiom. *Foundations of Physics*, 24(3):379–385, March 1994. doi:10.1007/BF02058098.

24 Christian Retoré. Pomset logic: A non-commutative extension of classical linear logic. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1210, pages 300–318, 1997. doi:10.1007/3-540-62688-3_43.

25 P. Selinger. A survey of graphical languages for monoidal categories, 2011. doi:10.1007/978-3-642-12821-9_4.

26 Will Simmons and Aleks Kissinger. Higher-order causal theories are models of BV-logic. *arxiv.org*, 2022. arXiv:2205.11219.

27 Matt Wilson and Giulio Chiribella. Causality in Higher Order Process Theories. *Electronic Proceedings in Theoretical Computer Science*, 343:265–300, July 2021. doi:10.4204/eptcs.343.12.

Space-Bounded Unitary Quantum Computation with Postselection

Seiichiro Tani
NTT Communication Science Laboratories, NTT Corporation, Japan
International Research Frontiers Initiative (IRFI), Tokyo Institute of Technology, Japan

Abstract

Space-bounded computation has been a central topic in classical and quantum complexity theory. In the quantum case, every elementary gate must be unitary. This restriction makes it unclear whether the power of space-bounded computation changes by allowing intermediate measurement. In the bounded error case, Fefferman and Remscrim [STOC 2021, pp.1343–1356] and Girish, Raz and Zhan [ICALP 2021, pp.73:1–73:20] recently provided the break-through results that the power does not change. This paper shows that a similar result holds for space-bounded quantum computation with *postselection*. Namely, it is proved possible to eliminate intermediate postselections and measurements in the space-bounded quantum computation in the bounded-error setting. Our result strengthens the recent result by Le Gall, Nishimura and Yakaryilmaz [TQC 2021, pp.10:1–10:17] that logarithmic-space bounded-error quantum computation with *intermediate* postselections and measurements is equivalent in computational power to logarithmic-space unbounded-error probabilistic computation. As an application, it is shown that bounded-error space-bounded one-clean qubit computation (DQC1) with postselection is equivalent in computational power to unbounded-error space-bounded probabilistic computation, and the computational supremacy of the bounded-error space-bounded DQC1 is interpreted in complexity-theoretic terms.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases quantum complexity theory, space-bounded computation, postselection

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.81

Related Version *Full Version*: https://arxiv.org/abs/2206.15122

Funding This work was partially supported by JSPS KAKENHI Grant Number JP20H05966 and JP22H00522.

Acknowledgements I am grateful to anonymous referees of MFCS 2022 for their valuable comments.

1 Introduction

1.1 Background

Space-bounded computation is one of the most fundamental topics in complexity theory that have been studied in the classical and quantum settings, since it reflects common practical situations where available memory space is much less than input size. Watrous [25, 26] initiated the study of space-bounded quantum computation based on quantum Turing machines and proved that, in the unbounded-error setting, space-bounded quantum computation is equivalent in computational power to space-bounded probabilistic computation: $\mathsf{PrQSPACE}(s) = \mathsf{PrSPACE}(s)$ for any space-constructible s with $s(n) \in \Omega(\log n)$. This and the classical results [5, 13] imply that unbounded-error space-bounded quantum computation can be simulated by deterministic computation with the squared amount of the space used by the former model.

There are some subtleties (see [16] for the details) in considering space-bounded quantum computation. The most relevant one is whether we allow intermediate measurements, that is, the measurements made during computation, which are allowed in, e.g., [25, 26, 24]. In

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 81; pp. 81:1–81:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the case of polynomial-time quantum computation, it is well-known that all intermediate measurements can be deferred to the end of computation by coherently copying the state of the qubits to be measured to ancilla qubits, and keeping their contents unchanged through the computation. This may require a polynomial number of ancilla qubits to store the copies, since there may exist a polynomial number of intermediate measurements in the original computation. This is acceptable in the polynomial-time quantum computation. However, this method is not applicable in general in the case of space-bounded quantum computation. For instance, if we consider a logarithmic-space quantum computation that runs in polynomial time, the above transformation may require a polynomial number of ancilla qubits, much more than the available space. Thus, it is a fundamental question in space-bounded quantum computation whether it is possible to space-efficiently eliminate intermediate measurements. Recently, Fefferman and Remscrim [7] and Girish, Raz and Zhan [12] independently provided the breakthrough results that it is possible in the bounded error setting.

Postselection is a fictitious function that projects a quantum state on a single qubit to the prespecified state (say, $|1\rangle$) with certainty, as far as the former state has a non-zero overlap with the latter. Since Aaronson [1] introduced postselection to the quantum complexity theory field, it has turned out to be very effective concept in the field although it is unrealistic. In particular, Aaronson succeeded in characterizing a classical complexity class in a quantum way by introducing postselection: $\mathsf{PP} = \mathsf{PostBQP}$ [1]. This characterization gives one-line proofs of the classical results [4, 8], for which only involved classical proofs had been known, and has been a foundation for establishing the quantum computational supremacy of subuniversal quantum computation models (e.g., [6, 2, 17]) under complexity-theoretic assumptions. Another example of characterizing classical complexity classes with postselection is that PSPACE is equal to $\mathsf{PostQMA}$ [19], the class of languages that can be recognized by quantum Merlin-Arthur proof systems with polynomial-time quantum verifier with the ability of postselection. Along this line of work, Le Gall et al. [11] recently considered logarithmic-space quantum computation with postselection in the bounded-error setting and proved that its associated complexity class $\mathsf{PostBQL}$ is equivalent to PL, the class of languages that can be recognized with unbounded error by logarithmic-space probabilistic computation. This beautiful result can be regarded as the equivalent of $\mathsf{PP} = \mathsf{PostBQP}$ in the logarithmic-space quantum computation. Their model allows intermediate postselections as well as intermediate measurements, which play a key role for space-efficiency since the qubits on which intermediate postselections or measurements are made can be reused as initialized ancilla qubits for subsequent computation. Thus, a straightforward question is whether it is possible to space-efficiently eliminate intermediate postselections and measurements. Our main result answers this question affirmatively.

1.2 Our Contribution

We consider the space-bounded quantum computation that allows postselections and measurements only at the end of computation, which we call space-bounded *unitary* quantum computation with postselection. Our result informally says that such quantum computation is equivalent in computational power to the space-bounded quantum computation that allows intermediate postselections and measurements. More concretely, for a space-constructible function s with $s(n) \in \Omega(\log n)$, let $\mathsf{PostBQSPACE}(s)$ be the class of languages that can be recognized with bounded-error by quantum computation with (intermediate) postselections and measurements that uses $O(s)$ qubits and runs in $2^{O(s)}$ time, and let $\mathsf{PostBQuSPACE}(s)$ be the unitary version of $\mathsf{PostBQSPACE}(s)$. By the definition, it holds $\mathsf{PostBQuSPACE}(s) \subseteq \mathsf{PostBQSPACE}(s)$. We show the converse is also true.

▶ **Theorem 1.** *For any space-constructible function s with $s(n) \in \Omega(\log n)$, it holds that*

$$\mathsf{PostBQuSPACE}(s) = \mathsf{PostBQSPACE}(s) = \mathsf{PrSPACE}(s).$$

This strengthens the result of $\mathsf{PostBQSPACE}(s) = \mathsf{PrSPACE}(s)$, which can be derived straightforwardly from the proof of $\mathsf{PostBQL} = \mathsf{PL}$ in [11, 21]. A special case of Theorem 1 with $s = \log n$ is the following corollary, where we define $\mathsf{PostBQL} \equiv \mathsf{PostBQSPACE}(\log)$ and $\mathsf{PostBQuL} \equiv \mathsf{PostBQuSPACE}(\log)$.

▶ **Corollary 2.** $\mathsf{PostBQuL} = \mathsf{PostBQL} = \mathsf{PL}$.

Theorem 1 holds even when the completeness and soundness errors are $2^{-2^{O(s)}}$ (see Theorem 10 for a more precise statement). This justifies defining the bounded-error class $\mathsf{PostBQuSPACE}$ (note that it is non-trivial to reduce errors in the space-bounded unitary computation).

As an application, we characterize the power of space-bounded computation with postselection on a quantum model that is inherently unitary. The deterministic quantum computation with one quantum bit (DQC1)[15], often mentioned as the one-clean-qubit model, is one of well-studied quantum computation models with limited computational resources (e.g., [3, 23, 17, 9, 18, 10]). This model was originally motivated by nuclear magnetic resonance (NMR) quantum information processing, where it is difficult to initialize qubits to a pure state. In the DQC1 model, the initial state is thus the completely mixed state except for a single qubit, i.e., $|0\rangle\langle 0| \otimes (\mathrm{I}/2)^{\otimes m}$, if the total number of qubits is $m + 1$. This model is inherently unitary, since if intermediate measurements or postselections were allowed, the completely mixed state could be projected to the all-zero state $|0^m\rangle$ and thus the DQC1 model would become the ordinary quantum computation model, which is supplied with the all-zero state as the initial state.

Although DQC1 is considered very weak, a polynomial-size DQC1 circuit followed by postselection is surprisingly powerful: The corresponding bounded-error class $\mathsf{PostBQ_{[1]}P}$ is equal to $\mathsf{PostBQP}$ $(= \mathsf{PP})$ [17, 10]. Although this class is unrealistic, it plays an essential role in giving a strong evidence of computational supremacy of the DQC1 computation over classical computation: If any polynomial-size DQC1 circuit is classically simulatable in polynomial time, then the polynomial hierarchy (PH) collapses [17, 10].

Let us consider the space-bounded version of $\mathsf{PostBQ_{[1]}P}$. For a space-constructible function s with $s(n) \in \Omega(\log n)$, let $\mathsf{PostBQ_{[1]}SPACE}(s)$ be the class of languages that can be recognized with bounded-error by DQC1 computation with postselection that uses $O(s)$ qubits and runs in $2^{O(s)}$ time, where all postselections and measurements are made at the the end of computation.

▶ **Theorem 3.** *For any space-constructible function s with $s(n) \in \Omega(\log n)$, it holds that*

$$\mathsf{PostBQ_{[1]}SPACE}(s) = \mathsf{PostBQuSPACE}(s) = \mathsf{PrSPACE}(s).$$

In particular, $\mathsf{PostBQ_{[1]}L} = \mathsf{PostBQuL} = \mathsf{PL}$.

This result relates quantum computational supremacy of space-bounded DQC1 computation with complexity theory as in the time-bounded case. Namely, if any s-space DQC1 computation can be classically simulated with space bound s, then it must hold that $\mathsf{PrSPACE}(s) \subseteq \mathsf{PostBSPACE}(s)$ by Theorem 3, where $\mathsf{PostBSPACE}(s)$ is the classical counterpart of $\mathsf{PostBQuSPACE}(s)$. This relation is the space-bounded equivalent of $\mathsf{PP} \subseteq \mathsf{PostBPP}$. Note that $\mathsf{PP} \subseteq \mathsf{PostBPP}$ leads to the collapse of PH [6], since $\mathsf{PostBPP}$ is in the third level of PH. However, it is open whether $\mathsf{PrSPACE}(s) \subseteq \mathsf{PostBSPACE}(s)$ implies implausible consequences.

1.3 Technical Outline

Since space-bounded quantum computation with postselection can trivially simulate the unitary counterpart by the definition, our main technical contribution is to show that the unitary counterpart can simulate unbounded-error probabilistic computation. Our starting point is the simulation of unbounded-error probabilistic computation by the space-bounded quantum computation with postselection [11]. The simulation [11] consists of two components: the first one, Q_x, simulates the unbounded-error probabilistic computation with acceptance probability p_a on input x to output the state (up to a normalizing factor) $|\Psi_\delta\rangle = (1/2 + p_a)|0\rangle + \delta(1/2 - p_a)|1\rangle$ for a given positive parameter δ; the second one decides whether $p_a > 1/2$ or $p_a < 1/2$ with bounded error by repeatedly running Q_x to prepare the states $|\Psi_\delta\rangle$ for various values of δ and measuring them in the basis $\{|+\rangle, |-\rangle\}$ (based on a modification of the idea [1]). We eliminate the intermediate postselection in Q_x space-efficiently by, every time postselection is made in Q_x, incrementing a counter coherently if the postselection qubit is in the non-postselecting state. It is not difficult to see that this works since Q_x includes only intermediate postselections (and does not include intermediate measurements). This gives a unitary version V_x of Q_x. However, the second component includes both intermediate postselections and measurements, and needs run Q_x sequentially (because running Q_x in parallel is not space-efficient). We then construct two subroutines U_+ and U_-, which accumulate the amplitudes of $|+\rangle$ and $|-\rangle$, respectively, in the states obtained by repeatedly running V_x for various values of δ, as the amplitude of the all-zero state. Finally, we run U_+ and U_- in orthogonal spaces, respectively, followed by postselecting the all-zero state on the qubits that U_+ and U_- act on. The resulting state is significantly supported by one of the spaces, which determines whether $p_a > 1/2$ or $p_a < 1/2$ with high probability.

1.4 Organization

Sec. 2 introduces definitions, basic claims and known theorems. Sec. 3 provides the formal statement of our main result and proves it by using a lemma, which is proved in Sec. 4. Sec. 5 provides an application of the result.

2 Prelimiaries

Let $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ be the sets of natural numbers, integers, and real numbers, respectively. For $m \in \mathbb{N}$, let $[m]$ be the set of $\{1, \dots, m\}$. Assume that Σ is the set $\{0, 1\}$. A *promise problem* $L = (L_Y, L_N)$ is a pair of disjoint subsets of Σ^*. In the special case of promise problems such that $L_Y \cup L_N = \Sigma^*$, we say that L_Y is a *language*.

Classical Space-Bounded Computation

We say that a function $s : \mathbb{N} \to \mathbb{N}$ is *space-constructible* if there exists a deterministic Turing machine (DTM) that compute $s(|x|)$ in space $O(s(|x|))$ on input x. Suppose that s is a space constructible function. Then, we say that a function $f : \mathbb{N} \to \mathbb{R}$ is *s-space computable* if there exists a DTM that computes $f(|x|)$ in space $O(s(|x|))$ on input x. For $s \in \Omega(\log n)$, $\mathsf{PrSPACE}(s)$ is the class of promise problems L such that there exists a probabilistic Turing machine (PTM) M running with space $O(s)$ that satisfies the following: For every input $x \in L_Y$, the probability that M accepts x is greater than $1/2$, and for every input $x \in L_N$, the probability that M accepts x is at most $1/2$. We can replace the condition in the case of $x \in L_N$ with "the probability that M accepts x is *less* than $1/2$" without changing the class $\mathsf{PrSPACE}\,(s)$. In this paper, we adopt the latter definition. It is known that the class $\mathsf{PrSPACE}(s)$ does not change even if we impose the time bound $2^{O(s)}$ on the corresponding PTM with space bound s [13] (see also [22]). We define $\mathsf{PL} \equiv \mathsf{PrSPACE}\,(\log)$.

Quantum Circuits

We below introduce just notations and terminologies relevant to this paper. For the basics of quantum computing, see standard textbooks (e.g., [20, 14]). Let $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$.

A *quantum gate* implements a unitary operator. We say that a quantum gate is *elementary* if it acts on a constant number of qubits. We may use a quantum gate and its unitary operator that the gate implements interchangeably. Examples of elementary gates are $\mathrm{H} \equiv |+\rangle\langle 0|+|-\rangle\langle 1|$, $\mathrm{X} \equiv |0\rangle\langle 1|+|1\rangle\langle 0|$, $\mathrm{T} \equiv |0\rangle\langle 0|+e^{i\pi/4}|1\rangle\langle 1|$, and $\mathrm{CNOT} \equiv |0\rangle\langle 0|\otimes \mathrm{I}+|1\rangle\langle 1|\otimes \mathrm{X}$. For unitary gate g, $\wedge_k(g)$ denotes the unitary gate acting on $k+1$ qubits such that it applies g to the last qubit if the contents of the first k qubits are all 1, and it applies the identity otherwise. We may simply say that $\wedge_k(g)$ is a *k-qubit-controlled* g. In the case of $k=1$, we may use $\wedge(g)$ to denote $\wedge_1(g)$. For instance, $\wedge_2(\mathrm{X})$ is the Toffoli gate and $\wedge(\mathrm{X})$ is the CNOT gate. We also define $\vee_k(g)$ as the unitary gate acting on $k+1$ qubits such that it applies g to the last qubit if the contents of the first k qubits are not all-zero, and it applies the identity otherwise. Gate set G is defined as $G_1 \cup G_2 \cup G_3$ for $G_1 \equiv \{\mathrm{H}, \mathrm{T}, \mathrm{CNOT}\}$, the set G_2 of a constant number of additional elementary gates used for block encoding in [11], and the set G_3 of $\wedge(g)$ for all $g \in G_1 \cup G_2$. This choice of gates is not essential for our results when completeness and soundness errors are allowed to be $2^{-O(s)}$ for space bound s, because of space-efficient version of Solovay-Kitaev theorem (see [16, Theorem 4.3]), which says that it is possible to ϵ-approximate every unitary gate with a sequence of gates in any fixed gate set that is finite and universal in $O(\mathrm{polylog}\,(1/\epsilon))$ deterministic time and $O(\log\,(1/\epsilon))$ deterministic space.

For simple descriptions, we will also use k-qubit-controlled gates $\wedge_k\,(g)$ for $g \in G_1 \cup G_2$ and $k \geq 2$ in the following section, since $\wedge_k\,(g)$ can be implemented with gate set G together with $O(1)$ reusable ancilla qubits with negligible gate overhead:

▷ Claim 4. For every gate g, $\wedge_k(g)$ can be implemented with $\wedge(g)$, and $O(k)$ CNOT and T gates together with $O(1)$ ancilla qubits initialized to $|0\rangle$. Similarly, $\vee_k(g)$ can be implemented with $\wedge(g)$, and $O(k)$ CNOT, T and X gates together with $O(1)$ ancilla qubits initialized to $|0\rangle$. Moreover, the states of the ancilla qubits return to $|0\rangle$ after applying $\wedge_k(g)$ $(\vee_k(g))$.

Since the overhead of $O(k)$ gates can be ignored in our setting of $O(s)$ space and $2^{O(s)}$ time computation, we can effectively use $\wedge_k(g)$ freely. The proofs of Claim 4 and the following Claim 5 are provided in the full version.

Next, we define a special gate that will be used many time in this paper. Let INC_{2^n} be the unitary gate acting on n qubits that transforms $\mathrm{INC}_{2^n} : |j\rangle \mapsto |(j+1) \mod 2^n\rangle$ for all $j \in \{0, \ldots, 2^n - 1\}$. Intuitively, INC_{2^n} increments a counter over $\mathbb{Z}_{2^n}$.

▷ Claim 5. $\wedge_k(\mathrm{INC}_{2^n})$ can be implemented with $O(k+n^2)$ CNOT and T gates with the help of $O(1)$ ancilla qubits initialized to $|0\rangle$. Similarly, $\vee_k(\mathrm{INC}_{2^n})$ can be implemented with $O(k+n^2)$ CNOT, T, and X gates with the help of $O(1)$ ancilla qubits initialized to $|0\rangle$. The states of the ancilla qubits return to $|0\rangle$ after applying $\wedge_k(\mathrm{INC}_{2^n})$ $(\vee_k(\mathrm{INC}_{2^n}))$.

Since the overhead of $O(k+n^2)$ gates with $n, k \in O(s)$ can be ignored in our setting of $O(s)$ space and $2^{O(s)}$ time computation, we can effectively use $\wedge_k(\mathrm{INC}_{2^n})$ freely.

A *quantum circuit* consists of quantum gates in a fixed universal set of a constant number of unitary gates, and (intermediate) measurements. A *quantum circuit with postselection* is a quantum circuit with the ability of postselection even at intermediate points in the circuit. Here, the *postselection* [1] is a fictitious function that projects a quantum state on a single qubit to the state $|1\rangle$ with certainty, as far as there exists a non-zero overlap between the state and $|1\rangle$. For instance, if we make postselection on the first qubit of quantum state $\alpha\,|0\rangle\,|\psi_0\rangle + \beta\,|1\rangle\,|\psi_1\rangle$ with $\beta \neq 0$, resulting state is $|1\rangle\,|\psi_1\rangle$. Since the qubit on which postselection has been made is in the state $|1\rangle$ by the definition, we can reuse them as initialized qubits for subsequent computation. This can greatly save the space (i.e., the number of ancilla qubits) as in the case of intermediate measurements. We say that $|1\rangle$ is the postselecting state, and the state orthogonal to the postselecting state, $|0\rangle$, is the non-postselecting state. To simplify descriptions, we may say "postselect $|\phi\rangle$" to mean that we first apply a single-qubit unitary U such that $|1\rangle = U|\phi\rangle$ and then postselect $|1\rangle$. We may also say "postselect $|\phi_1\rangle \otimes \cdots \otimes |\phi_m\rangle$," where $|\phi_i\rangle$ is a single-qubit pure state for every i, to mean postselecting $|\phi_i\rangle$ on the ith qubit for each $i = 1, \ldots,\ m$. For instance, we may say "postselecting the all-zero state" (i.e., postselecting $|0^m\rangle$). A *unitary quantum circuit* consists of only (unitary) quantum gates and does not include any measurement or postselection. To perform computational tasks, a unitary quantum circuit will be followed by measurements (and postselections).

We say that, for a promise problem L, a family of quantum circuits $\{Q_x : x \in L\}$ with postselections (a family of unitary quantum circuits $\{U_x : x \in L\}$) is *s-space uniform* if there exists a DTM that, on input $x \in L$, outputs a description of Q_x (U_x, respectively) with the use of space $O(s(|x|))$ (and hence in $2^{O(s(|x|))}$ time).

Let $s\colon \mathbb{N} \to \mathbb{N}$ be a space-constructible function with $s(n) = \Omega(\log n)$. Assume that functions $c, d\colon \mathbb{N} \to [0,1]$ are s-space computable, and $c(n) > d(n)$ for sufficiently large $n \in \mathbb{N}$.

▶ **Definition 6** (PostQSPACE). *Let* PostQSPACE $(s)\,[c,d]$ *be the class of promise problems* $L = (L_Y, L_N)$ *for which there exists an s-space uniform family of quantum circuits with postselection,* $\{Q_x : x \in L\}$*, that act on* $m = O(s\,(|x|))$ *qubits and consist of* $2^{O(s(|x|))}$ *elementary gates such that, when applying* Q_x *to* $|0\rangle^{\otimes m}$*, (1) the probability* p_{post} *of measuring* $|1\rangle$ *on every postselection qubit is strictly positive; (2) for* $x \in L_Y$*, conditioned on all the postselection qubits being* $|1\rangle$*, the probability that* Q_x *accepts is at least* $c\,(|x|)$*; (3) for* $x \in L_N$*, conditioned on all the postselection qubits being* $|1\rangle$*, the probability that* Q_x *accepts is at most* $d\,(|x|)$*.*

▶ **Definition 7** (PostQuSPACE). *Let* PostQuSPACE$(s)[c.d]$ *be the class of promise problems* $L = (L_Y, L_N)$ *for which there exists an s-space uniform family of unitary quantum circuits,* $\{U_x\colon x \in L\}$*, that act on* $m \in O(s(|x|))$ *qubits and consist of* $2^{O(s(|x|))}$ *elementary gates, followed by postselection on the first qubit and measurement on the output qubit (say, the second qubit) in the computational basis, such that, when applying* U_x *to* $|0\rangle^{\otimes m}$*, (1) the probability* p_{post} *of measuring* $|1\rangle$ *on the first qubits is strictly positive; (2) for* $x \in L_Y$*, conditioned on the first qubit being* $|1\rangle$*, the probability that* U_x *accepts is at least* $c(|x|)$*; (3) for* $x \in L_N$*, conditioned on the first qubit being* $|1\rangle$*, the probability that* U_x *accepts is at most* $d(|x|)$*.*

Definition 7 assumes that postselection is made only on a single qubit. This is general enough since, if there are k postselection qubits, then we can aggregate them into a single postselection qubit by using $\wedge_k(\mathrm{X})$ and $O(1)$ ancilla qubits. Obviously, this aggregation does not change p_{post} and the acceptance probability.

Define $\mathsf{PostBQSPACE}(s) \equiv \mathsf{PostQSPACE}(s)[2/3, 1/3]$ and $\mathsf{PostBQL} \equiv \mathsf{PostBQSPACE}(\log)$. Similarly, define $\mathsf{PostBQuSPACE}(s) \equiv \mathsf{PostQuSPACE}(s)[2/3, 1/3]$ and $\mathsf{PostBQuL} \equiv \mathsf{PostBQuSPACE}(\log)$. Le Gall, Nishimura and Yakaryilmaz [11] proved the following.

▶ **Theorem 8** ([11]). $\mathsf{PostBQL} = \mathsf{PL}$.

Moreover, it is straightforward to extend the result to the general space bound.

▶ **Theorem 9** ([11]). *For any space-constructible function* $s\colon \mathbb{N} \to \mathbb{N}$ *with* $s(n) \in \Omega(\log n)$*, it holds that* $\mathsf{PrSPACE}(s) = \mathsf{PostQSPACE}(s)[1 - 2^{-2^{O(s)}}, 2^{-2^{O(s)}}] = \mathsf{PostBQSPACE}(s)$.

In [11], $\mathsf{PostBQL}$ is defined based on the space-bounded quantum Turing machine (QTM), following the definition provided in [26]. It is not difficult to see that their proof works for the circuit-based definition. Consequently, the QTM-based definition and the circuit-based definition are equivalent in computational power.

3 Main Results

Theorem 10 provides a formal statement of our main result, which shows that the class of promise problems that can be solved with an s-space uniform family of *unitary* quantum circuits with postselection by using $O(s)$ space and $2^{O(s)}$ time in the *bounded-error* setting is equal to the class of promise problems by *unbounded-error* probabilistic computation with space bound s.

▶ **Theorem 10.** *For any space-constructible function* $s : \mathbb{N} \to \mathbb{N}$ *with* $s(n) = \Omega(\log n)$*,*

$$\mathsf{PrSPACE}(s) = \mathsf{PostQuSPACE}(s)\left[1 - 2^{-2^{O(s)}}, 2^{-2^{O(s)}}\right] = \mathsf{PostBQuSPACE}(s).$$

In particular, $\mathsf{PostBQuL} = \mathsf{PL}$.

Theorems 9 and 10 imply that intermediate postselections and measurements add no extra computational power, as stated formally in the following corollary.

▶ **Corollary 11** (Restatement of Theorem 1). *For any space-constructible function* $s : \mathbb{N} \to \mathbb{N}$ *with* $s(n) = \Omega(\log n)$*,*

$$\mathsf{PostBQuSPACE}(s) = \mathsf{PostBQSPACE}(s) = \mathsf{PrSPACE}(s).$$

In particular, $\mathsf{PostBQL} = \mathsf{PostBQuL}$.

Proof of Theorem 10. By the definition, we have $\mathsf{PostQuSPACE}(s)\left[1 - 2^{-2^{O(s)}}, 2^{-2^{O(s)}}\right] \subseteq \mathsf{PostBQuSPACE}(s)$. Then, the theorem follows from Lemmas 12 and 13, stated as follows. ◀

▶ **Lemma 12.** *For any space-constructible function* $s : \mathbb{N} \to \mathbb{N}$ *with* $s(n) = \Omega(\log n)$*,*

$$\mathsf{PostBQuSPACE}(s) \subseteq \mathsf{PrSPACE}(s).$$

Proof. By the definition, we have $\mathsf{PostBQuSPACE}(s) \subseteq \mathsf{PostBQSPACE}(s)$. Since $\mathsf{PostBQSPACE}(s) = \mathsf{PrSPACE}(s)$ by Theorem 9, the lemma follows. ◀

▶ **Lemma 13.** *For any space-constructible function* $s : \mathbb{N} \to \mathbb{N}$ *with* $s(n) = \Omega(\log n)$*,*

$$\mathsf{PrSPACE}(s) \subseteq \mathsf{PostQuSPACE}(s)\left[1 - 2^{-2^{O(s)}}, 2^{-2^{O(s)}}\right].$$

The proof is provided in the following section.

4 Proof of Lemma 13

To prove Lemma 13, we use the fact proved in [11].

▶ **Lemma 14** ([11])**.** *Suppose that, for any input x, a PTM with space bound $s = s\left(|x|\right)$ accepts with probability $p_a = p_a(x)$ and rejects with probability $1 - p_a$ after running in a prespecified time $T(|x|) \in 2^{O(s)}$. There exists an s-space uniform family of quantum circuits Q_x on $m + \lceil \log_2\left(T+1\right)\rceil$ qubits for $m \in O(s)$ that consist of $2^{O(s)}$ elementary gates in the gate set G and intermediate postselections such that, for every $k \in \{0, \dots, T\}$, it holds that*

$$Q_x|0^m\rangle|k\rangle = \frac{|\Psi_k\rangle|0^{m-1}\rangle|k\rangle}{\||\Psi_k\rangle|0^{m-1}\rangle|k\rangle\|},$$

where $|\Psi_k\rangle \equiv \left(1/2 + p_a\right)|0\rangle + 2^{T-k}\left(1/2 - p_a\right)|1\rangle$.

Let L be a promise problem in $\mathsf{PrSPACE}(s)$. Then, there exists a PTM M that recognizes L with unbounded error in space $O(s\left(|x|\right))$ on input x. By the result by Jung [13] (see also [22]), we assume without loss of generality that M runs in time $T(|x|) \in 2^{O(s(|x|))}$. Since the computation path is split into two paths in each step with equal probability, the accepting probability p_a is of the form $a/2^T$ for some integer $a \in \{0, \cdots, 2^T\} \setminus 2^{T-1}$ (assuming $p_a \neq 1/2$ without loss of generality). There exist an s-space uniform family of quantum circuits Q_x defined in Lemma 14. Note that Q_x makes intermediate postselections and thus Q_x is not unitary. Since we have assumed $p_a \neq 1/2$, we can decide whether p_a is larger or smaller than $1/2$ with unbounded error by measuring $|\Psi_T\rangle$ in the basis $\{|+\rangle, |-\rangle\}$. To distinguish the two cases with bounded error, we need to reduce error probability. For this, Le Gall et al. [11] uses essentially the same idea as is used in [1], repeating the following operations for every k: prepare $|\Psi_k\rangle$ and measure it in the basis $\{|+\rangle, |-\rangle\}$. Since the qubits on which measurements or postselections have been made can be reused by initializing them using block encoding with postselection, the space requirement is bounded by $O\left(s\right)$. Thus, intermediate postselections and measurements play a key role in space efficiency.

4.1 Base Unitary Circuit V_x

Our goal is to move every postselection and measurement down to the end of computation while increasing the space requirement by at most a constant factor.

This is not difficult for the Q_x part. The following modification can make Q_x unitary: We prepare an $N \in O(s)$ bit counter C initialized to the all-zero state $\left|0^N\right\rangle$ in a quantum register C. Here, we take a sufficiently large integer in $O(s)$ as N. Then, every time postselection is made in Q_x, we instead increment the counter C coherently if the postselection qubit is in the non-postselecting state, and perform the other operations (i.e., unitary gates) in the same way as in the original circuit Q_x. If we assume without loss of generality that non-postselecting state is $|0\rangle$, then the counter C is incremented by applying the X gate to the postselection qubit, applying $\wedge(\mathrm{INC}_{2^N})$ gate controlled by that qubit, and then applying the X gate to that qubit, namely, $(\mathrm{X} \otimes \mathrm{I})\left(\wedge\left(\mathrm{INC}_{2^N}\right)\right)(\mathrm{X} \otimes \mathrm{I})$. Let V_x denote the modified circuit.

By the above construction and the standard analysis (e.g., [25]), if we measure the counter C and postselect the all-zero state after applying V_x to $|0^m\rangle|k\rangle$, the output state is $|\Psi_k\rangle|0^{m-1}\rangle|k\rangle$ up to a normalizing factor. More concretely, every time postselection is made in Q_x, the modified circuit V_x moves the non-postselecting state into the space associated with the counter value being non-zero, that is, the space orthogonal to the space where the postselecting state lies. By setting N so that the maximum counter value $2^N - 1$ is larger than the number of postselections in Q_x, it holds that, once the counter C is incremented, the counter value never returns to zero. Thus, the quantum interference between the states associated with the counter-values being zero and non-zero never occurs. This implies that, if the content of counter register C is zero, the state must be projected to the postselecting state in *every* postselection points in Q_x, and thus the entire register except C is in the state that is equal to $Q_x|0^m\rangle|k\rangle$. Thus, for certain normalized states $|bad_k(j)\rangle$ and $\ell = m + O(1) \in O(s)$, we can write

$$V_x|0^\ell\rangle_{\mathsf{R}}|0^N\rangle_{\mathsf{C}}|k\rangle_{\mathsf{K}} = \left[\gamma_k|\Psi_k\rangle_{\mathsf{R1}}|0^{\ell-1}\rangle_{\mathsf{R2}}|0^N\rangle_{\mathsf{C}} + \sqrt{1-\|\gamma_k|\Psi_k\rangle\|^2}\sum_{j\geq 1}|bad_k(j)\rangle_{\mathsf{R}}|j\rangle_{\mathsf{C}}\right]|k\rangle_{\mathsf{K}}, \tag{1}$$

where $0 < \gamma_k < 1$, and

$$|\Psi_k\rangle \equiv (1/2 + p_a)\,|0\rangle + 2^{T-k}\,(1/2 - p_a)\,|1\rangle = \alpha_k|+\rangle + \beta_k|-\rangle, \tag{2}$$

for $\alpha_k = \langle +|\Psi_k\rangle$ and $\beta_k = \langle -|\Psi_k\rangle$. Here, register K consists of $\lceil \log_2(T+1)\rceil$ qubits and stores the argument $k \in \{0,\dots,T\}$. The first register $\mathsf{R} \equiv (\mathsf{R}_1, \mathsf{R}_2)$ is the working register except registers C and K, where R_1 is the subregister of R corresponding to the first qubit and R_2 consists of the remaining qubits. Note that V_x uses the register C in addition to registers (R, K) of $O(s)$ qubits, and applies $\wedge(\mathrm{INC}_{2^N})$ instead of every intermediate postselection made in Q_x. Since Q_x consists of $2^{O(s)}$ unitary gates and intermediate postselections on $O(s)$ qubits, and since Claim 5 implies that $\wedge(\mathrm{INC}_{2^N})$ is implementable with $O(N^2)$ $(= O(s))$ CNOT and T gates with $O(1)$ ancilla qubits, which are reusable for other $\wedge(\mathrm{INC}_{2^N})$, it holds that V_x consists of $2^{O(s)}$ gates in G and acts on $O(s)$ qubits.

4.2 Subroutine Unitary Circuits U_+ and U_-

In the following subsections, we will describe the entire algorithm, which includes the error-reduction step. For this, we first provide two unitary subroutines U_+ and U_- in Figure 1. They use V_x and act on registers $(\mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$, where D is an $O(s)$-qubit quantum register used as another $O(s)$-bit counter D. We assume without loss of generality that the two registers C and D consist of N qubits for sufficiently large $N \in O(s)$. The following lemmas tell us about the actions of U_+ and U_-.

▶ **Lemma 15.** *For every $k \in \{0,\dots,T\}$, let α_k and γ_k be the coefficients appearing in Eqs. (1) and (2). Then, U_+ given in Figure 1 acts on $O(s)$ qubits, consists of $2^{O(s)}$ gates in the gate set G, and satisfies*

$$U_+|0\rangle_{\mathsf{R}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}} = \left[\gamma|\alpha_T|^2\cdots|\alpha_0|^2|0\rangle_{\mathsf{R,C}}|0\rangle_{\mathsf{D}} + \sum_{j\geq 1}|\phi_T(j)\rangle_{\mathsf{R,C}}|j\rangle_{\mathsf{D}}\right]|0\rangle_{\mathsf{K}}$$

for certain unnormalized quantum states $|\phi_T(j)\rangle$ on registers (R, C) for each $j \geq 1$, where $\gamma = |\gamma_T|^2 \cdots |\gamma_0|^2$. Moreover, for every $r \in \mathbb{N} \cap 2^{O(s)}$, $(U_+)^r$ acts on $O(s)$ qubits, consists of $2^{O(s)}$ gates in the gate set G, and satisfies

$$(U_+)^r|0\rangle_{\mathsf{R}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}} = \left[\left(\gamma|\alpha_T|^2\cdots|\alpha_0|^2\right)^r|0\rangle_{\mathsf{R,C}}|0\rangle_{\mathsf{D}} + \sum_{j\geq 1}|\phi_T^{(r)}(j)\rangle_{\mathsf{R,C}}|j\rangle_{\mathsf{D}}\right]|0\rangle_{\mathsf{K}},$$

where $|\phi_T^{(r)}(j)\rangle$ is a certain unnormalized quantum state on registers (R, C) for each $j \geq 1$.

Subroutine U_+ associated with V_x

Repeat the following steps $T+1$ times.

1. Perform V_x on $(\mathsf{R},\mathsf{C},\mathsf{K})$.
2. If the content of C is non-zero or the first qubit in R is in state $|-\rangle$, then apply INC_{2^N} to register D to increment the counter D.
3. If the content of D is zero, then invert the Step1 on $(\mathsf{R},\mathsf{C},\mathsf{K})$, i.e, apply $V_x^\dagger$.
4. If the content of (R,C) is not all-zero, then apply INC_{2^N} to register D to increment the counter D.
5. Apply INC_{T+1} to register K to increment the content in register K.

Apply $\mathrm{INC}_{T+1}^\dagger$ $T+1$ times to initialize register K to the all-zero state.

Subroutine U_- associated with V_x

Same as U_+ except that Step 2 is replaced with the following operation.

2. If the content of C is non-zero or the first qubit in R is in state $|+\rangle$, then apply INC_{2^N} to register D to increment the counter D.

Figure 1 Subroutines U_+ and U_- associated with V_x.

Proof. We first give the analysis of the first repetition on registers $(\mathsf{R},\mathsf{C},\mathsf{D},\mathsf{K})$ initialized to the all-zero state. Step 1 applies V_x to $|0^\ell\rangle_{\mathsf{R}}|0^N\rangle_{\mathsf{C}}|k\rangle_{\mathsf{K}}$ with $k=0$. By Eq (1), the resulting state in (R,C) is

$$\gamma_k(\alpha_k|+\rangle+\beta_k|-\rangle)_{\mathsf{R1}}|0^{\ell-1}\rangle_{\mathsf{R2}}|0\rangle_{\mathsf{C}}+\sqrt{1-\|\gamma_k|\Psi_k\rangle\|^2}\sum_{j\geq 1}|bad_k(j)\rangle_{\mathsf{R}}|j\rangle_{\mathsf{C}},$$

where we omit the register K for simplicity. Then, Step 2 appends register D and increments the counter D if the content of the register C is not zero or the register R_1 is in the state $|-\rangle$. Thus, the resulting state is

$$\begin{aligned}&\mapsto \gamma_k\alpha_k|+\rangle_{\mathsf{R1}}|0^{\ell-1}\rangle_{\mathsf{R2}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}\\&\quad+\left(\gamma_k\beta_k|-\rangle_{\mathsf{R1}}|0^{\ell-1}\rangle_{\mathsf{R2}}|0\rangle_{\mathsf{C}}+\sqrt{1-\|\gamma_k|\Psi_k\rangle\|^2}\sum_{j\geq 1}|bad_k(j)\rangle_{\mathsf{R}}|j\rangle_{\mathsf{C}}\right)|1\rangle_{\mathsf{D}}\\&\qquad\qquad\qquad=\gamma_k\alpha_k|+\rangle_{\mathsf{R1}}|0^{\ell-1}\rangle_{\mathsf{R2}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}+|\phi\rangle_{\mathsf{R,C}}|1\rangle_{\mathsf{D}},\end{aligned}$$

where $|\phi\rangle_{\mathsf{R,C}}=\gamma_k\beta_k|-\rangle_{\mathsf{R1}}|0^{\ell-1}\rangle_{\mathsf{R2}}|0\rangle_{\mathsf{C}}+\sqrt{1-\|\gamma_k|\Psi_k\rangle\|^2}\sum_{j\geq 1}|bad_k(j)\rangle_{\mathsf{R}}|j\rangle_{\mathsf{C}}$. Step 3 then inverts the Step1 on $(\mathsf{R},\mathsf{C},\mathsf{K})$, i.e, applies $V_x^\dagger$, if the content of D is zero. Since $\langle+|_{\mathsf{R1}}\langle 0^{\ell-1}|_{\mathsf{R2}}\langle 0|_{\mathsf{C}}\langle k|_{\mathsf{K}}V_x|0^\ell\rangle_{\mathsf{R}}|0^N\rangle_{\mathsf{C}}|k\rangle_{\mathsf{K}}=\gamma_k\alpha_k$, the resulting state is

$$\mapsto \gamma_k\alpha_k\left((\gamma_k\alpha_k)^*|0\rangle_{\mathsf{R}}|0\rangle_{\mathsf{C}}+\sqrt{1-|\gamma_k\alpha_k|^2}|0^\perp\rangle_{\mathsf{R,C}}\right)|0\rangle_{\mathsf{D}}+|\phi\rangle_{\mathsf{R,C}}|1\rangle_{\mathsf{D}},$$

where $|0^\perp\rangle$ is a certain state orthogonal to the all-zero state. Step 4 then increments the counter D if the content of (R,C) is not all-zero; we have

$$\mapsto |\gamma_k\alpha_k|^2|0\rangle_{\mathsf{R}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}+\sum_{j=1}^{2}|\phi_k(j)\rangle_{\mathsf{R,C}}|j\rangle_{\mathsf{D}},$$

for certain states $|\phi_k(j)\rangle$ for $j = 1, 2$. Step 5 increments the content in register K to get the state on $(\mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$:

$$\left[|\gamma_k \alpha_k|^2 \, |0\rangle_{\mathsf{R}} \, |0\rangle_{\mathsf{C}} \, |0\rangle_{\mathsf{D}} + \sum_{j=1}^{2} |\phi_k(j)\rangle_{\mathsf{R},\mathsf{C}} \, |j\rangle_{\mathsf{D}} \right] |k+1\rangle_{\mathsf{K}} \, .$$

Then, we repeat the same procedure. A simple induction on k shows that the final state after applying U_+ to $|0^\ell\rangle_{\mathsf{R}}|0^N\rangle_{\mathsf{C}}|0^N\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}}$ is

$$U_+|0^\ell\rangle_{\mathsf{R}}|0^N\rangle_{\mathsf{C}}|0^N\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}} = \left[\gamma|\alpha_T|^2 \cdots |\alpha_0|^2 |0\rangle_{\mathsf{R},\mathsf{C}}|0\rangle_{\mathsf{D}} + \sum_{j\geq 1} |\phi_T(j)\rangle_{\mathsf{R},\mathsf{C}}|j\rangle_{\mathsf{D}} \right] |0\rangle_{\mathsf{K}},$$

where $\gamma = |\gamma_T|^2 \cdots |\gamma_0|^2$. Thus, if we repeat U_+ r times, the resulting state is

$$(U_+)^r|0^\ell\rangle_{\mathsf{R}}|0^N\rangle_{\mathsf{C}}|0^N\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}} = \left[\left(\gamma|\alpha_T|^2 \cdots |\alpha_0|^2\right)^r |0\rangle_{\mathsf{R},\mathsf{C}}|0\rangle_{\mathsf{D}} + \sum_{j\geq 1} |\phi_T^{(r)}(j)\rangle_{\mathsf{R},\mathsf{C}}|j\rangle_{\mathsf{D}} \right] |0\rangle_{\mathsf{K}},$$

for some unnormalized states $|\phi_T^{(r)}(j)\rangle$.

Next, we consider the space and gate complexities of U_+ (see the full version for a rigorous analysis). Recall that V_x can be implemented with $2^{O(s)}$ gates in G by using $O(1)$ reusable ancilla qubits. One can show by using Claims 4 and 5 that every other step in U_+ can also be implemented with $2^{O(s)}$ gates in G by using $O(1)$ reusable ancilla qubits. Consequently, U_+ can be implemented with $2^{O(s)}$ gates in G and requires $O(1)$ ancilla qubits in addition to $(\mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$, which is $O(s)$ qubits in total. Since the ancilla qubits are reusable, $(U_+)^r$ is also implementable with $r \cdot 2^{O(s)} = 2^{O(s)}$ gates in G and acts on $O(s)$ qubits. ◀

We can prove the following lemma for U_- in almost the same way.

▶ **Lemma 16.** *For every $k \in \{0, \ldots, T\}$, let β_k and γ_k be the coefficients appearing in Eqs. (1) and (2). Then, U_- given in Figure 1 acts on $O(s)$ qubits, consists of $2^{O(s)}$ gates in the gate set G, and satisfies*

$$U_-|0\rangle_{\mathsf{R}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}} = \left[\gamma|\beta_T|^2 \cdots |\beta_0|^2 |0\rangle_{\mathsf{R},\mathsf{C}}|0\rangle_{\mathsf{D}} + \sum_{j\geq 1} |\psi_T(j)\rangle_{\mathsf{R},\mathsf{C}}|j\rangle_{\mathsf{D}} \right] |0\rangle_{\mathsf{K}}$$

for certain unnormalized quantum states $|\psi_T(j)\rangle$ on registers (R, C) for each $j \geq 1$, where $\gamma = |\gamma_T|^2 \cdots |\gamma_0|^2$. Moreover, for every $r \in \mathbb{N} \cap 2^{O(s)}$, $(U_-)^r$ acts on $O(s)$ qubits, consists of $2^{O(s)}$ gates in the gate set G, and satisfies

$$(U_-)^r|0\rangle_{\mathsf{R}}|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{D}}|0\rangle_{\mathsf{K}} = \left[\left(\gamma|\beta_T|^2 \cdots |\beta_0|^2\right)^r |0\rangle_{\mathsf{R},\mathsf{C}}|0\rangle_{\mathsf{D}} + \sum_{j\geq 1} |\psi_T^{(r)}(j)\rangle_{\mathsf{R},\mathsf{C}}|j\rangle_{\mathsf{D}} \right] |0\rangle_{\mathsf{K}},$$

where $|\psi_T^{(r)}(j)\rangle$ is a certain unnormalized quantum state on registers (R, C) for each $j \geq 1$.

4.3 Final Unitary Circuit

Figure 2 shows a unitary quantum circuit with postselection acting on five registers $(\mathsf{W}, \mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$, where W is a single-qubit register. Now, we finalize the proof of Lemma 13 with this circuit.

Unitary Quantum Circuit with Postselection for $\mathsf{PrSPACE}(s)$ problems

Initialize registers $(\mathsf{W}, \mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$ to the all-zero state.

1. Apply the Hadamard gate H to register W.
2. If the content of W is 0, then apply $(U^{+})^{r}$ to registers $(\mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$; otherwise, apply $(U^{-})^{r}$ to registers $(\mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$.
3. Postselect the all-zero state on register D.
4. Measure register W in the basis $\{|0\rangle, |1\rangle\}$. If the outcome 0, then accept ($p_a > 1/2$); otherwise reject (i.e., $p_a < 1/2$).

Figure 2 Unitary Quantum Circuit with Postselection for $\mathsf{PrSPACE}(s)$ problems.

For simplicity, we first assume that $r = 1$. It is straightforward to extend the proof to the general r. It follows from Lemmas 15 and 16 that after Step 2, the state in the register $(\mathsf{W}, \mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K}))$ is

$$\frac{1}{\sqrt{2}}|0\rangle_{\mathsf{W}}\left[\gamma|\alpha_T|^2\cdots|\alpha_0|^2|0\rangle_{\mathsf{R,C}}|0\rangle_{\mathsf{D}}+\sum_{j\geq 1}|\phi_T(j)\rangle_{\mathsf{R,C}}|j\rangle_{\mathsf{D}}\right]|0\rangle_{\mathsf{K}}$$
$$+\frac{1}{\sqrt{2}}|1\rangle_{\mathsf{W}}\left[\gamma|\beta_T|^2\cdots|\beta_0|^2|0\rangle_{\mathsf{R,C}}|0\rangle_{\mathsf{D}}+\sum_{j\geq 1}|\psi_T(j)\rangle_{\mathsf{R,C}}|j\rangle_{\mathsf{D}}\right]|0\rangle_{\mathsf{K}}. \quad (3)$$

Step 3 postselects the all-zero state in register D. Thus, the resulting state in the register $(\mathsf{W}, \mathsf{R}, \mathsf{C})$ is

$$\gamma'\left(|\alpha_T|^2\cdots|\alpha_0|^2|0\rangle_{\mathsf{W}}+|\beta_T|^2\cdots|\beta_0|^2|1\rangle_{\mathsf{W}}\right)|0\rangle_{\mathsf{R,C}} \quad (4)$$

where γ' is the normalizing factor. Here, the probability of measuring postselecting state is

$$p_{post}=\frac{1}{2}\left(\left(\gamma|\alpha_T|^2\cdots|\alpha_0|^2\right)^2+\left(\gamma|\beta_T|^2\cdots|\beta_0|^2\right)^2\right).$$

Recall that $\alpha_k = \langle +|\Psi_k\rangle$ and $\beta_k = \langle -|\Psi_k\rangle$, where $|\Psi_k\rangle$ is defined as Eq. (2). If $p_a > 1/2$, then $\beta_k > 0$ for all k. If $p_a < 1/2$, then $\alpha_k > 0$ for all k. Since $\gamma \neq 0$, p_{post} is strictly positive in both cases.

If $p_a < 1/2$, then $|\alpha_k|^2 > |\beta_k|^2 \geq 0$ for all k, and $|\alpha_k|^2 > (1+\delta)|\beta_k|^2$ for some k and some constant δ, say, $16/9$. Since $\frac{|\beta_T|^2\cdots|\beta_0|^2}{|\alpha_T|^2\cdots|\alpha_0|^2} < \frac{1}{1+\delta} = \frac{9}{25}$, the probability that $|0\rangle_W$ is measured in Step 4, that is, the probability of obtaining the outcome 0 when measuring register W in Eq. (4) in the basis $\{|0\rangle, |1\rangle\}$, is

$$\frac{|\alpha_T|^4\cdots|\alpha_0|^4}{|\alpha_T|^4\cdots|\alpha_0|^4+|\beta_T|^4\cdots|\beta_0|^4}=\frac{1}{1+\left(|\beta_T|^4\cdots|\beta_0|^4\right)/\left(|\alpha_T|^4\cdots|\alpha_0|^4\right)}>\frac{1}{1+(9/25)^2}=\frac{625}{706}.$$

If $p_a > 1/2$, then $0 \leq |\alpha_k|^2 < |\beta_k|^2$ for all k, and $(1+\delta)|\alpha_k|^2 < |\beta_k|^2$ for some k and a constant $\delta = 16/9$. Thus, the probability that $|1\rangle_{\mathsf{W}}$ is measured in Step 4 is at least $\frac{625}{706}$ in the same analysis as in the case of $p_a < 1/2$.

For a general $r \in 2^{O(s)}$, the state in the register $(\mathsf{W}, \mathsf{R}, \mathsf{C})$ after Step 3 is

$$\gamma''\left(\left(|\alpha_T|^2\cdots|\alpha_0|^2\right)^r|0\rangle_{\mathsf{W}}+\left(|\beta_T|^2\cdots|\beta_0|^2\right)^r|1\rangle_{\mathsf{W}}\right)|0\rangle_{\mathsf{R,C}},$$

where γ'' is the normalizing factor. For $p_a < 1/2$, thus, the probability that $|0\rangle$ is measured in Step 4 is

$$\frac{\left(|\alpha_T|^4 \cdots |\alpha_0|^4\right)^r}{\left(|\alpha_T|^4 \cdots |\alpha_0|^4\right)^r + \left(|\beta_T|^4 \cdots |\beta_0|^4\right)^r} > \frac{1}{1+(9/25)^{2r}} > 1 - \frac{81^r}{625^r + 81^r} > 1 - \frac{1}{2^r}.$$

Similarly, for $p_a > 1/2$, the probability that $|1\rangle$ is measured in Step 4 is at least $1 - \frac{1}{2^r}$.

Finally, we consider the space and gate complexities. The quantum circuit in Figure 2 acts on a single-qubit register W in addition to $(\mathsf{R}, \mathsf{C}, \mathsf{D}, \mathsf{K})$. In this circuit, every gate $g \in G$ used in $(U_+)^r$ and $(U_-)^r$ is replaced with $\wedge(g)$, which can be implemented with $O(1)$ gates in G with $O(1)$ resuable ancilla qubits by Claim 4. Since Step 2 is dominant, and $(U_+)^r$ and $(U_-)^r$ use $O(s)$ qubits and $2^{O(s)}$ gates in G by Lemmas 15 and 16, the entire circuit uses $O(s)$ qubits and $2^{O(s)}$ gates in G.

5 Application to One-Clean Qubit Model

As introduced in Section 1, DQC1 is a model of quantum computing such that the input state is completely mixed except for one qubit, which is initialized to $|0\rangle$.

▶ **Definition 17** ($\mathsf{PostQ_{[1]}SPACE}$)**.** *Let s be any space-constructible function with $s(n) \in \Omega(\log n)$, and let c, d be s-space computable functions.* $\mathsf{PostQ_{[1]}SPACE}(s)[c,d]$ *is the class of promised problems $L = (L_Y, L_N)$ for which there exists an s-space uniform family of unitary quantum circuits $\{U_x : x \in L\}$ consisting of $2^{O(s)}$ elementary gates on $m+1$ qubits for $m \in O(s)$ such that, when applying U_x to the $m+1$ qubits in state $|0\rangle\langle 0| \otimes (\mathrm{I}/2)^{\otimes m}$, followed by postselections and measurements, (1) the probability p_{post} of measuring $|1\rangle$ on all postselection qubits is strictly positive; (2) for $x \in L_Y$, conditioned on all the postselection qubits being $|1\rangle$, the probability that U_x accepts is at least $c(|x|)$; (3) for $x \in L_N$, conditioned on all the postselection qubits being $|1\rangle$, the probability that U_x accepts is at most $d(|x|)$. In particular, define* $\mathsf{PostBQ_{[1]}SPACE}(s) \equiv \mathsf{PostQ_{[1]}SPACE}(s)[2/3, 1/3]$*, and* $\mathsf{PostBQ_{[1]}L} \equiv \mathsf{PostBQ_{[1]}SPACE}(\log)$*.*

In the above definition, we allow postselection to be made on multiple qubits, since it does not seem possible in general to aggregate multiple postselection qubits to a single qubit due to the lack of initialized qubits. Theorem 3 follows from Theorems 10 and 18.

▶ **Theorem 18.** *For any space-constructible function s with $s(n) \in \Omega(\log n)$ and any s-space computable functions c and d such that $c(n) > d(n)$ for sufficiently large $n \in \mathbb{N}$,*

$$\mathsf{PostQ_{[1]}SPACE}(s)[c,d] = \mathsf{PostQuSPACE}(s)[c,d].$$

In particular, $\mathsf{PostBQ_{[1]}SPACE}(s) = \mathsf{PostBQuSPACE}(s)$ *and* $\mathsf{PostBQ_{[1]}L} = \mathsf{PostBQuL}$*.*

The proof is provided in the full version.

References

1 Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A*, 461(2063):3473–3482, 2005. doi:10.1098/rspa.2005.1546.

2 Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. *Theory of Computing*, 9:143–252, 2013. doi:10.4086/toc.2013.v009a004.

3 Andris Ambainis, Leonard J. Schulman, and Umesh V. Vazirani. Computing with highly mixed states. *Journal of the ACM*, 53(3):507–531, 2006. doi:10.1145/1147954.1147962.

4 R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995. doi:10.1006/jcss.1995.1017.

5 A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1):113–136, 1983. doi:10.1016/S0019-9958(83)80060-6.

6 Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society A*, 467:459–472, 2010. doi:10.1098/rspa.2010.0301.

7 Bill Fefferman and Zachary Remscrim. Eliminating intermediate measurements in space-bounded quantum computation. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 1343–1356, 2021. doi:10.1145/3406325.3451051.

8 Lance Fortnow and Nick Reingold. PP is closed under truth-table reductions. *Inf. Comput.*, 124(1):1–6, 1996. doi:10.1006/inco.1996.0001.

9 Keisuke Fujii, Hirotada Kobayashi, Tomoyuki Morimae, Harumichi Nishimura, Shuhei Tamate, and Seiichiro Tani. Power of quantum computation with few clean qubits. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.13.

10 Keisuke Fujii, Hirotada Kobayashi, Tomoyuki Morimae, Harumichi Nishimura, Shuhei Tamate, and Seiichiro Tani. Impossibility of classically simulating one-clean-qubit model with multiplicative error. *Phys. Rev. Lett.*, 120:200502, 2018. doi:10.1103/PhysRevLett.120.200502.

11 François Le Gall, Harumichi Nishimura, and Abuzer Yakaryilmaz. Quantum logarithmic space and post-selection. In *Proceedings of the 16th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2021*, volume 197 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.TQC.2021.10.

12 Uma Girish, Ran Raz, and Wei Zhan. Quantum logspace algorithm for powering matrices with bounded norm. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 73:1–73:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.73.

13 Hermann Jung. On probabilistic time and space. In *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, pages 310–317. Springer-Verlag, 1985. doi:10.1007/BFb0015756.

14 Alexei Yu. Kitaev, Alexander H. Shen, and Mikhail N. Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate Studies in Mathematics.* AMS, 2002.

15 E. Knill and R. Laflamme. Power of one bit of quantum information. *Phys. Rev. Lett.*, 81:5672–5675, December 1998. doi:10.1103/PhysRevLett.81.5672.

16 Dieter van Melkebeek and Thomas Watson. Time-space efficient simulations of quantum computations. *Theory of Computing*, 8(1):1–51, 2012. doi:10.4086/toc.2012.v008a001.

17 Tomoyuki Morimae, Keisuke Fujii, and Joseph F. Fitzsimons. Hardness of classically simulating the one-clean-qubit model. *Phys. Rev. Lett.*, 112:130502, 2014. doi:10.1103/PhysRevLett.112.130502.

18 Tomoyuki Morimae, Keisuke Fujii, and Harumichi Nishimura. Power of one nonclean qubit. *Phys. Rev. A*, 95:042336, 2017. doi:10.1103/PhysRevA.95.042336.

19 Tomoyuki Morimae and Harumichi Nishimura. Merlinization of complexity classes above BQP. *Quantum Info. Comput.*, 17(11–12):959–972, 2017. doi:10.26421/QIC17.11-12-3.

20 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000. doi:10.1017/CBO9780511976667.

21 Harumichi Nishimura. Personal communication, 2021.

22 M. Saks. Randomization and derandomization in space-bounded computation. In *Proceedings of Computational Complexity (Formerly Structure in Complexity Theory)*, pages 128–149, 1996. doi:10.1109/CCC.1996.507676.

23 Peter W. Shor and Stephen P. Jordan. Estimating Jones polynomials is a complete problem for one clean qubit. *Quantum Information & Computation*, 8(8):681–714, 2008. doi:10.26421/QIC8.8-9-1.

24 Amnon Ta-Shma. Inverting well conditioned matrices in quantum logspace. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, STOC '13*, pages 881–890, 2013. doi:10.1145/2488608.2488720.

25 John Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *Journal of Computer and System Sciences*, 62(2):376–391, 2001. doi:10.1006/jcss.2000.1732.

26 John Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12:48–84, 2003. doi:10.1007/s00037-003-0177-8.

Computing the Minimum Bottleneck Moving Spanning Tree

Haitao Wang ✉
Department of Computer Science, Utah State University, Logan, UT, USA

Yiming Zhao[1] ✉
Department of Computer Science, Utah State University, Logan, UT, USA

Abstract

Given a set P of n points that are moving in the plane, we consider the problem of computing a spanning tree for these moving points that does not change its combinatorial structure during the point movement. The objective is to minimize the bottleneck weight of the spanning tree (i.e., the largest Euclidean length of all edges) during the whole movement. The problem was solved in $O(n^2)$ time previously [Akitaya, Biniaz, Bose, De Carufel, Maheshwari, Silveira, and Smid, WADS 2021]. In this paper, we present a new algorithm of $O(n^{4/3}\log^3 n)$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases minimum spanning tree, moving points, unit-disk range emptiness query, dynamic data structure

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.82

Related Version *Full Version*: `https://arxiv.org/abs/2206.12500`

Funding This research was supported in part by NSF under Grant CCF-2005323.

1 Introduction

Given a set P of n points in the plane, let G_P be the complete graph whose vertex set is P such that the weight of each edge connecting two points p and q of P is the Euclidean distance between p and q. The *Euclidean minimum spanning tree (EMST)* of P is the spanning tree of G_P with minimum sum of edge weights. The *Euclidean minimum bottleneck spanning tree (EMBST)* of P is the spanning tree of G_P whose largest edge weight is minimized. It is well known that a Delaunay triangulation of P contains an EMST of P [24] and thus an EMST of P can be computed in $O(n \log n)$ time by constructing a Delaunay triangulation of P first. This is also the case for the bottleneck problem.

In this paper, motivated by visualizations of time-varying spatial data [2], we consider a moving version of the EMBST problem where every point of P is moving during a time interval. Without loss of generality, we assume that the time interval is $[0, 1]$. A *moving point* $p \in P$ is a continuous function $p : [0, 1] \to \mathbb{R}^2$. Let $p(t)$ denote the location of p at time $t \in [0, 1]$. We assume that p moves on a straight line segment with a constant velocity, i.e., $p(t)$ is linear in t and points of $\{p(t)|\ t \in [0, 1]\}$ form a straight line segment in the plane (see Fig. 1; different points may have different velocities). A *moving spanning tree* T of P connects all points of P and does not change its connection during the whole time interval (i.e., for any two points $p, q \in P$, the path connecting p and q in T always contains the same set of edges). We use $T(t)$ to denote the tree at the time t. The *instantaneous bottleneck* $b_T(t)$ at time t is the maximum length of all edges in $T(t)$. The *bottleneck* $b(T)$ of

[1] Corresponding author.

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 82; pp. 82:1–82:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

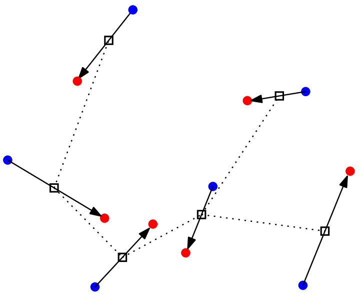


Figure 1 Each pair of red and blue points connected by an arrow represents a moving point. Blue points denote locations at $t = 0$ and red points are locations at $t = 1$. Black boxes are locations of these moving points at certain time and the dashed segments form a spanning tree.

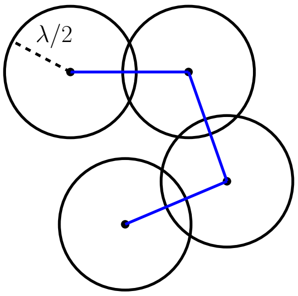


Figure 2 Illustrating a unit-disk graph. Two points are connected (by a blue segment) if their distance is less than or equal to λ. In other words, two points are connected if congruent disks centered at them with radius $\lambda/2$ intersect.

the moving spanning tree T is defined to be the maximum instantaneous bottleneck during the whole time interval, i.e., $b(T) = \max_{t\in[0,1]} b_T(t)$. The *Euclidean minimum bottleneck moving spanning tree* (or *moving-EMBST* for short) T^* refers to the moving spanning tree of P with minimum bottleneck.

In this paper, we study the problem of computing the moving-EMBST T^* for a set P of n moving points in the plane as defined above. Previously, this problem was solved in $O(n^2)$ time by Akitaya, Biniaz, Bose, De Carufel, Maheshwari, Silveira, and Smid [2]. To solve the problem, the authors of [2] first proved the following *key property*: The function of the distance between two moving points over time is convex (this is because each point moves linearly with constant velocity), implying that the maximum distance between two moving points is achieved at $t = 0$ or $t = 1$ (note that this does not mean T^* is attained at either $t = 0$ or $t = 1$; a counterexample is provided in [2]). Using the above property, the authors of [2] proposed the following simple algorithm to compute T^*. First, compute a complete graph G with P as the vertex set such that the weight of each edge connecting two points p and q of P is defined as the maximum length of their distances at $t = 0$ and at $t = 1$. Then the authors [2] showed that a minimum bottleneck spanning tree (MBST) of G is also a moving-EMBST of P and thus it suffices to compute an MBST in G. Since an MBST of a graph can be computed in linear time in the graph size [7], the entire algorithm for computing T^* runs in $O(n^2)$ time in total [2].

1.1 Our result

We present an algorithm of $O(n^{4/3}\log^3 n)$ time to compute T^*. We sketch the main idea below.

For any two points p and q in the plane, let $|pq|$ denote their Euclidean distance. Due to the above key property from [2], we observe that $b(T^*)$ must be equal to $|pq|_{\max}$ for two moving points p and q of P, where $|pq|_{\max} = \max\{|p(0)q(0)|, |p(1)q(1)|\}$, i.e., $b(T^*) \in \{|pq|_{\max} \mid p, q \in P\}$. As such, our main idea is to find $b(T^*)$ in $\{|pq|_{\max} \mid p, q \in P\}$ by binary search. To this end, we first solve a *decision problem*: Given any value $\lambda > 0$, decide whether $b(T^*) \leq \lambda$. We reduce the decision problem to the problem of finding a common spanning tree in two unit-disk graphs. Specifically, the *unit-disk graph* $G_\lambda(Q)$ for a set Q of points in the plane with respect to a parameter λ is an undirected graph whose vertex set is Q such that an edge connects two points $p, q \in Q$ if $|pq| \leq \lambda$ (alternatively, $G_\lambda(Q)$ can be viewed as the intersection graph of the set of congruous disks centered at the points of Q with radius $\lambda/2$, i.e., two vertices are connected if their disks intersect; see Fig. 2). Observe that $b(T^*) \leq \lambda$ if and only if the unit-disk graph $G_\lambda(P)$ for P at time $t = 0$ and the unit-disk graph $G_\lambda(P)$ for P at time $t = 1$ share a common spanning tree. To determine whether the two unit-disk graphs share a common spanning tree, we apply breadth-first-search (BFS) on the two graphs simultaneously. To avoid quadratic time, we do not compute these unit-disk graphs explicitly. Instead, we use a batched range searching technique of Katz and Sharir [19] to obtain a compact representation for searching one graph. For searching the other graph, we derive a semi-dynamic data structure for the following *deletion-only unit-disk range emptiness query* problem: Preprocess a set Q of n points in the plane with respect to λ so that the following two operations can be performed efficiently: (1) given a query point p, determine whether Q has a point q such that $|pq| \leq \lambda$, and if yes, return such a point q; (2) delete a point from Q. We refer to the first operation as *unit-disk range emptiness query* (or UDRE query for short). We build a data structure of $O(n)$ space in $O(n \log n)$ time such that each UDRE query can be answered in $O(\log n)$ time while each deletion can be performed in $O(\log n)$ amortized time. This result might be interesting in its own right. Combining this result with the batched range searching [19], we implement the BFS simultaneously on the two unit-disk graphs in $O(n^{4/3} \log^2 n)$ time, which solves the decision problem.

Next, equipped with the above decision algorithm, we find $b(T^*)$ from the set $\{|pq|_{\max} \mid p, q \in P\}$ by binary search. Computing the set explicitly would take $\Omega(n^2)$ time. We avoid doing so by resorting to the distance selection algorithm of Katz and Sharir [19], which can compute the k-th smallest distance among all interpoint distances of a set of n points in the plane in $O(n^{4/3} \log^2 n)$ time for any k with $1 \leq k \leq \binom{n}{2}$. Combining with our decision algorithm, $b(T^*)$ can be computed in $O(n^{4/3} \log^3 n)$ time. Applying the value $\lambda = b(T^*)$ to the decision algorithm can produce the optimal spanning tree T^* in additional $O(n^{4/3} \log^2 n)$ time.

1.2 Related work

Similar to the moving-EMBST problem, one can consider the Euclidean minimum moving spanning tree (moving-EMST) for a set of moving points (i.e., minimizing the total sum of the edge weights instead). The authors of [2] proved that the moving-EMST problem is NP-hard and they gave an $O(n^2)$ time 2-approximation algorithm and another $O(n \log n)$ time $(2+\epsilon)$-approximation algorithm for any $\epsilon > 0$. These spanning tree problems for moving points are relevant in the realm of moving networks that is motivated by the increase in mobile data consumption and the network architecture containing mobile nodes [2].

Geometric problems for moving objects have been studied extensively in the literature, e.g., [3, 4]. In particular, kinetic data structures were proposed to maintain the minimum spanning tree for moving points in the plane [3, 25]. Different from our problem, research in this domain focuses on bounds of the number of combinatorial changes in the minimum spanning tree during the point movement [4].

For solving the deletion-only UDRE query problem, by the standard lifting transformation, one can reduce the problem to maintaining the lower envelope of a dynamic set of planes in $\mathbb{R}^3$, which has been extensively studied [1, 9, 15, 18]. Applying Chan's recent work [11] for the problem can achieve the following result: With $O(n \log n)$ preprocessing time, each UDRE query can be answered in $O(\log^2 n)$ time and each point deletion can be handled in $O(\log^4 n)$ amortized time (the data structure is actually fully-dynamic and can also handle each point insertion in $O(\log^2 n)$ amortized time). The same problem in 2D (whose dual problem becomes maintaining the convex hull for a dynamic set of points) is easier and has also been studied extensively, e.g., [5, 8, 17, 23]. In addition, Wang [26] studied the unit-disk range counting query problem for a static set of points in the plane, by extending the techniques for half-plane range counting query problem [10, 20, 21].

Our algorithm for the decision problem uses some techniques for unit-disk graphs. Many problems on unit-disk graphs have been studied, i.e., shortest paths and reverse shortest paths [6, 12, 13, 27–30], clique [14], independent set [22], diameter [12, 13, 16], etc. Although a unit-disk graph of n vertices may have $\Omega(n^2)$ edges, many problems can be solved in subquadratic time by exploiting its underlying geometric structures, e.g., computing shortest paths [6, 27]. Our $O(n^{4/3}\log^2 n)$ time algorithm for finding a common spanning tree in two unit-disk graphs adds one more problem to this category.

Outline. In the following, we present our algorithm for the moving-EMBST problem in Section 2. The algorithm uses our data structure for the deletion-only unit-disk range emptiness query problem, which is given in Section 3. Section 4 concludes. Due to the space limit, some proofs are omitted but can be found in the full paper.

2 Algorithm for moving-EMBST

We follow the notation in Section 1, e.g., P, t, $b(T)$, $b_T(t)$, T^*, $|pq|$, $|pq|_{\max}$, $G_\lambda(P)$, etc. Given a set P of n points in the plane, our goal is to compute $b(T^*)$. As discussed in Section 1.1, we first consider the decision problem: Given any $\lambda > 0$, decide whether $b(T^*) \leq \lambda$. We refer to the original problem for computing $b(T^*)$ as the *optimization problem*. In what follows, we solve the decision problem in Section 2.1 and the algorithm for the optimization problem is described in Section 2.2.

2.1 The decision problem

Given any $\lambda > 0$, the decision problem is to decide whether $b(T^*) \leq \lambda$.

For any time $t \in [0, 1]$, we use $P(t)$ to denote the set of points of P at their locations at time t, i.e., $P(t) = \{p(t) \mid p \in P\}$. Consider the two unit-disk graphs $G_\lambda(P(0))$ and $G_\lambda(P(1))$. To simplify the notation, we use $G_\lambda(t)$ to refer to $G_\lambda(P(t))$ for any $t \in [0, 1]$. For every point $p \in P$, we consider $p(0)$ in $G_\lambda(0)$ and $p(1)$ in $G_\lambda(1)$ as the same vertex p, and thus define $G_\lambda = G_\lambda(0) \cap G_\lambda(1)$ as the *intersection graph* of $G_\lambda(0)$ and $G_\lambda(1)$, i.e., the vertex set of G_λ is P and G_λ has an edge connecting two vertices p and q if and only $G_\lambda(0)$ has an edge connecting $p(0)$ and $q(0)$ and $G_\lambda(1)$ has an edge connecting $p(1)$ and $q(1)$. A spanning tree of G_λ is called a *common spanning tree* of $G_\lambda(0)$ and $G_\lambda(1)$.

The following observation has been proved in [2].

▶ **Observation 1** ([2]). *$\max_{t\in[0,1]} |p(t)q(t)| = \max\{|p(0)q(0)|, |p(1)q(1)|\}$ holds for every pair of points $p, q \in P$.*

Using the above observation, the following lemma reduces the decision problem to the problem of finding a common spanning tree of $G_\lambda(0)$ and $G_\lambda(1)$. The proof can be found in the full paper.

▶ **Lemma 2.** *Given any $\lambda > 0$, $b(T^*) \leq \lambda$ if and only if $G_\lambda(0)$ and $G_\lambda(1)$ have a common spanning tree.*

In light of Lemma 2, to solve the decision problem, it suffices to determine whether $G_\lambda(0)$ and $G_\lambda(1)$ have a common spanning tree, or alternatively, whether the intersection graph G_λ has a spanning tree, which is true if and only if the graph is connected. To determine whether G_λ is connected, we perform a breadth-first search (BFS) in G_λ, or equivalently, we perform a BFS on $G_\lambda(0)$ and $G_\lambda(1)$ simultaneously; we do so without computing the two unit-disk graphs explicitly to avoid the quadratic time. Our algorithm relies on the following lemma for the deletion-only UDRE query problem, which will be proved in Section 3.

▶ **Theorem 3.** *Given a value λ and a set Q of n points in the plane, we can build a data structure of $O(n)$ space in $O(n \log n)$ time such that the following first operation can be performed in $O(\log n)$ worst case time while the second operation can be performed in $O(\log n)$ amortized time.*

1. Unit-disk range emptiness (UDRE) query*: Given a point p, determine whether there exists a point $q \in Q$ such that $|pq| \leq \lambda$, and if yes, return such a point q.*
2. Deletion*: delete a point from Q.*

In the following, we begin with an algorithm overview and then flesh out the details.

Algorithm overview. Starting from an arbitrary point $s \in P$, we run BFS in the graph G_λ. For each $i = 0, 1, 2, \ldots$, let P_i be the set of points whose shortest path lengths from s in G_λ are equal to i. In each i-th iteration, the algorithm computes P_i. Initially, $P_0 = \{s\}$. The algorithm stops once we have $P_i = \emptyset$, after which we check whether all points of P have been discovered. If yes, then the BFS tree is a spanning tree of G_λ; otherwise, G_λ is not connected. Consider the i-th iteration. Suppose P_{i-1} is already known. For each point $p \in P_{i-1}$, we wish to find the set $S(p)$ of all points $q \in P$ such that (1) q has not been discovered yet, i.e., $q \notin \bigcup_{j=0}^{i-1} P_j$; (2) $|p(0)q(0)| \leq \lambda$; (3) $|p(1)q(1)| \leq \lambda$. To implement this step efficiently, we use two techniques. First, we use a batched range searching technique of Katz and Sharir [19] to obtain a compact representation of all points of $P(0)$. The compact representation can provide us with a collection $\mathcal{N}(p)$ of canonical subsets of P whose union is exactly the subset of points q of P such that $|p(0)q(0)| \leq \lambda$. Second, for each subset Q of $\mathcal{N}(p)$, a data structure of Theorem 3 is constructed for $Q(1) = \{q(1) \mid q \in Q\}$, i.e., the set of points of Q at their locations at time $t = 1$. Then, we apply the UDRE query with $p(1)$ as the query point; if the query returns a point $q(1)$, then we know that q is in $S(p)$ and we delete q from Q (we also delete q from other canonical subsets of the compact representation that contain q; the deletion guarantees that points of P already discovered by the BFS have been removed from the canonical subsets of the compact representation) and applying the UDRE query with $p(1)$ again. We keep doing this until the UDRE query does not return any point, and then we process the next subset of $\mathcal{N}(p)$ in the same way. In this way, $S(p)$ will be computed, which is a subset of P_i. Processing every point $p \in P_{i-1}$ as above will produce P_i. The details of the algorithm are given below.

Preprocessing. Before running BFS, we conduct some preprocessing work.

First, using a batched range searching technique [19], we have the following lemma (which is essentially Theorem 3.3 in [19]) for computing a *compact representation* of all pairs (p, q) of points of P with $|p(0)q(0)| \leq \lambda$.

▶ **Lemma 4** (Theorem 3.3 [19]). *We can compute a collection $\{X_r \times Y_r\}_r$ of complete edge-disjoint bipartite graphs in $O(n^{4/3} \log n)$ time and space, where $X_r, Y_r \subseteq P$, with the following properties.*

1. *For any r, $|p(0)q(0)| \leq \lambda$ holds for any point $p \in X_r$ and any point $q \in Y_r$.*
2. *The number of these complete edge-disjoint bipartite graphs is $O(n^{4/3})$, and both $\sum_r |X_r|$ and $\sum_r |Y_r|$ are bounded by $O(n^{4/3} \log n)$.*
3. *For any two points $p, q \in P$ with $|p(0)q(0)| \leq \lambda$, there exists a unique r such that $p \in X_r$ and $q \in Y_r$.*

We refer to each X_r (resp., Y_r) as a *canonical subset* of P. After the collection $\{X_r \times Y_r\}_r$ is computed, we further do the following. For each point $p \in P$, if p is in X_r, then we add (the index of) Y_r to $\mathcal{N}(p)$. By Lemma 4(3), subsets of $\mathcal{N}(p)$ are pairwise disjoint and the union of them is exactly the subset of points $q \in P$ with $|p(0)q(0)| \leq \lambda$. Similarly, for each point $p \in P$, if p is in Y_r, then we add (the index of) Y_r to $\mathcal{M}(p)$. The purpose of having $\mathcal{M}(p)$ is that after a point p is identified in P_i, we will need to remove p from all subsets Y_r that contain p (so $\mathcal{M}(p)$ helps us to keep track of these subsets Y_r). We can compute $\mathcal{N}(p)$ and $\mathcal{M}(p)$ for all points $p \in P$ in $O(n^{4/3}\log n)$ time since both $\sum_r |X_r|$ and $\sum_r |Y_r|$ are $O(n^{4/3}\log n)$ by Lemma 4(2). For the same reason, both $\sum_{p\in P} |\mathcal{N}(p)|$ and $\sum_{p\in P} |\mathcal{M}(p)|$ are bounded by $O(n^{4/3}\log n)$.

In addition, for each canonical subset Y_r, we construct the data structure of Theorem 3 for $Y_r(1) = \{q(1) \mid q \in Y_r\}$, denoted by $\mathcal{D}(Y_r)$. Since $\sum_r |Y_r| = O(n^{4/3}\log n)$, constructing the data structures for all Y_r can be done in $O(n^{4/3}\log^2 n)$ time and $O(n^{4/3}\log n)$ space.

This finishes our preprocessing work, which takes $O(n^{4/3}\log^2 n)$ time in total.

Implementing the BFS algorithm. We next implement the BFS algorithm as overviewed above (we follow the same notation).

For each point $p \in P_{i-1}$, the key step is to compute the subset $S(p)$ of P. We implement this step as follows. For each $Y_r \in \mathcal{N}(p)$, we perform a UDRE query with $p(1)$ on the data structure $\mathcal{D}(Y_r)$. If the query returns a point $q(1)$, then we add q to $S(p)$ and delete $q(1)$ from the data structure $\mathcal{D}(Y'_r)$ for every $Y'_r \in \mathcal{M}(q)$. Next, we perform a UDRE query with $p(1)$ on $\mathcal{D}(Y_r)$ again and repeat the same process as above until the query does not return any point. According to the definitions of $\mathcal{N}(p)$ and $\mathcal{M}(p)$ and also due to the deletions on $\mathcal{D}(Y'_r)$ for all $Y'_r \in \mathcal{M}(q)$, the union of $S(p)$ thus computed for all $p \in P_{i-1}$ is exactly P_i. This finishes the i-th iteration of the BFS algorithm.

For the time analysis, since both $\sum_{p\in P} |\mathcal{N}(p)|$ and $\sum_{p\in P} \mathcal{M}(p)$ are $O(n^{4/3}\log n)$, the total number of UDRE queries and deletions on the data structures $\mathcal{D}(Y_r)$ in the entire algorithm is $O(n^{4/3}\log n)$, which together take $O(n^{4/3}\log^2 n)$ time. Therefore, the BFS algorithm runs in $O(n^{4/3}\log^2 n)$ time.

The following theorem summarizes our result for the decision problem.

▶ **Theorem 5.** *Given a value $\lambda > 0$, we can decide whether $b(T^*) \leq \lambda$ in $O(n^{4/3}\log^2 n)$ time, and if yes, a moving spanning tree T of P with $b(T) \leq \lambda$ can be found in $O(n^{4/3}\log^2 n)$ time.*

2.2 The optimization problem

As discussed in Section 1, by Observation 1, $b(T^*)$ is equal to $|p(0)q(0)|$ or $|p(1)q(1)|$ for two moving points $p, q \in P$. As such, we can compute $b(T^*)$ by searching the two sets $S(0)$ and $S(1)$ using our decision algorithm in Theorem 5, where $S(t)$ is defined as $\{|p(t)q(t)| \mid p, q \in P\}$ for any $t \in [0,1]$. To avoid explicitly computing $S(0)$ and $S(1)$, which would take $\Omega(n^2)$ time, we resort to the distance selection algorithm of Katz and Sharir [19], which can compute the k-th smallest distance among all interpoint distances of a set of n points in the plane in $O(n^{4/3}\log^2 n)$ time for any k with $1 \leq k \leq \binom{n}{2}$. Combining the distance selection algorithm and our decision algorithm, we can compute $b(T^*)$ in $O(n^{4/3}\log^3 n)$ time by doing binary search on the values of $S(0)$ and $S(1)$. The details are given in the proof of the following theorem, which can be found in the full paper.

▶ **Theorem 6.** *Given a set P of n moving points in the plane, we can compute a Euclidean minimum bottleneck moving spanning tree for them in $O(n^{4/3}\log^3 n)$ time.*

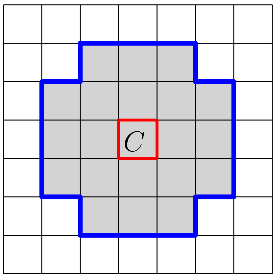


Figure 3 The cells in the gray region bounded by the blue curve are all neighbors of the red cell.

3 Deletion-only unit-disk range emptiness query data structure

In this section, we prove Theorem 3. We follow the notation in the theorem, e.g., Q, λ.

We use a *unit-disk* to refer to a disk with radius λ. For any point p in the plane, we use A_p to denote the unit-disk centered at p. With this notation, a unit-disk range emptiness (UDRE) query with query point p becomes the following: Determine whether $A_p \cap Q$ is empty, and if not, return a point from $A_p \cap Q$.

We use a grid Ψ_λ to capture the neighboring information of the points of Q, which partitions the plane into square cells of side length $\lambda/\sqrt{2}$ by horizontal and vertical lines, so that the distance of any two points in each cell is at most λ. For ease of discussion, we assume that each point of Q is in the interior of a cell of Ψ_λ. Define $Q(C)$ as the subset of points of Q lying in a cell C. A cell C' of Ψ_λ is a *neighbor* of another cell C if the minimum distance between a point of C and a point of C' is at most λ (see Fig. 3). For each cell C, we use $N(C)$ to denote the set of neighbors of C in Ψ_λ; for convenience, we let $N(C)$ include C itself. Note that the number of neighbors of each cell of Ψ_λ is $O(1)$ and each cell is a neighbor of $O(1)$ cells (since $C' \in N(C)$ if and only if $C \in N(C')$). Let $\mathcal{C}$ denote the set of cells of Ψ_λ that contain at least one point of Q as well as their neighbors. Note that $\mathcal{C}$ has $O(n)$ cells. By the definition of $\mathcal{C}$, the following observation is self-evident.

▶ **Observation 7.** *For any point p in the plane, if p is not in any cell of $\mathcal{C}$, then $A_p \cap Q = \emptyset$.*

The grid technique was widely used in algorithms for unit-disk graphs [12,27,29,30]. The following lemma has been proved in [26].

▶ **Lemma 8** ([26]).

1. *The set $\mathcal{C}$, along with the subsets $Q(C)$ and $N(C)$ for all cells $C \in \mathcal{C}$, can be computed in $O(n \log n)$ time and $O(n)$ space.*
2. *With $O(n \log n)$ time and $O(n)$ space preprocessing, given any point p in the plane, we can do the following in $O(\log n)$ time: Determine whether p is in a cell C of $\mathcal{C}$, and if yes, return C and the set $N(C)$.*

Note that we do not compute the entire grid Ψ_λ but only compute the information in Lemma 8. We next prove Theorem 3 using the information computed in Lemma 8.

Consider a UDRE query with a query point p. By Lemma 8(2), we can determine whether p is in a cell $C \in \mathcal{C}$. If not, by Observation 7, we are done with the query. Below we assume that p is in a cell $C \in \mathcal{C}$. In this case, $A_p \cap Q \neq \emptyset$ if and only if $A_p \cap Q(C') \neq \emptyset$ for a cell $C' \in N(C)$. As such, as $|N(C)| = O(1)$, it suffices to check for each cell $C' \in N(C)$, whether $A_p \cap Q(C') = \emptyset$. In this way, we reduce our original problem for Q to $Q(C')$. As such, below we construct a data structure $\mathcal{D}_C(C')$ for $Q(C')$ with respect to C. Note that we also need to handle deletions for $Q(C')$. Depending on whether $C' = C$, there are two cases.

If $C' = C$, then all points of $Q(C')$ are in the disk A_p and thus we can return an arbitrary point of $Q(C')$ as the answer to the UDRE query. To support the deletions on $Q(C')$, we build a balanced binary search tree $T(C')$ for all points of $Q(C')$ sorted by their indices (we can arbitrarily assign indices to points of Q) as our data structure $\mathcal{D}_C(C')$. In this way, deleting a point from $\mathcal{D}_C(C')$ can be done in $O(\log n)$ time. Therefore, in the case where $C' = C$, we can perform each UDRE query and each deletion in $O(\log n)$ time.

In what follows, we assume that $C' \neq C$, which is our main focus. In this case, C' and C are separated by an axis-parallel line. Without loss of generality, we assume that they are separated by a horizontal line ℓ such that C' is above ℓ and C is below ℓ. We further assume that ℓ contains the upper edge of C. The rest of this section is organized as follows. In Section 3.1, we first present some observations which our approach is based on. We describe our preprocessing algorithm for $Q(C')$ in Section 3.2 while handling the UDRE queries and deletions is discussed in Section 3.3. Section 3.4 finally summarizes everything. In the following, we let $m = |Q(C')|$.

3.1 Observations

Our basic idea is to maintain the portion $\mathcal{U}$ inside C of the lower envelope of the unit-disks centered at points of $Q(C')$. Then, $A_p \cap Q(C') \neq \emptyset$ if and only if p is above $\mathcal{U}$. Determining whether p is above $\mathcal{U}$ can be easily done by binary search because $\mathcal{U}$ is x-monotone. To handle deletions, we borrow an idea from Hershberger and Suri [17] for maintaining the convex hull of a semi-dynamic (deletion-only) set of points in the plane. To make our approach work, we first present some observations in this subsection.

Recall that A_q denotes a unit-disk centered at point q. We use ∂A_q to denote the boundary of A_q, which is a unit-circle. Let $\xi_q = \partial A_q \cap C$, i.e., the portion of the circle ∂A_q inside C. Note that it is possible that $\xi_q = \emptyset$, in which case either $A_q \cap C = \emptyset$ or $C \subseteq A_q$. If $A_q \cap C = \emptyset$, then $|pq| > \lambda$ holds for all points $p \in C$ and thus q can be ignored from constructing our data structure $\mathcal{D}_C(C')$. If $C \subseteq A_q$, then $|pq| \leq \lambda$ always holds for all points $p \in C$ and thus we can process all such points q in the same way as the above case $C' = C$. As such, in the following we assume that $\xi_q \neq \emptyset$ for every point $q \in Q(C')$. Because the radius of A_q is λ while the side-length of C is $\lambda/\sqrt{2}$, ξ_q consists of at most two arcs of ∂A_q. Further, ξ_q has exactly two arcs only if ∂A_q intersects the lower edge of C. For simplicity of discussion, we remove the lower edge from C and make C a bottom-unbounded rectangle (i.e., C's upper edge does not change, its two vertical edges extend downwards to the infinity, and its lower edge is removed); so now C has three edges. In this way, $\xi_q = \partial A_q \cap C$ is always a single arc.

Since q is above the horizontal line ℓ, which contains the upper edge of C, ξ_q must be x-monotone. This means the lower envelope $\mathcal{U}$ of $\Xi = \{\xi_q \mid q \in Q(C')\}$ is also x-monotone (see Fig. 4). We will show that $\mathcal{U}$ can be computed in linear time by a Graham's scan style algorithm once the arcs of Ξ are ordered in a certain way. To define this special order, we first introduce some notation below.

Recall that the boundary ∂C consists of three edges. Let l^* denote the lower endpoint of the left edge of C at $-\infty$; similarly, let r^* denote the lower endpoint of the right edge of C (see Fig. 4). For any two points a and b on ∂C, we say that a is *left of* b if a is counterclockwise from b around C (i.e., if we traverse from l^* to r^* along ∂C, a will be encountered earlier than b). For each arc ξ_q, if a and b are its two endpoints and a is left of b (see Fig. 4), then we call a the *left endpoint* of ξ_q and b the *right endpoint.* For ease of exposition, we make a general position assumption that no two arcs of Ξ share a common endpoint. The special order mentioned above for the Graham's scan style algorithm is the order of arcs of Ξ by their right endpoints on ∂C, called *right-endpoint left-to-right order.* To justify the correctness, we prove some properties for the lower envelope $\mathcal{U}$ below.

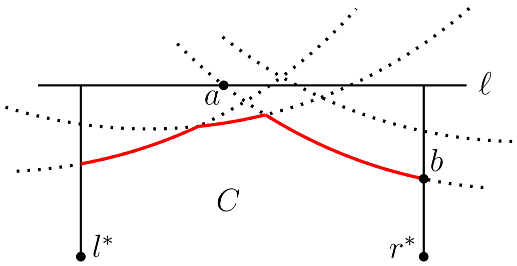


Figure 4 Illustrating the lower envelope (the red curve).

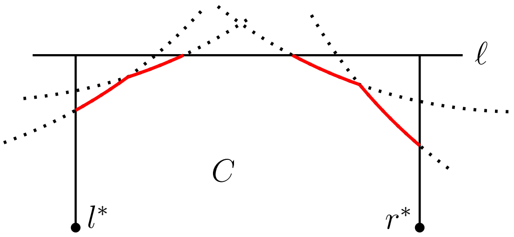


Figure 5 Illustrating a lower envelope (the red curve) that has two connected components.

Suppose we traverse on ∂C from l^* until we meet $\mathcal{U}$, and then we traverse on $\mathcal{U}$ until we come back on ∂C again. We keep traversing. We may meet $\mathcal{U}$ again if $\mathcal{U}$ has multiple connected components (see Fig. 5). We continue in this way until we arrive at r^*. The order of the arcs of Ξ that appear on $\mathcal{U}$ encountered during the above traversal is called the *traversal order* of $\mathcal{U}$. The following is a crucial lemma that our algorithm relies on. The proof can be found in the full paper.

▶ **Lemma 9.** *Every arc of Ξ has at most one portion on $\mathcal{U}$ and the traversal order of $\mathcal{U}$ is consistent with the right-endpoint left-to-right order of Ξ (i.e., if an arc ξ appears in the front of another arc ξ' in the traversal order of $\mathcal{U}$, then the right endpoint of ξ is to the left of that of ξ').*

3.2 Preprocessing

We perform the following preprocessing algorithm for $Q(C')$. Due to Lemma 9, we are able to extend to our problem a technique from Hershberger and Suri [17] for maintaining the convex hull for a semi-dynamic (deletion-only) set of points in the plane (in the dual plane, the problem is to maintain the lower/upper envelope for a semi-dynamic set of lines). Recall that $m = |Q(C')|$.

We first compute the arcs of Ξ and sort them by their right endpoints from left to right on ∂C. Let T be a complete binary tree whose leaves correspond to arcs in the above order. For each node v, let $\Xi(v)$ denote the subset of arcs in the leaves of the subtree of T rooted at v.

For any subset Ξ' of Ξ, let $\mathcal{U}(\Xi')$ denote the lower envelope of the arcs of Ξ'. We use a tree $T(\Xi')$ (which can be considered as a subtree of T) to represent $\mathcal{U}(\Xi')$. Initially, we have the tree $T(\Xi)$, and later $T(\Xi)$ is modified due to point deletions from $Q(C')$ (and correspondingly arc deletions from Ξ). The tree $T(\Xi')$ is defined as follows. For each arc $\xi \in \Xi'$, we copy the leaf of T storing ξ along with all ancestors of the leaf into $T(\Xi')$. If we define $\Xi'(v) = \Xi(v) \cap \Xi'$ for any node v of T, then v is copied into $T(\Xi')$ if and only if $\Xi'(v) \neq \emptyset$. Later we will add some additional node-fields to $T(\Xi')$ to represent the lower envelope $\mathcal{U}(\Xi')$. We call $T(\Xi')$ an *envelope tree.*

We wish to have each node v of $T(\Xi')$ represent the lower envelope $\mathcal{U}(\Xi'(v))$ of arcs of $\Xi'(v)$, i.e., arcs stored in the leaves of the subtree of $T(\Xi')$ rooted at v. We add a node-field $arcs(v)$ for that purpose. Storing the entire lower envelope $\mathcal{U}(\Xi'(v))$ at each $arcs(v)$ of $T(\Xi')$ leads to superlinear total space. To achieve $O(m)$ space, we use the following standard approach (which has been used elsewhere, e.g., [17, 23]): For each arc ξ stored in a leave $v \in T(\Xi')$, ξ is stored only at $arcs(u)$ for the highest ancestor u of v in $T(\Xi')$ such that ξ

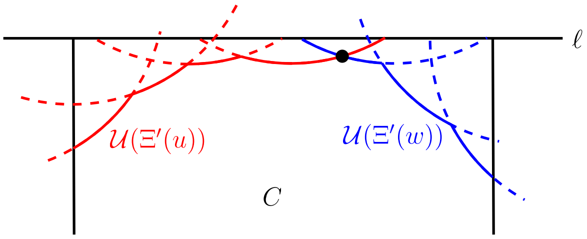


Figure 6 Illustrating Lemma 10: The red (resp., blue) arcs are those from $\Xi'(u)$ (resp., $\Xi'(w)$). There is only one intersection between $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w))$.

contributes an arc in the lower envelope $\mathcal{U}(\Xi'(u))$. Arcs of $arcs(v)$ in each node v of $T(\Xi')$ are stored in a doubly linked list. Note that if v is the root of $T(\Xi')$, then $arcs(v)$ stores the whole lower envelope $\mathcal{U}(\Xi')$ of Ξ'.

The following lemma, which can be easily obtained from Lemma 9, is crucial to the success of our approach. The proof can be found in the full paper.

▶ **Lemma 10.** *For each node $v \in T(\Xi')$, the lower envelopes $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w))$ have at most one intersection, where u and w are the left and right children of v, respectively (see Fig. 6).*

By Lemma 10, we add another node-field $X(v)$ for each node $v \in T(\Xi')$ to store the two arcs that define the intersection of $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w))$, where u and w are the left and right children of v in $T(\Xi')$, respectively. If $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w))$ do not intersect, then $X(v)$ stores the rightmost arc of $\mathcal{U}(\Xi'(u))$ and the leftmost arc of $\mathcal{U}(\Xi'(w))$. As will be seen later in Section 3.3, the two node-fields $X(v)$ and $arcs(v)$ in $T(\Xi')$ allow us to efficiently maintain the envelope tree $T(\Xi')$ subject to deletions of arcs. We next have the following lemma for constructing $T(\Xi)$ initially.

▶ **Lemma 11.** *Given the set Ξ of m arcs, we can build the envelope tree $T(\Xi)$ in $O(m \log m)$ time.*

Proof. First of all, we can construct the tree T in $O(m \log m)$ time by sorting the arcs of Ξ by their right endpoints on ∂C. The rest of the work is thus to compute the fields $arcs(v)$ and $X(v)$ for all nodes v of T. This can be done in a bottom-up manner as follows.

At the outset, we have $arcs(v) = \Xi(v) = \{\xi\}$ for each leaf node $v \in T$, where ξ is the arc stored at v. We also set $X(v)$ to null. Next, we compute $arcs(\cdot)$ and $X(\cdot)$ for other nodes by merging the lower envelopes of their children. Specifically, consider a node v whose left and right children are u and w, respectively. We assume that $arcs(u)$ and $arcs(w)$ store the lower envelopes $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$ in their traversal orders, respectively. The first thing is to compute the lower envelope $\mathcal{U}(\Xi(v))$. By Lemma 10, $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$ have at most one intersection. Since each lower envelope is x-monotone, $\mathcal{U}(\Xi(v))$, which is also the lower envelope of $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$, can be computed by a standard line sweep procedure. Specifically, a vertical sweeping line ℓ' sweeps the plane from left to right. During the sweeping, we maintain the two arcs of $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$ intersecting ℓ', respectively. An event happens if ℓ' hits a vertex of either $\mathcal{U}(\Xi(u))$ or $\mathcal{U}(\Xi(w))$. The sweeping procedure takes $O(|\Xi(v)|)$ time (note that $\Xi(v) = \Xi(u) \cup \Xi(w)$).

- If $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$ do not have any intersection, then $\mathcal{U}(\Xi(v))$ is just the concatenation of $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$, i.e., we concatenate $arcs(u)$ and $arcs(w)$ and store the result at $arcs(v)$; we also need to reset both $arcs(u)$ and $arcs(w)$ to null. In addition, $X(v)$ is set to including the rightmost arc of $\mathcal{U}(\Xi(u))$ and the leftmost arc of $\mathcal{U}(\Xi(w))$.

- If $\mathcal{U}(\Xi(u))$ and $\mathcal{U}(\Xi(w))$ have an intersection, say, a^*, then let $\xi_u \in \mathcal{U}(\Xi(u))$ and $\xi_v \in \mathcal{U}(\Xi(v))$ be the two arcs that intersect at a^*. We concatenate the part of $\mathcal{U}(\Xi(u))$ left to a^* and the part of $\mathcal{U}(\Xi(w))$ right to a^* (ξ_u and ξ_w are cut off at a^*); the result is $\mathcal{U}(\Xi(v))$ and we store it into $arcs(v)$. Further, arcs left to a^* (including ξ_u) in $\mathcal{U}(\Xi(u))$ and arcs right to a^* (including ξ_w) in $\mathcal{U}(\Xi(w))$ are removed from $arcs(u)$ and $arcs(w)$, respectively. In addition, $X(v)$ is set to $\{\xi_u, \xi_w\}$.

As such, computing the node-fields of v takes $O(|\Xi(v)|)$ time. Doing this for all nodes v in the same level of the tree takes $O(m)$ time as the union of $\Xi(v)$ of all nodes v in the same level is exactly Ξ. Therefore, the construction of the envelope tree $T(\Xi)$ can be done in $O(m \log m)$ time in total. ◀

The above finishes our preprocessing for the points $Q(C')$, which takes $O(m \log m)$ time and $O(m)$ space. Our preprocessing builds the envelope tree $T(\Xi)$, which is our data structure $\mathcal{D}_C(C')$. Once points from $Q(C')$ are deleted we use Ξ' to refer to the subset of Ξ defined by the remaining points and use $T(\Xi')$ to refer to the corresponding envelope tree.

3.3 Handling UDRE queries and point deletions

We now discuss how to handle the UDRE queries and point deletions.

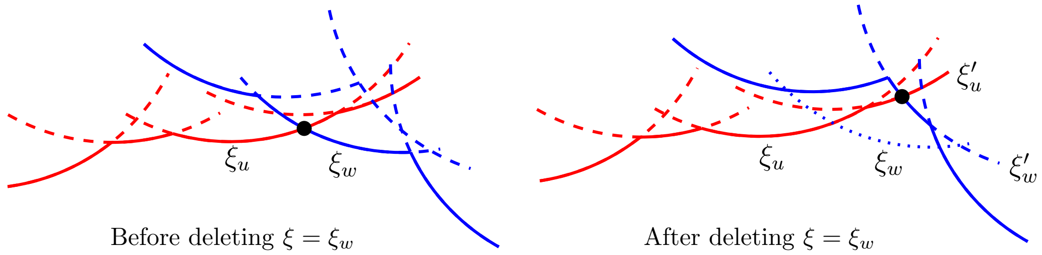
UDRE queries. Handling the UDRE queries is relatively easy. Consider a query point p in the cell C. We wish to determine whether $A_p \cap Q(C') = \emptyset$, and if not, return a point $q \in A_p \cap Q(C')$. Let Ξ' be the set of arcs defined by the points in the current set $Q(C')$. As discussed before, it suffices to determine whether p is above the lower envelope $\mathcal{U}(\Xi')$. To this end, since $\mathcal{U}(\Xi')$ is x-monotone, let a and b be the two adjacent vertices of $\mathcal{U}(\Xi')$ such that p's x-coordinate is between those of a and b. Let ξ_q be the arc that contains the portion of $\mathcal{U}(\Xi')$ between a and b, where q is the center of the arc (and thus $q \in Q(C')$). As such, p is above $\mathcal{U}(\Xi')$ if and only if p is above ξ_q (i.e., p is inside the unit-disk A_q). If yes, then $q \in A_p \cap Q(C')$ and thus we can return q as the answer to the query. Therefore, it suffices to compute the arc ξ_q. To this end, one may attempt to perform binary search on the vertices of $\mathcal{U}(\Xi')$ to find a and b first. However, although the whole $\mathcal{U}(\Xi')$ is stored in $arcs(v)$ at the root v, arcs of $arcs(v)$ are stored in a doubly linked list, which does not support binary search. To circumvent the issue, we can actually perform binary search using the node-fields $X(\cdot)$ of $T(\Xi')$ as follows.

Observe that each vertex of $\mathcal{U}(\Xi')$ appears as the intersection of the two arcs of $X(v)$ for some node $v \in T(\Xi')$. The subtree of $T(\Xi')$ rooted at any node v represents $\mathcal{U}(\Xi'(v))$ by the intersections of the arcs of $X(\cdot)$ stored at its nodes. To find ξ_q, starting from the root, for each node v of $T(\Xi')$, we compute the intersection a^* of the arcs of $X(v)$. If the x-coordinate of p is smaller or equal to that of a^*, we proceed on the left subtree of v recursively; otherwise, we proceed on the right subtree. At the end we will reach a leaf and the arc stored at the leaf is ξ_q. As such, ξ_q can be found in $O(\log m)$ time.

Therefore, each UDRE query can be answered in $O(\log m)$ time.

Deletions. Next, we discuss point deletions. To delete a point q from $Q(C')$, it boils down to deleting the arc ξ_q defined by q from the envelope tree $T(\Xi')$. The next lemma provides an algorithm for this.

▶ **Lemma 12.** *Deleting an arc from the envelope tree $T(\Xi')$ can be done in $O(\log m)$ amortized time.*



Before deleting $\xi = \xi_w$

After deleting $\xi = \xi_w$

Figure 7 Illustrating the deletion of $\xi = \xi_w$. The red (resp., blue) arcs are those from $\Xi'(u)$ (resp., $\Xi'(w)$).

Proof. Let ξ be the arc we wish to delete from $T(\Xi')$ and let z be the leaf node of the tree storing ξ. To delete ξ, we need to update $arcs(\cdot)$ and $X(\cdot)$ for all ancestors of z.

The algorithm is recursive. Starting from the root, for each node v, we process it by calling Delete(ξ, v) as follows. We assume that $arcs(v)$ now stores the whole lower envelope $\mathcal{U}(\Xi'(v))$, which is true initially when v is the root. Let u and w denote the left and right children of v, respectively. We assume that the leaf z is in the right subtree of v since the other case is symmetric. Let $X(v) = \{\xi_u, \xi_w\}$, with $\xi_u \in \mathcal{U}(\Xi'(u))$ and $\xi_w \in \mathcal{U}(\Xi'(w))$, i.e., the intersection of ξ_u and ξ_w, denoted by a^*, is the intersection between $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w))$. We first restore $\mathcal{U}(\Xi'(u))$, by concatenating the part of $arcs(v)$ left to a^* and $arcs(u)$. Restoring $\mathcal{U}(\Xi'(w))$ can be done in a similar way. Depending on whether $w = z$, there are two cases.

If w is the leaf z (which is the base case of our recursive algorithm), then $arcs(w) = \{\xi\}$ and we reset the right child of v and field $X(v)$ to null. We also reset $arcs(v) = arcs(u)$ and $arcs(u) = null$.

If w is not z, then to update $arcs(v)$ and $X(v)$, observe that if $\xi \notin X(v)$, then deleting ξ does not affect the intersection between $\mathcal{U}(\Xi'(u))$ and the new lower envelope $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$, i.e., $X(v)$ does not change. Hence, if $\xi \notin X(v)$, we proceed on w by calling Delete(ξ, w). After Delete(ξ, w) is returned, the new $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$ is stored in $arcs(w)$ and we cut $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$ using $X(v)$ to obtain $arcs(v)$ in the same way as the tree construction algorithm in Lemma 11, which takes $O(1)$ time as each $arcs(\cdot)$ is stored by a doubly linked list. In the following, we discuss the case where $\xi \in X(v) = \{\xi_u, \xi_w\}$.

Since ξ is in the right subtree of v, ξ must be ξ_w. In this case, $X(v)$ will be changed after the deletion of ξ and thus we need to compute the new arcs that define the intersection of $\mathcal{U}(\Xi'(u))$ and the new lower envelope $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$ (see Fig. 7). We proceed on w by calling Delete(ξ, w). After Delete(ξ, w) is returned, the new $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$ is stored in $arcs(w)$. Let $\{\xi'_u, \xi'_w\}$ be the new $X(v)$ to be computed, with ξ'_v and ξ'_w in $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$, respectively. Observe that ξ'_u cannot lie to the left of ξ_u in $arcs(u)$ while ξ'_w must lie on the part of the new $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$ between the two old neighbors of ξ (=ξ_w) on $\mathcal{U}(\Xi'(w))$ (see Fig. 7). As such, we compute ξ'_u and ξ'_w using a line sweep procedure that is similar to the algorithm in Lemma 11, but to make the algorithm faster, due to the above observation it suffices to start the sweeping line from the left of the following two arcs: ξ_u and the left neighbor of ξ in the original lower envelope $\mathcal{U}(\Xi'(w))$. We stop the sweeping once the intersection of $\mathcal{U}(\Xi'(u))$ and $\mathcal{U}(\Xi'(w) \setminus \{\xi\})$ is found, after which, we reset $arcs(v)$ as well as $arcs(u)$ and $arcs(w)$ in constant time in a way similar to the algorithm in Lemma 11.

The pseudocode summarizing the algorithm can be found in the full paper.

For the time analysis, the time we spend on each node v is $O(1)$ except the line sweep procedure for computing ξ'_u and ξ'_w in the case where $\xi \in X(v)$. The procedure takes time $O(1 + k_u + k_w)$, where k_u is the number of arcs between ξ_u and ξ'_u in $\mathcal{U}(\Xi'(u))$ and k_w is the number of arcs between ξ_w and ξ'_w in $\mathcal{U}(\Xi'(w))$. Observe that the arcs between ξ_u and ξ'_u in $\mathcal{U}(\Xi'(u))$ are moved up from node u to node v after the deletion of ξ (i.e., they were originally stored at $arcs(u)$ but are stored at $arcs(v)$ after the deletion). Similarly, the arcs between ξ_w and ξ'_w in $\mathcal{U}(\Xi'(w))$ are moved up to w from some lower levels after the deletion (see Fig. 7). Because each arc can be moved up at most $O(\log m)$ times for all m point deletions of $Q(C')$, the total sum of $k_u + k_w$ for all deletions is bounded by $O(m \log m)$. As such, each deletion takes $O(\log m)$ amortized time. ◀

3.4 Putting everything together

The above shows that we can build a data structure $\mathcal{D}_C(C')$ for the points of $Q(C')$ with respect to C in $O(m \log m)$ time and $O(m)$ space, such that each UDRE query with a query point in C can be answered in $O(\log m)$ time and deleting a point from $Q(C')$ can be handled in $O(\log m)$ amortized time.

To solve our original problem on Q, i.e., proving Theorem 3, for each cell $C \in \mathcal{C}$, we build data structures $\mathcal{D}_C(C')$ for all cells $C' \in N(C)$. Because $|N(C)| = O(1)$ for every $C \in \mathcal{C}$ and each cell C' is in $N(C)$ for a constant number of cells $C \in \mathcal{C}$, the total space for all these data structures $\mathcal{D}_C(C')$ is $O(n)$ and the total preprocessing time is $O(n \log n)$.

For each UDRE query with a query point p, we first use Lemma 8(2) to determine whether p is in a cell of $\mathcal{C}$. If not, then by Observation 7, $A_p \cap Q = \emptyset$ and thus we are done with the query. Otherwise, Lemma 8(2) will return the cell C that contains p as well as $N(C)$. Then, for each $C' \in N(C)$, we solve the query using the data structure $\mathcal{D}_C(C')$. The total query time is $O(\log n)$ as $|N(C)| = O(1)$.

To delete a point q from Q, using Lemma 8(2) we first find the cell C' that contains q as well as $N(C')$. Notice that $N(C')$ exactly consists of those cells C with $C' \in N(C)$. We then delete q from the data structure $\mathcal{D}_C(C')$ for each $C \in N(C')$. As $|N(C')| = O(1)$, the total deletion time is $O(\log n)$ amortized time. This proves Theorem 3.

4 Conclusion

In this paper, we presented an $O(n^{4/3} \log^3 n)$ time algorithm for computing a Euclidean minimum bottleneck moving spanning tree for a set of n moving points in the plane, which significantly improves the previous $O(n^2)$ time solution [2]. To solve the problem, we first solved the decision problem in $O(n^{4/3} \log^2 n)$ time. This is done by reducing it to the problem of computing a common spanning tree in two unit-disk graphs. To avoid computing the unit-disk graphs explicitly, which would cost $\Omega(n^2)$ time, we used a batched range searching technique [19] to obtain a compact representation for searching one graph, and derived a semi-dynamic (deletion-only) unit-disk range emptiness query data structure for searching the other graph. We believe our data structure is interesting in its own right and will certainly find applications elsewhere. We finally remark that although in our problem each moving point is required to move linearly with constant velocity, our algorithm still works for other types of point movements as long as Observation 1 holds.

References

1 Pankaj K. Agarwal and Jiří Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.

2 Hugo A. Akitaya, Ahmad Biniaz, Prosenjit Bose, Jean-Lou De Carufel, Anil Maheshwari, Luís Fernando Schultz Xavier da Silveira, and Michiel Smid. The minimum moving spanning tree problem. In *Proceedings of the 17th Workshop on Algorithms and Data Structures (WADS)*, pages 15–28, 2021.

3 Mikhail J. Atallah. Dynamic computational geometry. In *Proceedings of 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 92–99, 1983.

4 Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31:1–28, 1999.

5 Gerth S. Brodal and Riko Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 617–626, 2002.

6 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry: Theory and Applications*, 48(4):360–367, 2015.

7 Paolo M. Camerini. The min-max spanning tree problem and some extensions. *Information Processing Letters*, 7:10–14, 1978.

8 Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmaic amortized time. *Journal of the ACM*, 48:1–12, 2001.

9 Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *Journal of the ACM*, 57:16:1–16:15, 2010.

10 Timothy M. Chan. Optimal partition trees. *Discrete and Computational Geometry*, 47:661–690, 2012.

11 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete and Computational Geometry*, 64:1235–1252, 2020.

12 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

13 Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 24:1–24:13, 2018.

14 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete mathematics*, 86:165–177, 1990.

15 David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete and Computational Geometry*, 13:111–122, 1995.

16 Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

17 John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numerical Mathematics*, 32(2):249–267, 1992.

18 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2495–2504, 2017.

19 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.

20 Jiří Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8:315–334, 1992.

21 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10:157–182, 1993.

22 Tomomi Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proceedings of the Japanese Conference on Discrete and Computational Geometry (JCDCG)*, pages 194–200, 1998.

23 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer System Sciences*, 23(2):166–204, 1981.

24 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

25 Zahed Rahmati and Alireza Zarei. Kinetic Euclidean minimum spanning tree in the plane. *Journal of Discrete Algorithms*, 16:2–11, 2012.

26 Haitao Wang. Unit-disk range searching and applications. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 32:1–32:17, 2022.

27 Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020.

28 Haitao Wang and Yiming Zhao. An optimal algorithm for L_1 shortest paths in unit-disk graphs. In *Proceedings of the 33rd Canadian Conference on Computational Geometry (CCCG)*, pages 211–218, 2021.

29 Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. In *Proceedings of the 17th International Symposium of Algorithms and Data Structures (WADS)*, pages 655–668, 2021.

30 Haitao Wang and Yiming Zhao. Reverse shortest path problem for weighted unit-disk graphs. In *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 135–146, 2022.

Improved Approximation Algorithms for the Traveling Tournament Problem

Jingyang Zhao
University of Electronic Science and Technology of China, Chengdu, China

Mingyu Xiao[1]
University of Electronic Science and Technology of China, Chengdu, China

Chao Xu
University of Electronic Science and Technology of China, Chengdu, China

Abstract

The Traveling Tournament Problem (TTP) is a well-known benchmark problem in the field of tournament timetabling, which asks us to design a double round-robin schedule such that each pair of teams plays one game in each other's home venue, minimizing the total distance traveled by all n teams (n is even). TTP-k is the problem with one more constraint that each team can have at most k consecutive home games or away games. The case where $k = 3$, TTP-3, is one of the most investigated cases. In this paper, we improve the approximation ratio of TTP-3 from $(1.667 + \varepsilon)$ to $(1.598 + \varepsilon)$, for any $\varepsilon > 0$. Previous schedules were constructed based on a Hamiltonian cycle of the graph. We propose a novel construction based on triangle packing. Then, by combining our triangle packing schedule with the Hamiltonian cycle schedule, we obtain the improved approximation ratio. The idea of our construction can also be extended to $k \geq 4$. We demonstrate that the approximation ratio of TTP-4 can be improved from $(1.750+\varepsilon)$ to $(1.700+\varepsilon)$ by the same method. As an additional product, we also improve the approximation ratio of LDTTP-3 (TTP-3 where all teams are allocated on a straight line) from $4/3$ to $(6/5 + \varepsilon)$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Scheduling algorithms

Keywords and phrases Sports scheduling, Traveling Tournament Problem, Approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.83

Funding The work is supported by the National Natural Science Foundation of China, under grant 61972070.

1 Introduction

In the field of the tournament schedule, the traveling tournament problem (TTP) is a widely known benchmark problem that was first systematically introduced in [10]. This problem aims to find a double round-robin tournament satisfying some constraints, minimizing the total distance traveled by all participant teams. In a double round-robin tournament of n teams, each team will play 2 games against each of the other $n - 1$ teams, one at its home venue and one at its opponent's home venue. Additionally, each team should play one game a day, all games need to be scheduled on $2(n - 1)$ consecutive days, and so there are exactly $n/2$ games on each day. According to the definition, we know that n is always even. For TTP, we have the following two basic constraints or assumptions on the double round-robin tournament.

[1] Corresponding author

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 83; pp. 83:1–83:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Traveling Tournament Problem (TTP).

- *No-repeat*: Two teams cannot play against each other on two consecutive days.
- *Direct-traveling*: Before the first game, all teams are at home, and they will return home after the last game. Furthermore, a team travels directly from its game venue on ith day to its game venue on $(i+1)$th day.

TTP-k, a well-known variant of TTP, is to add the following constraint on the maximum number of consecutive home games and away games.

- *Bounded-by-k*: Each team can have at most k consecutive home games or away games.

The smaller the value of k, the more frequently a team has to return home. By contrast, if k is very large, say $k = n - 1$, then this constraint loses meaning, and TTP-k becomes TTP where a team can schedule their travel distance as short as that in the traveling salesman problem (TSP).

A weight function w on the complete undirected graph is called a *metric* if it satisfies the symmetry and triangle inequality properties: $w(a,b) = w(b,a)$ and $w(a,c) \leq w(a,b) + w(b,c)$ for all $a,b,c \in V$.

The input of TTP-k contains a complete graph where each vertex is a team, and the weight between two vertices u and v is the distance from the home of team u to the home of team v. In this paper, we only consider the case when the weight function w is a metric. Due to page limitations, the proofs of some lemmas and theorems marked with '*' may be omitted or incomplete. The full proofs can be found in the full version of this paper.

1.1 Related Work

TTP and TTP-k are difficult optimization problems. The NP-hardness of TTP and TTP-k with $k \geq 3$ has been established [2, 24, 5]. Although the hardness of TTP-2 is still not formally proved, it is believed that TTP-2 is also hard since it is also not easy to construct a feasible solution to it. In the literature there is a large number of contributions on approximation algorithms [25, 28, 31, 30, 6, 21, 29, 18, 27, 16, 17] and heuristic algorithms [11, 20, 1, 9, 13, 14].

For heuristic algorithms, most known works are concerned with the case of $k = 3$. Since the search spaces of TTP and TTP-k are usually very large, many instances of TTP-3 with more than 10 teams in the online benchmark [26, 3] have not been completely solved even by using high-performance machines.

In terms of approximation algorithms, most results are based on the assumption that the distance holds the symmetry and triangle inequality properties. This is natural and practical in the sports schedule. For $k = 2$, one significant contribution to TTP-2 was done by Thielen and Westphal [25]. They proposed a $(1 + 16/n)$-approximation algorithm for $n/2$ being even and a $(3/2 + O(1/n))$-approximation algorithm for $n/2$ being odd. Currently, approximation ratios for these two cases have been improved to $(1 + 3/n)$ and $(1 + 12/n)$, respectively [30, 31]. For $k = 3$, the first approximation algorithm, proposed by Miyashiro *et al.*, admits a $2 + O(1/n)$ approximation ratio using the Modified Circle Method [21]. Then, the approximation ratio was improved to $5/3 + O(1/n)$ by Yamaguchi *et al.* [29]. Their approximation algorithm also works for $3 < k \ll n$. For $k = 4$, the ratio is $1.750 + O(1/n)$ and for $4 < k \ll n$, the ratio is $(5k - 7)/(2k) + O(k/n)$ [29]. For $k = \Theta(n)$, Westphal and Noparlik proposed a 5.875 approximation algorithm for any choice of $k \geq 4$ and $n \geq 6$ [27], and Imahori *et al.* gave a 2.75 approximation algorithm for $k = n - 1$ where they also proved the approximation ratio can be further improved to 2.25 if the optimal TSP is given [18]. The current best approximation ratio for $k = 3$ is still $5/3 + O(1/n)$ [29].

We refer the readers to [3] for more variants of TTP (including the traveling tournament problem with predefined venues, the time-relaxed traveling tournament problem, etc.) with detailed benchmarks for each.

1.2 Our Results

In this paper, we consider TTP-k with the cases of $k = 3$ and $k = 4$. Our contributions can be summarized as follows.

We mainly focus on TTP-3. Firstly, we analyze the structural properties in more detail, which leads us to strengthen some of the current-known lower bounds and propose several new ones. Secondly, we design an approximation algorithm that improves the approximation ratio from $(5/3 + \varepsilon)$ to $(139/87 + \varepsilon)$. Our algorithm consists of two constructions. The first construction is based on the Hamiltonian cycle, which is a well-known method. The Hamiltonian cycle used is commonly generated by the Christofides-Serdyukov algorithm, while we propose a new Hamiltonian cycle that uses the minimum weight perfect matching. The second construction proposed by us is based on triangle packing which can be seen as a generalization of the matching packing schedule in TTP-2.

For a special case of TTP-3 where all teams are located on a line, Linear Distance TTP-3 (LDTTP-3), we prove that the approximation ratio of our triangle packing construction can achieve $(6/5 + \varepsilon)$, which improves the previous approximation ratio of $4/3$ [15].

Finally, we extend our method to TTP-4 and show that we can improve the approximation ratio from $(7/4 + \varepsilon)$ to $(17/10 + \varepsilon)$.

2 Preliminaries

We will always use n to denote the total number of teams in the problem. The set of n teams is denoted by $V = \{t_1, t_2, \dots, t_n\}$. Recall that n is even. For TTP-3, there are three cases of n we consider: $n \equiv 0 \pmod 6$, $n \equiv 2 \pmod 6$ and $n \equiv 4 \pmod 6$. Due to the different structural properties, these three cases have to be handled separately. We mainly describe the case of $n \equiv 0 \pmod 6$ due to page limitations. So from here on, we assume that n is a number divisible by 6.

We use $G = (V, E)$ to denote the complete graph on the n vertices representing the n teams. There is a positive weight function $w : E \to \mathbb{R}_{\geq 0}$ on the edges of G. We often write $w(u, v)$ to mean the weight of the edge uv, instead of $w(uv)$. Note that $w(u, v)$ would be the same as the distance between the home of team u and the home of team v. For any weight function $w : X \to \mathbb{R}_{\geq 0}$, we extend it to subsets of X. Define $w(Y) = \sum_{x \in Y} w(x)$ for $Y \subseteq X$.

The weight of a minimum weight spanning tree in G is denoted by $\mathrm{MST}(G)$. We use $\delta(u)$ to denote the set of edges incident on u in G. We also use $\deg(u)$ to denote the weighted degree of a vertex. That is the total weight of all edges incident on u in G, i.e., $\deg(u) = w(\delta(u))$. We also let Δ to be the sum of the weighted degrees, i.e., $\Delta = \sum_{u \in V} \deg(u) = 2w(E)$.

A cycle on k vertices is called a k-cycle. A triangle uvw is a 3-cycle on three different vertices $\{u, v, w\}$. Two subgraphs or sets of edges are *vertex-disjoint* if they do not share a common vertex. A *triangle packing* in G, denoted by T, is a set of edges such that each component is a triangle, and all vertices are covered. Equivalently, it can be seen as the edges of m vertex-disjoint triangles. Similarly, a P_3 *path* is a simple path on three different vertices $\{u, v, w\}$, which can be represented by u-v-w. A P_3 *path packing* in G is a set of edges such that every component is a P_3 path, and each vertex is covered. We can obtain a triangle packing from a P_3 packing by connecting the two non-adjacent vertices in each component.

Let $m = \frac{n}{3}$, and then m is an even number. For a fixed triangle packing T of G, let the components be $u_1, \dots, u_m$. $U = \{u_1, \dots, u_m\}$ is a partition of V. Each $u \in U$ is referred as a *super-team*. We define a complete graph $H = (U, F)$ on U. We will also have a cost function $c(u, v)$ defined on vertices in U. It is $c(u, v) = \sum_{u' \in u, v' \in v} w(u', v')$ for each edge $uv \in F$. For simplicity we also define $c(u, u) = 0$. Despite not using this property, it is worth noting that the cost function c is also a metric.

We also define $c(u)$ for $u \in U$ to be the same as $w(a, b) + w(b, c) + w(c, a)$ for $a, b, c \in u$. Note that $c(U) = \sum_{u \in U} c(u) = w(T)$. We can easily get

$$\frac{1}{2}\Delta = w(E) = c(F) + c(U) = c(F) + w(T). \tag{1}$$

The remaining parts of the paper are organized as follows. In Section 3, we focus on TTP-3. Specifically, in Section 3.1, we introduce some basic notations and propose several new lower bounds. In Section 3.2, we give a brief introduction to the well-known construction based on the Hamiltonian cycle. In Section 3.3, we propose a novel construction based on triangle packing. Although these two constructions cannot make any improvement separately, in Section 3.4, we show that together they can guarantee an improved approximation ratio for TTP-3. In Section 4, we analyze the approximation ratio of our algorithm for LDTTP-3. In Section 5, we extend our methods and prove that we can also improve the approximation ratio for TTP-4.

3 TTP-3

3.1 The Independent Lower Bounds

For TTP and TTP-k, a well-known method to obtain the lower bounds is to use an independent relaxation [4, 10]. The basic idea is to obtain a lower bound on the traveling distance of each team independently without considering the feasibility of other teams and then sum them together. Although there exist many independent lower bounds for TTP-3 [21, 29, 27], we can not use them directly to get our result. We are interested in a stronger bound. Before we make some observations on the independent lower bounds, we first need to explore some properties of TTP-k.

For TTP-k, each team needs to visit each other team's home once in the tournament. A *road trip* of a team v is a simple cycle starting and ending at v. A *k-itinerary* of v, is a connected subgraph of G that consists of road trips with each simple cycle of length at most $k+1$, and each vertex other than v in V has degree 2.

For TTP-3, the length of each road trip is at most 4. For simplicity, we omit k when k is implicit. In the remainder of this section, $k = 3$.

An itinerary is *optimal* for a team if it is the itinerary of minimum weight.

Considering an optimal itinerary I_v for team v, we will use $w(I_v)$ to denote the weight of the optimal itinerary for team v. Then, $\psi = \sum_{v \in V} w(I_v)$ is a simple independent lower bound for the minimum weight solution of TTP-3. Note that this lower bound was also used in the experiment [10]. However, it is NP-hard to compute $w(I_v)$. Hence, we want to find an alternative lower bound for each team's optimal itinerary.

When $n \equiv 0 \pmod 6$ and $n \equiv 4 \pmod 6$, we can prove that there always exists an optimal itinerary with no 2-cycles.

▶ **Lemma 1.** *For TTP-3 with the cases of* $n \equiv 0 \pmod 3$ *and* $n \equiv 1 \pmod 3$*, there exists an optimal itinerary with no 2-cycles for each team.*

Proof. Consider an optimal itinerary I_v of team v with the minimum number of 2-cycles. Assume I_v contains a 2-cycle. Since all cycles share the vertex v, by the triangle inequality, we can get a 3-cycle by shortcutting two 2-cycles, and the 3-cycle has a weight no greater than the sum weight of the two 2-cycles. This shows that I_v contains exactly one 2-cycle. Similarly, by the triangle inequality, a 4-cycle can be obtained by shortcutting one 2-cycle and one 3-cycle without increasing the total weight. This shows there cannot be any 3-cycle. Thus, there exists only one 2-cycle, and the rest of the cycles are all 4-cycles. Thus, we will get $n \equiv 2 \pmod 3$, a contradiction to $n \equiv 0 \pmod 3$ or $n \equiv 1 \pmod 3$. ◀

3.1.1 Bounds on optimal itinerary

For this subsection, we fix a single team v and start to consider the optimal itinerary I_v for this team. We will write I as I_v to simplify the notation within this subsection. We will give a more refined analysis than previous results [29].

Recall that we consider the case of $n \equiv 0 \pmod 6$ here. By Lemma 1, there exists an optimal itinerary I with no 2-cycles, and it consists of a set of 4-cycles and a set of 3-cycles. Hence we will partition I into two sets of edges I_3 and I_4, where I_3 consists of all 3-cycles and I_4 consists of all 4-cycles. Define $0 \leq \gamma \leq 1$, such that $w(I_4) = \gamma w(I)$, and so $w(I_3) = (1-\gamma)w(I)$. Hence γ measures the proportion of weights of the 4-cycles compared to the entire itinerary. The edges in G incident to v in I are called *home-edges*, which is the same as $\delta(v) \cap I$. Let a and b be the proportion of weights of the home-edges in I_4 and I_3, respectively. Namely, $aw(I_4) = w(I_4 \cap \delta(v))$ is the weight of all home-edges in I_4, and $bw(I_3) = w(I_3 \cap \delta(v))$ is the weight of all home-edges in I_3.

Now, we are ready to give some stronger bounds for the optimal itinerary.

▶ **Lemma 2.** *Let C be a minimum weight Hamiltonian cycle in G. Then $w(I) \geq w(C)$.*

Proof. According to the definition of the itinerary, we know that each vertex in I has an even degree. Then, we can get an Euler tour in I and obtain a Hamiltonian cycle in graph G by shortcutting I. Hence $w(I) \geq w(C)$. ◀

We also recall the following result.

▶ **Lemma 3** ([7, 23]). *For a graph G, let C be a minimum weight Hamiltonian cycle, and C' be a Hamiltonian cycle obtained by the Christofides-Serdyukov algorithm. Then $w(C') \leq MST(G) + \frac{1}{2}w(C)$.*

For ease of proofs in the rest of the section, we also define V_3 and V_4 that partition the vertices in $V \setminus \{v\}$, where V_3 consists of all vertices in I_3 except v, and V_4 consists of all vertices in I_4 except v. Our results are mostly simple counting arguments.

▶ **Lemma 4.** $(1 - \frac{1}{2}\gamma + a\gamma)w(I) \geq (\frac{1}{2} + a)\gamma w(I) + b(1-\gamma)w(I) \geq \deg(v)$.

Proof. By the definition of b, we have $b \leq 1$. So, we have $(1 - \frac{1}{2}\gamma + a\gamma)w(I) = (\frac{1}{2} + a)\gamma w(I) + (1-\gamma)w(I) \geq (\frac{1}{2} + a)\gamma w(I) + b(1-\gamma)w(I)$. The left inequality holds. Now we show the right inequality.

First, one can see $\deg(v) = \sum_{u \in V_3} w(u,v) + \sum_{u \in V_4} w(u,v)$.

Next, recall that $b(1-\gamma)w(I) = w(I_3 \cap \delta(v))$ is the total weight of all home-edges in 3-cycles. Thus, we have $\sum_{u \in V_3} w(u,v) = b(1-\gamma)w(I)$. Similarly $a\gamma w(I) = w(I_4 \cap \delta(v))$.

For any 4-cycle $vv_1v_2v_3$ in I_4, we note that the edges v_1v and v_3v are home-edges which can be counted by $w(I_4 \cap \delta(v))$, but the edge v_2v is not a home-edge. By the triangle inequality, we know that $w(v,v_2) \leq \frac{1}{2}(w(v,v_1) + w(v_1,v_2) + w(v_2,v_3) + w(v_3,v))$. Therefore, the weight of the uncounted edge is at most half that of the 4-cycle.

Thus, we have $(\frac{1}{2}+a)\gamma w(I) \geq \sum_{u\in V_4} w(u,v)$. Then, we have $(\frac{1}{2}+a)\gamma w(I)+b(1-\gamma)w(I) \geq \sum_{u\in V_3} w(u,v) + \sum_{u\in V_4} w(u,v) = \deg(v)$. ◀

▶ **Lemma 5.** $(1-\frac{1}{2}a)\gamma w(I) + (1-\frac{1}{2}b)(1-\gamma)w(I) \geq MST(G)$.

Proof. In the optimal itinerary of v, we note that the total weight of all home-edges is $a\gamma w(I) + b(1-\gamma)w(I)$. For each cycle, there are exactly two home-edges. We can delete the longer edge from each cycle from I, and we can get a spanning tree with the total weight less than $(1-\frac{1}{2}a)\gamma w(I) + (1-\frac{1}{2}b)(1-\gamma)w(I)$. Since the weight of a minimum spanning tree is $\mathrm{MST}(G)$, we have that $(1-\frac{1}{2}a)\gamma w(I) + (1-\frac{1}{2}b)(1-\gamma)w(I) \geq \mathrm{MST}(G)$. ◀

Note that our bounds in Lemmas 4 and 5 are stronger than that in [29]. Next, we will propose two new lower bounds on minimum weight matching and triangle packing.

▶ **Lemma 6.** *Let M be a minimum weight perfect matching in G. Then $\frac{1}{2}\gamma w(I) + (1-b)(1-\gamma)w(I) \geq w(M)$.*

Proof. We note that the number of vertices in V_3 is even but odd in V_4. Since any pair of 3-cycles only share one common vertex v, after we delete both of home-edges for each 3-cycle, we can get a perfect matching M_1 in graph $G[V_3]$ with a total weight of $(1-b)(1-\gamma)w(I)$.

Then, by a similar argument with the proof of Lemma 2, we know that $\gamma w(I) = w(I_4)$, is greater than the weight of the minimum weight Hamiltonian cycle in graph $G[V_4 \cup \{v\}]$. Since the number of vertices in this graph is even, any Hamiltonian cycle in this graph can be decomposed into two perfect matching. Thus, we can get a perfect matching M_2 in this graph with a total weight less than $\frac{1}{2}\gamma w(I)$. Therefore, $M_1 \cup M_2$ is a perfect matching, and $w(M) \leq w(M_1 \cup M_2) \leq \frac{1}{2}\gamma w(I) + (1-b)(1-\gamma)w(I)$. ◀

▶ **Lemma 7.** *For a graph G, let P^* be a minimum weight P_3 packing, and T^* be a minimum weight triangle packing. Then $(\frac{2}{3}+\frac{1}{3}\gamma-a\gamma)w(I) = (1-a)\gamma w(I) + \frac{2}{3}(1-\gamma)w(I) \geq w(P^*) \geq \frac{1}{2}w(T^*)$.*

Proof. First, we show $w(P^*) \geq \frac{1}{2}w(T^*)$. Let T' be the triangle packing obtained by completing the P_3 packing. For any P_3 path in P^*, saying uvw, we obtain $w(u,v)+w(v,w) \geq w(u,w)$ by the triangle inequality. This shows $2w(P^*) \geq w(T') \geq w(T^*)$, and we are done.

We note that the number of vertices is divisible by 3 in V_4 but not in V_3. Since any pair of 4-cycles only share one common vertex v, after we delete both of home-edges for each 4-cycle, we can get a P_3 path packing P' in graph $G[V_4]$ such that $w(P') = (1-a)\gamma w(I)$.

Then, by a similar argument with the proof of Lemma 2, we know that $(1-\gamma)w(I) = w(I_3) \geq w(C)$ where C is the minimum weight Hamiltonian cycle in graph $G[V_3 \cup \{v\}]$. Since the number of vertices in this graph equals n minus the number of vertices in V_4, which is divisible by 3, we can delete some edges in C to get a vertex disjoint P_3 path packing P'' such that $w(P'') \leq \frac{2}{3}w(C)$. $P' \cup P''$ is a P_3 packing in $G[V]$. Thus, we have that $(1-a)\gamma w(I) + \frac{2}{3}(1-\gamma)w(I) \geq w(P' \cup P'') \geq w(P^*)$. ◀

3.1.2 Independent lower bounds

Now, we are ready for the independent lower bounds, which are found by summing the individual optimal itineraries. Note that the notations I, I_3, I_4, V_3, V_4, γ, a and b in the previous subsection refer to I_v, $I_{v,3}$, $I_{v,4}$, $V_{v,3}$, $V_{v,4}$, γ_v, a_v and b_v. We omitted the subscripts for the simplification.

For each team v, I_v is the optimal itinerary of v. Recall that $\psi = \sum_{v\in V} w(I_v)$. $I_{v,3}$ and $I_{v,4}$ consist of all 3-cycles and all 4-cycles of I_v, respectively.

Define γ, a and b so that $\gamma\psi = \sum_{v\in V} w(I_{v,4}) = \sum_{v\in V} \gamma_v w(I_v)$, $a\gamma\psi = \sum_{v\in V} w(\delta(v) \cap I_{v,4}) = \sum_{v\in V} a_v\gamma_v w(I_v)$, and $b(1-\gamma)\psi = \sum_{v\in V} w(\delta(v) \cap I_{v,3}) = \sum_{v\in V} b_v(1-\gamma_v)w(I_v)$ hold. Then, we have $0 \leq \gamma, a, b \leq 1$.

The lemmas we proved previously can be summed together to obtain different independent lower bounds.

▶ **Lemma 8.** *Let C be a minimum weight Hamiltonian cycle in G. Then $\psi \geq nw(C)$.*

Proof. Recall that $\psi = \sum_{v\in V} \psi_v$. By Lemma 2, we have that $\psi = \sum_{v\in V} w(I_v) \geq nw(C)$. ◀

▶ **Lemma 9.** $(1 - \frac{1}{2}\gamma + a\gamma)\psi \geq (\frac{1}{2} + a)\gamma\psi + b(1-\gamma)\psi \geq \Delta$.

Proof. By the definitions of γ, a and b, we know that $(1 - \frac{1}{2}\gamma + a\gamma)\psi = \sum_{v\in V}(1 - \frac{1}{2}\gamma_v + a_v\gamma_v)w(I_v)$ and $(\frac{1}{2} + a)\gamma\psi + b(1-\gamma)\psi = \sum_{v\in V}((\frac{1}{2} + a_v)\gamma_v w(I_v) + b_v(1-\gamma_v)w(I_v))$.

Recall that $\Delta = \sum_{v\in V} \deg(v)$. By Lemma 4, it holds that $\sum_{v\in V}(1 - \frac{1}{2}\gamma_v + a_v\gamma_v)w(I_v) \geq \sum_{v\in V}((\frac{1}{2} + a_v)\gamma_v w(I_v) + b_v(1-\gamma_v)w(I_v)) \geq \sum_{v\in V} \deg(v) = \Delta$. Therefore, we have that $(1 - \frac{1}{2}\gamma + a\gamma)\psi \geq (\frac{1}{2} + a)\gamma\psi + b(1-\gamma)\psi \geq \Delta$. ◀

We note that $(1 - \frac{1}{2}\gamma + a\gamma) \leq \frac{3}{2}$. Thus, we have that $\Delta = O(1)\psi$.

▶ **Lemma 10.** $(1 - \frac{1}{2}a)\gamma\psi + (1 - \frac{1}{2}b)(1-\gamma)\psi \geq nMST(G)$.

Proof. Note that $(1 - \frac{1}{2}a)\gamma\psi + (1 - \frac{1}{2}b)(1-\gamma)\psi = \sum_{v\in V}((1 - \frac{1}{2}a_v)\gamma_v w(I_v) + (1 - \frac{1}{2}b_v)(1-\gamma_v)w(I_v))$.

By Lemma 5, we know that $(1 - \frac{1}{2}a)\gamma\psi + (1 - \frac{1}{2}b)(1-\gamma)\psi = \sum_{v\in V}((1 - \frac{1}{2}a_v)\gamma_v w(I_v) + (1 - \frac{1}{2}b_v)(1-\gamma_v)w(I_v)) \geq n\mathrm{MST}(G)$. ◀

By Lemmas 9 and 10, we have that

$$(1 + \tfrac{1}{4}\gamma)\psi \geq \tfrac{1}{2}\Delta + n\mathrm{MST}(G). \tag{2}$$

▶ **Lemma 11.** *Let M be a minimum weight perfect matching in G. Then $\frac{1}{2}\gamma\psi + (1-b)(1-\gamma)\psi \geq nw(M)$.*

Proof. Note that $\frac{1}{2}\gamma\psi + (1-b)(1-\gamma)\psi = \sum_{v\in V}(\frac{1}{2}\gamma_v w(I_v) + (1-b_v)(1-\gamma_v)w(I_v))$.

By Lemma 6, we know that $\frac{1}{2}\gamma\psi + (1-b)(1-\gamma)\psi = \sum_{v\in V}(\frac{1}{2}\gamma_v w(I_v) + (1-b_v)(1-\gamma_v)w(I_v)) \geq nw(M)$. ◀

By Lemmas 9 and 11, we have that

$$(1 + a\gamma)\psi \geq \Delta + nw(M) \tag{3}$$

for a minimum weight matching M.

▶ **Lemma 12.** *For a graph G, let P^* be a minimum weight P_3 packing, and T^* be a minimum weight triangle packing. Then $(\frac{2}{3} + \frac{1}{3}\gamma - a\gamma)\psi = (1-a)\gamma\psi + \frac{2}{3}(1-\gamma)\psi \geq nw(P^*) \geq \frac{1}{2}nw(T^*)$.*

Proof. Note that $(\frac{2}{3} + \frac{1}{3}\gamma - a\gamma)\psi = \sum_{v\in V}(\frac{2}{3} + \frac{1}{3}\gamma_v - a_v\gamma_v)w(I_v)$ and $(1-a)\gamma\psi + \frac{2}{3}(1-\gamma)\psi = \sum_{v\in V}((1-a_v)\gamma_v w(I_v) + \frac{2}{3}(1-\gamma_v)w(I_v))$.

By Lemma 7, it holds that $\sum_{v\in V}(\frac{2}{3} + \frac{1}{3}\gamma_v - a_v\gamma_v)w(I_v) \geq \sum_{v\in V}((1-a_v)\gamma_v w(I_v) + \frac{2}{3}(1-\gamma_v)w(I_v)) \geq nw(P^*) \geq \frac{1}{2}nw(T^*)$. Therefore, we have that $(\frac{2}{3} + \frac{1}{3}\gamma - a\gamma)\psi = (1-a)\gamma\psi + \frac{2}{3}(1-\gamma)\psi \geq nw(P^*) \geq \frac{1}{2}nw(T^*)$. ◀

Next, we will describe our algorithm. Our algorithm consists of two constructions, where the first is based on the Hamiltonian cycle and the second is based on the triangle packing. The approximation quality of each will be analyzed after showing the construction. Finally, we will make a trade-off between them and get the final approximation ratio.

3.2 The Hamiltonian Cycle Construction

In our algorithm, the idea of Hamiltonian cycle construction is to make use of the canonical schedule [19, 8] and a Hamiltonian cycle. For TTP-k, there are many approximation algorithms using this method [29, 27, 18], and hence we will directly use the well-analyzed schedule in [29]. However, we will give a tighter analysis. Next, we give a brief introduction to this construction.

Roughly speaking, this schedule is generated by a rotation scheme that can make sure that almost all road trips of each team t_i are 4-cycles and in each road trip, team t_i visits a set of consecutive teams along the Hamiltonian cycle.

▶ **Lemma 13** ([29]). *Let C be a Hamiltonian cycle in graph G. For TTP-3, there is a polynomial-time algorithm that can generate a solution with a total weight of at most $\frac{2}{3}nw(C) + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi$.*

Note that the Hamiltonian cycle used in [29] is generated by the Christofides-Serdyukov algorithm. In our algorithm, we also consider another Hamiltonian cycle that uses the minimum weight perfect matching. We will select the better one between these two cycles.

▶ **Lemma 14** (*). *Let M be a perfect matching in graph G. Then there is a polynomial-time algorithm that can generate a Hamiltonian cycle C such that $w(C) = w(M) + \frac{1}{n}w(E) + O(\frac{1}{n^2})w(E)$.*

Note that Lemma 14 holds for any perfect matching. We will consider the Hamiltonian cycle that uses a minimum weight perfect matching.

▶ **Theorem 15.** *For any $\varepsilon > 0$, there is a polynomial-time algorithm that can generate a feasible schedule for TTP-3 with an approximation ratio of $\min\{\frac{4}{3} + \frac{1}{3}a\gamma,\ 1 - \frac{1}{6}\gamma + a\gamma\} + \varepsilon$.*

Proof. Here we use C to denote a minimum weight Hamiltonian cycle in G. If we use the Hamiltonian cycle C' obtained by the Christofides-Serdyukov algorithm, we can construct a feasible schedule with a total weight of at most $\frac{2}{3}nw(C') + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi$ by Lemma 13. Then, we have that

$$\begin{aligned}\frac{2}{3}nw(C') + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi &\leq \frac{2}{3}n\left(\mathrm{MST}(G) + \frac{1}{2}w(C)\right) + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi\\ &= \frac{2}{3}n\mathrm{MST}(G) + \frac{1}{3}nw(C) + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi\\ &\leq (\frac{4}{3} + \frac{1}{3}a\gamma)\psi + O(\frac{1}{n})\psi,\end{aligned}$$

where the first inequality follows from Lemma 3 and the second follows from Lemmas 8 and 9, and (2).

Similarly, if we use a minimum weight perfect matching M to obtain the Hamiltonian cycle C_M in Lemma 14, we can construct a feasible schedule with a total weight of at most

$$\begin{aligned}\frac{2}{3}nw(C_M) + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi &\leq \frac{2}{3}n\left(w(M) + \frac{1}{n}w(E) + O(\frac{1}{n^2})w(E)\right) + \frac{2}{3}\mathit{\Delta} + O(\frac{1}{n})\psi\\ &\leq \frac{2}{3}nw(M) + \mathit{\Delta} + O(\frac{1}{n})\psi\\ &\leq (1 - \frac{1}{6}\gamma + a\gamma)\psi + O(\frac{1}{n})\psi,\end{aligned}$$

where the first inequality follows from Lemma 14, the second follows from $w(E) = \frac{1}{2}\mathit{\Delta} \leq O(1)\psi$, and the last follows from Lemma 9 and (3).

Since we select the better one between these two Hamiltonian cycles, then the approximation ratio is $\min\{\frac{4}{3}+\frac{1}{3}a\gamma,\ 1-\frac{1}{6}\gamma+a\gamma\}+O(\frac{1}{n})$. Hence, there exists a constant c such that the ratio is bounded by $\min\{\frac{4}{3}+\frac{1}{3}a\gamma,\ 1-\frac{1}{6}\gamma+a\gamma\}+\frac{c}{n}$. For an arbitrary $\varepsilon>0$, if $n\leq\frac{c}{\varepsilon}=O(1)$, we can find an optimal solution by brute force, otherwise we use the approximation algorithm. This establishes the approximation ratio of $\min\{\frac{4}{3}+\frac{1}{3}a\gamma,\ 1-\frac{1}{6}\gamma+a\gamma\}+\varepsilon$. ◀

Note that $\max_{0\leq a,\gamma\leq 1}\min\{\frac{4}{3}+\frac{1}{3}a\gamma,\ 1-\frac{1}{6}\gamma+a\gamma\}+\varepsilon$ is maximized when $a=\gamma=1$ with value $(\frac{5}{3}+\varepsilon)$. However, this would require $0=(\frac{2}{3}+\frac{1}{3}\gamma-a\gamma)\psi\geq nw(P^*)\geq\frac{1}{2}nw(T^*)$ by Lemma 12. If we use any constant ratio approximation algorithm for minimum weight P_3 path packing or minimum weight triangle packing, we may get a much better schedule based on them and therefore, we will show that we can do better than $(\frac{5}{3}+\varepsilon)$ by combining the Hamiltonian cycle construction with the triangle packing construction shown next.

3.3 The Triangle Packing and P_3 Path Packing Constructions

In this section, we will construct a feasible schedule based on a triangle packing or a P_3 path packing. The idea is similar to the packing schedule based on a minimum weight perfect matching for TTP-2 [25, 28, 30, 31, 17]. Given a triangle packing (resp., a P_3 path packing), we consider the three normal teams in a triangle (resp., a P_3 path) as a super-team. The packing construction is to first arrange a single round-robin for super-teams and then extend the single round-robin into a double round-robin for normal teams. Although the construction is similar for a given triangle packing and a given P_3 packing, the analysis and the approximation ratio may be different.

3.3.1 Construction

First, we will introduce the single round-robin of super-teams.

Given a triangle packing T (resp., P_3 path packing P) of G, recall that we take the three teams in each triangle (resp., P_3 path) as a super-team. There are n normal teams and then there are $m=\frac{n}{3}$ super-teams. The set of super-teams is $U=\{u_1,u_2,\dots,u_{m-1},u_m\}$. We relabel the n teams such that $u_i=\{t_{3i-2},t_{3i-1},t_{3i}\}$ for each i.

In the construction, the case of $n=6$ is easy, and hence we assume here that $n\geq 12$. Each super-team will attend $m-1$ super-games in $m-1$ time slots. Each super-game in the first $m-2$ time slots will be extended to normal games that span six days, and each super-game in the last time slot will be extended to normal games that span ten days. Therefore, we have $6\times(m-2)+10=6m-2=2n-2$ days of normal games in total. This is the number of days in a double round-robin. We will construct the schedule for super-teams from the first time slot to the last time slot $m-1$. In each of the $m-1$ time slots, we have $\frac{m}{2}$ super-games. The schedule in the last time slot is different from the schedules in the first $m-2$ time slots.

For the first time slot, the $\frac{m}{2}$ super-games are arranged as shown in Figure 1. The super-game including super-team u_m is called *left super-game* and we put a letter 'L' on the edge to indicate it. All other super-games are called *normal super-games*. Each super-game is represented by a directed edge, where a directed edge from u_i to u_j means a super-game between them at the home of u_j. The information of which will be used to extend super-games to normal games between normal teams.

In Figure 1, we can see that the last super-team u_m is denoted as a dark node and all other super-teams $u_1,\dots,u_{m-1}$ are denoted as white nodes which form a cycle. In the second time slot, we keep the position of u_m and change the positions of white super-teams in the cycle by moving one position in the clockwise direction, and we also change the direction of each edge. In the second time slot, there are still $\frac{m}{2}-1$ normal super-games and one left super-game.

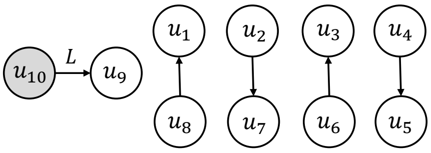


Figure 1 The super-game schedule in the first time slot for an instance with $m = 10$.

Table 1 Extending normal super-games where home games are marked in bold.

	$6q-5$	$6q-4$	$6q-3$	$6q-2$	$6q-1$	$6q$
a	x	z	y	$\boldsymbol{x}$	$\boldsymbol{z}$	$\boldsymbol{y}$
b	y	x	z	$\boldsymbol{y}$	$\boldsymbol{x}$	$\boldsymbol{z}$
c	z	y	x	$\boldsymbol{z}$	$\boldsymbol{y}$	$\boldsymbol{x}$
x	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{c}$	a	b	c
y	$\boldsymbol{b}$	$\boldsymbol{c}$	$\boldsymbol{a}$	b	c	a
z	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	c	a	b

Table 2 Extending left super-games where home games are marked in bold.

	$6q-5$	$6q-4$	$6q-3$	$6q-2$	$6q-1$	$6q$
a	x	z	$\boldsymbol{y}$	$\boldsymbol{x}$	$\boldsymbol{z}$	y
b	y	x	$\boldsymbol{z}$	$\boldsymbol{y}$	$\boldsymbol{x}$	z
c	z	y	$\boldsymbol{x}$	$\boldsymbol{z}$	$\boldsymbol{y}$	x
x	$\boldsymbol{a}$	$\boldsymbol{b}$	c	a	b	$\boldsymbol{c}$
y	$\boldsymbol{b}$	$\boldsymbol{c}$	a	b	c	$\boldsymbol{a}$
z	$\boldsymbol{c}$	$\boldsymbol{a}$	b	c	a	$\boldsymbol{b}$

The schedules for the other middle slots are derived analogously. Before we introduce the super-games in the last time slot, we first explain how to extend the super-games in the first $m-2$ time slots to normal games. In these time slots, we have two different kinds of super-games: normal super-games and left super-games. We first consider normal super-games.

Case 1. Normal super-games: Each normal super-game will be extended to eighteen normal games in six days. Assume that in a normal super-game, super-team u_i plays against the super-team u_j at the home of u_j in time slot q ($1 \leq i, j \leq m$ and $1 \leq q \leq m-1$). For ease of presentation, we let $u_i = \{t_{3i-2}, t_{3i-1}, t_{3i}\} = \{a, b, c\}$ and $u_j = \{t_{3j-2}, t_{3j-1}, t_{3j}\} = \{x, y, z\}$. The super-game will be extended to eighteen normal games in six corresponding days from $6q-5$ to $6q$, as shown in Table 1 where home games are marked in bold.

Case 2. Left super-games: Each left super-game will be extended to eighteen normal games in six days. Assume that in a left super-game, super-team $u_i = \{a, b, c\}$ plays against super-team $u_j = \{x, y, z\}$ at the home of u_j in time slot q ($2 \leq i \leq m-1$ and $1 \leq q \leq m-2$). The super-game will be extended to normal games in six corresponding days from $6q-5$ to $6q$, as shown in Table 2.

The first $m-2$ time slots will be extended to $6 \times (m-2) = 2n-12$ days according to the above rules. Each normal team will have ten remaining games, which correspond to the super-games in the last time slot. Figure 2 shows the schedule in the last time slot. All super-games in the last time slot are *last super-game* where we put a letter 'Z' to indicate it.

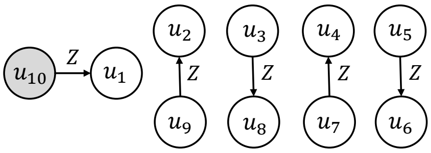


Figure 2 The super-game schedule in the last time slot for an instance with $m = 10$.

Table 3 Extending last super-games where home games are marked in bold.

	$6q-5$	$6q-4$	$6q-3$	$6q-2$	$6q-1$	$6q$	$6q+1$	$6q+2$	$6q+3$	$6q+4$
a	x	z	y	$\boldsymbol{x}$	$\boldsymbol{z}$	$\boldsymbol{y}$	c	b	$\boldsymbol{c}$	$\boldsymbol{b}$
b	y	x	c	$\boldsymbol{y}$	$\boldsymbol{x}$	$\boldsymbol{c}$	z	$\boldsymbol{a}$	$\boldsymbol{z}$	a
c	z	y	$\boldsymbol{b}$	$\boldsymbol{z}$	$\boldsymbol{y}$	b	$\boldsymbol{a}$	$\boldsymbol{x}$	a	x
x	$\boldsymbol{a}$	$\boldsymbol{b}$	z	a	b	$\boldsymbol{z}$	y	c	$\boldsymbol{y}$	$\boldsymbol{c}$
y	$\boldsymbol{b}$	$\boldsymbol{c}$	$\boldsymbol{a}$	b	c	a	$\boldsymbol{x}$	z	x	$\boldsymbol{z}$
z	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{x}$	c	a	x	$\boldsymbol{b}$	$\boldsymbol{y}$	b	y

Case 3. Last super-games: Each normal super-game will be extended to thirty normal games in ten days. Assume that in the last time slot $q = m - 1$, super-team $u_i = \{a, b, c\}$ plays against super-team $u_j = \{x, y, z\}$ $(1 \leq i, j \leq m)$ at the home of u_j. The super-game will be extended to thirty normal games in ten corresponding days from $6q - 5$ to $6q + 4$, as shown in Table 3.

▶ **Theorem 16** **(*).** *For TTP-3 with n teams such that $n \equiv 0 \pmod 6$, the above construction can generate a feasible schedule.*

3.3.2 Analyzing the Approximation Quality

Note that all teams play three consecutive away games and three consecutive home games in a normal super-game. Indeed, all teams are at home before and after each normal super-game in the schedule, otherwise, it will break the bounded-by-3 property. Using this property, we can get the total cost of normal super-games by analyzing each normal super-game separately.

▶ **Lemma 17** **(*).** *If there is a normal super-game between super-teams u_i and u_j at the home of u_j, then the cost of all normal teams in u_i and u_j is at most $\frac{4}{3}c(u_i, u_j) + 2c(u_i) + 2c(u_j)$.*

To analyze the total weight of our schedule, we first analyze the total cost of normal super-games. Recall that there is exactly one super-game between each pair of super-teams in U and then, there are $\frac{m(m-1)}{2}$ super-games in total. We define $R(u_i, u_j) = 1$ if the super-game between super-teams u_i and u_j is a normal super-game, and $R(u_i, u_j) = 0$ otherwise. By Lemma 17 and (1), we know that the total cost of all normal super-games E_0 satisfies that

$$\begin{array}{rcl} E_0 & \leq & \sum_{1 \leq i < j \leq m} \left(\frac{4}{3}c(u_i, u_j) + 2c(u_i) + 2c(u_j)\right) R(u_i, u_j) \\ & \leq & \sum_{1 \leq i < j \leq m} \left(\frac{4}{3}c(u_i, u_j) + 2c(u_i) + 2c(u_j)\right) \\ & = & \frac{4}{3}c(F) + 2(m-1)w(T) \\ & \leq & \frac{2}{3}\Delta + \frac{2}{3}nw(T). \end{array} \tag{4}$$

The total cost of all left super-games and all last super-games are denoted by E_1 and E_2, respectively.

▶ **Lemma 18** **(*).** *By using $O(n^3)$ time to reorder all super-teams, we can make*

$$E_1 + E_2 = O(\tfrac{1}{n})\psi.$$

Based on Lemma 18 and (4), we can get

▶ **Theorem 19.** *For TTP-3 with the case of $n \equiv 0 \pmod 6$, suppose there exists a triangle packing T in graph G, then there is a polynomial-time algorithm to generate a feasible schedule with a total weight of at most $\frac{2}{3}\Delta + \frac{2}{3}nw(T) + O(\frac{1}{n})\psi$.*

▶ **Theorem 20.** *Suppose there exist ρ_t and ρ_p approximation algorithms for minimum weight triangle packing and P_3 path packing problems respectively, then for an arbitrary $\varepsilon > 0$, there is a polynomial-time algorithm for TTP-3 with the case of $n \equiv 0 \pmod 6$, whose approximation ratio is $\frac{8\rho+6}{9} + \frac{4\rho-3}{9}\gamma - \frac{4\rho-2}{3}a\gamma + \varepsilon$, where $\rho = \min\{\rho_p, \rho_t\}$.*

Proof. First, we consider that the schedule uses a triangle packing T of G. By Theorem 19, we know that the total weight is bounded by $\frac{2}{3}\Delta + \frac{2}{3}nw(T) + O(\frac{1}{n})\psi$. Suppose the triangle packing T is obtained by using a ρ_t-approximation algorithm, i.e., $w(T) \leq \rho_t w(T^*)$, then by Lemmas 9 and 12, we have that

$$\begin{aligned}\frac{2}{3}\Delta + \frac{2}{3}nw(T) + O(\frac{1}{n})\psi &\leq \frac{2}{3}\Delta + \frac{2}{3}\rho_t nw(T^*) + O(\frac{1}{n})\psi\\ &\leq \frac{2}{3}\Delta + \frac{4}{3}\rho_t nw(P^*) + O(\frac{1}{n})\psi\\ &\leq (\frac{8\rho_t+6}{9} + \frac{4\rho_t-3}{9}\gamma - \frac{4\rho_t-2}{3}a\gamma)\psi + O(\frac{1}{n})\psi.\end{aligned}$$

Then, we consider that the schedule uses a P_3 path packing P of G. Suppose the P_3 path packing P is obtained by using a ρ_p-approximation algorithm, i.e., $w(P) \leq \rho_p w(P^*)$, then we can get a triangle packing T' by completing the P_3 path packing P such that $w(T') \leq 2w(P) \leq 2\rho_p w(P^*)$ by the triangle inequality. Thus, by Lemmas 9 and 12, the total weight of our schedule is bounded by

$$\begin{aligned}\frac{2}{3}\Delta + \frac{2}{3}nw(T') + O(\frac{1}{n})\psi &\leq \frac{2}{3}\Delta + \frac{4}{3}\rho_p nw(P^*) + O(\frac{1}{n})\psi\\ &\leq (\frac{8\rho_p+6}{9} + \frac{4\rho_p-3}{9}\gamma - \frac{4\rho_p-2}{3}a\gamma)\psi + O(\frac{1}{n})\psi.\end{aligned}$$

By selecting the better schedule, we can get the approximation ratio of $\frac{8\rho+6}{9} + \frac{4\rho-3}{9}\gamma - \frac{4\rho-2}{3}a\gamma + \varepsilon$, where $\rho = \min\{\rho_p, \rho_t\}$. Note that the algorithm of our triangle packing construction runs in polynomial time since it takes $O(n^3)$ time to reorder all super-teams and $O(n^2)$ time to construct the schedule. ◀

3.4 Trade-off between Two Constructions

▶ **Theorem 21 (*).** *Suppose there exist ρ_t and ρ_p approximation algorithms for minimum weight triangle packing and P_3 path packing problems respectively, for TTP-3 with an arbitrary $\varepsilon > 0$, there is an approximation algorithm whose ratio is $\frac{11}{6} - \frac{5}{2(4\rho+1)} + \varepsilon$ when $\rho \leq \frac{9}{4}$, and $\frac{5}{3} - \frac{2}{3(4\rho-1)} + \varepsilon$ otherwise, where $\rho = \min\{\rho_t, \rho_p\}$. Given $\rho = \frac{8}{3}$, the approximation ratio is $\frac{139}{87} + \varepsilon < 1.598 + \varepsilon$ which improves the previous ratio of $\frac{5}{3} + \varepsilon < 1.667 + \varepsilon$.*

Proof. For the case of $n \equiv 0 \pmod 6$, our algorithm will select the better schedule from the two constructions. By Theorem 15, the ratio of the Hamiltonian cycle construction is $\min\{\frac{4}{3} + \frac{1}{3}a\gamma,\ 1 - \frac{1}{6}\gamma + a\gamma\} + \varepsilon$, and by Theorem 20, the ratio of the triangle packing construction is $\frac{8\rho+6}{9} + \frac{4\rho-3}{9}\gamma - \frac{4\rho-2}{3}a\gamma + \varepsilon$. Therefore, the ratio of our algorithm in the worst case is

$$\max_{0 \leq a, \gamma \leq 1} \min\left\{\frac{4}{3} + \frac{1}{3}a\gamma,\ 1 - \frac{1}{6}\gamma + a\gamma,\ \frac{8\rho+6}{9} + \frac{4\rho-3}{9}\gamma - \frac{4\rho-2}{3}a\gamma\right\} + \varepsilon.$$

We can transform it into the following linear programming where we let $\widetilde{\gamma} = a\gamma$.

$$\begin{aligned}
\max\quad & y \\
\text{s.t.}\quad & y \le \frac{4}{3} + \frac{1}{3}\widetilde{\gamma}, \\
& y \le 1 - \frac{1}{6}\gamma + \widetilde{\gamma}, \\
& y \le \frac{8\rho+6}{9} + \frac{4\rho-3}{9}\gamma - \frac{4\rho-2}{3}\widetilde{\gamma}, \\
& 0 \le \widetilde{\gamma} \le \gamma \le 1.
\end{aligned}$$

It shows that the ratio is $\frac{11}{6} - \frac{5}{2(4\rho+1)} + \varepsilon$ when $\rho \le \frac{9}{4}$, and $\frac{5}{3} - \frac{2}{3(4\rho-1)} + \varepsilon$ otherwise.

To our best knowledge, both of the current best-known ratios for minimum weight triangle packing and minimum weight P_3 path packing problems are $\frac{8}{3}$ [12]. Therefore, given $\rho = \rho_p = \rho_t = \frac{8}{3}$, the final approximation ratio of our algorithm is $\frac{139}{87} + \varepsilon < 1.598 + \varepsilon$.

Note that the cases $n \equiv 2 \pmod 6$ and $n \equiv 4 \pmod 6$ have not been analyzed yet. Since n is not divisible by 3 for these two cases, there is no perfect P_3 path packing or triangle packing in G. However, with some modifications, we can extend the second construction and its analysis to get the same approximation ratio. ◀

By Theorem 21, we know that the approximation ratio can achieve $(4/3 + \varepsilon)$ if $\rho = 1$.

▶ **Corollary 22.** *If a minimum weight triangle packing or P_3 path packing is given, there exists a $(4/3 + \varepsilon)$-approximation algorithm for TTP-3, for an arbitrary $\varepsilon > 0$.*

4 LDTTP-3

When all teams are located on a straight line, this problem is known as Linear Distance TTP-3 (LDTTP-3) [15]. For this problem, the minimum weight triangle packing and P_3 path packing can be found in polynomial time. Thus, for LDTTP-3, our algorithm has an approximation ratio of $(4/3 + \varepsilon)$, which also matches the current best-known ratio of 4/3 (however, note that their construction only works for the case of $n \equiv 4 \pmod 6$) [15]. Their ratio is based on a stronger lower bound of LDTTP-3. If we use this lower bound, the approximation ratio of our triangle packing construction can be proved to be $(6/5 + \varepsilon)$.

▶ **Theorem 23 (*).** *For LDTTP-3, there is an approximation algorithm whose ratio is $(6/5 + \varepsilon)$, for an arbitrary $\varepsilon > 0$.*

5 TTP-4

It is natural to use the same idea to solve TTP-4. For the problem of minimum weight P_4 path packing, to our best knowledge, the current best-known ratio is 3/2 [22]. If we use the construction based on the Hamiltonian cycle and the construction based on P_4 path packing, we can get a $(17/10 + \varepsilon)$-approximation algorithm for TTP-4 which improves the previous approximation ratio of $(7/4 + \varepsilon)$ [29].

▶ **Theorem 24 (*).** *For TTP-4 with any $\varepsilon > 0$, there is an algorithm whose approximation ratio is $(17/10 + \varepsilon)$ which improves the previous approximation ratio of $(7/4 + \varepsilon)$.*

For TTP-k with $k \ge 5$, we note that both of the best-known approximation ratios for minimum weight k-cycle packing and P_k path packing are $4(1 - 1/k) > 3$ [12]. The approximation ratios are too large and we can not improve TTP-k by using the same idea.

References

1 Aris Anagnostopoulos, Laurent Michel, Pascal Van Hentenryck, and Yannis Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

2 Rishiraj Bhattacharyya. Complexity of the unconstrained traveling tournament problem. *Operations Research Letters*, 44(5):649–654, 2016.

3 David Van Bulck, Dries R. Goossens, Jörn Schönberger, and Mario Guajardo. Robinx: A three-field classification and unified data format for round-robin sports timetabling. *Eur. J. Oper. Res.*, 280(2):568–580, 2020.

4 Robert Thomas Campbell and Der-San Chen. A minimum distance basketball scheduling problem. *Management science in sports*, 4:15–26, 1976.

5 Diptendu Chatterjee. Complexity of traveling tournament problem with trip length more than three. *CoRR*, abs/2110.02300, 2021. `arXiv:2110.02300`.

6 Diptendu Chatterjee and Bimal Kumar Roy. An improved scheduling algorithm for traveling tournament problem with maximum trip length two. In *ATMOS 2021*, volume 96, pages 16:1–16:15, 2021.

7 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

8 Dominique De Werra. Scheduling in sports. *Studies on graphs and discrete programming*, 11:381–395, 1981.

9 Luca Di Gaspero and Andrea Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13(2):189–207, 2007.

10 Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem: description and benchmarks. In *7th International Conference on Principles and Practice of Constraint Programming*, pages 580–584, 2001.

11 Kelly Easton, George Nemhauser, and Michael Trick. Solving the travelling tournament problem: a combined integer programming and constraint programming approach. In *4th International Conference of Practice and Theory of Automated Timetabling IV*, pages 100–109, 2003.

12 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

13 Marc Goerigk, Richard Hoshino, Ken Kawarabayashi, and Stephan Westphal. Solving the traveling tournament problem by packing three-vertex paths. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2271–2277, 2014.

14 Marc Goerigk and Stephan Westphal. A combined local search and integer programming approach to the traveling tournament problem. *Ann. Oper. Res.*, 239(1):343–354, 2016.

15 Richard Hoshino and Ken-ichi Kawarabayashi. Generating approximate solutions to the TTP using a linear distance relaxation. *J. Artif. Intell. Res.*, 45:257–286, 2012.

16 Richard Hoshino and Ken-ichi Kawarabayashi. An approximation algorithm for the bipartite traveling tournament problem. *Mathematics of Operations Research*, 38(4):720–728, 2013.

17 Shinji Imahori. A 1+O(1/N) approximation algorithm for TTP(2). *CoRR*, abs/2108.08444, 2021. `arXiv:2108.08444`.

18 Shinji Imahori, Tomomi Matsui, and Ryuhei Miyashiro. A 2.75-approximation algorithm for the unconstrained traveling tournament problem. *Annals of Operations Research*, 218(1):237–247, 2014.

19 Thomas P Kirkman. On a problem in combinations. *Cambridge and Dublin Mathematical Journal*, 2:191–204, 1847.

20 Andrew Lim, Brian Rodrigues, and Xingwen Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research*, 174(3):1459–1478, 2006.

21 Ryuhei Miyashiro, Tomomi Matsui, and Shinji Imahori. An approximation algorithm for the traveling tournament problem. *Annals of Operations Research*, 194(1):317–324, 2012.

22 Jérôme Monnot and Sophie Toulouse. Approximation results for the weighted p_4 partition problem. *J. Discrete Algorithms*, 6(2):299–312, 2008.

23 Anatolii Ivanovich Serdyukov. Some extremal bypasses in graphs. *Upravlyaemye Sistemy*, 17:76–79, 1978.

24 Clemens Thielen and Stephan Westphal. Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4):345–351, 2011.

25 Clemens Thielen and Stephan Westphal. Approximation algorithms for TTP(2). *Mathematical Methods of Operations Research*, 76(1):1–20, 2012.

26 Michael Trick. Challenge traveling tournament instances. Accessed: 2022-4-01, 2022.

27 Stephan Westphal and Karl Noparlik. A 5.875-approximation for the traveling tournament problem. *Annals of Operations Research*, 218(1):347–360, 2014.

28 Mingyu Xiao and Shaowei Kou. An improved approximation algorithm for the traveling tournament problem with maximum trip length two. In *MFCS 2016*, volume 58, pages 89:1–89:14, 2016.

29 Daisuke Yamaguchi, Shinji Imahori, Ryuhei Miyashiro, and Tomomi Matsui. An improved approximation algorithm for the traveling tournament problem. *Algorithmica*, 61(4):1077–1091, 2011.

30 Jingyang Zhao and Mingyu Xiao. A further improvement on approximating TTP-2. In *COCOON 2021*, volume 13025 of *Lecture Notes in Computer Science*, pages 137–149. Springer, 2021.

31 Jingyang Zhao and Mingyu Xiao. The traveling tournament problem with maximum tour length two: A practical algorithm with an improved approximation bound. In *IJCAI 2021*, pages 4206–4212, 2021.