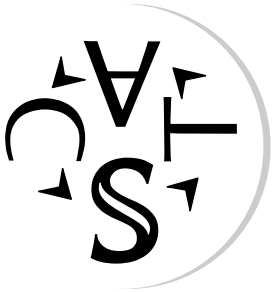
36th International Symposium on Theoretical Aspects of Computer Science

STACS 2019, March 13–16, 2019, Berlin, Germany

Edited by

Rolf Niedermeier

Christophe Paul



Editors

Rolf Niedermeier
Algorithmics and Computational Complexity, Fakultät IV, TU Berlin, Berlin, Germany
rolf.niedermeier@tu-berlin.de

Christophe Paul
CNRS, Université de Montpellier, LIRMM, Montpellier, France
christophe.paul@lirmm.fr

ACM Classification 2012
Theory of computation → Models of computation; Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory; Theory of computation → Formal languages and automata theory; Theory of computation → Logic; Theory of computation → Parameterized complexity and exact algorithms

ISBN 978-3-95977-100-9

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/978-3-95977-100-9.

Publication date
March, 2019

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.STACS.2019.0

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

Foreword
Rolf Niedermeier and Christophe Paul 0:ix

Invited Talks

Computational Complexity and Partition Functions
Leslie Ann Goldberg 1:1–1:3

The Many Facets of String Transducers
Anca Muscholl and Gabriele Puppis 2:1–2:21

Algorithmic Data Science
Petra Mutzel 3:1–3:15

Tutorials

Fine-Grained Complexity Theory
Karl Bringmann 4:1–4:7

From Graph Theory to Network Science:
The Natural Emergence of Hyperbolicity
Tobias Friedrich 5:1–5:9

Regular Contributions

The Semialgebraic Orbit Problem
Shaull Almagor, Joël Ouaknine, and James Worrell 6:1–6:15

Best-Of-Two-Worlds Analysis of Online Search
Spyros Angelopoulos, Christoph Dürr, and Shendan Jin 7:1–7:17

Bipartite Diameter and Other Measures Under Translation
Boris Aronov, Omrit Filtser, Matthew J. Katz, and Khadijeh Sheikhan 8:1–8:14

Solving Simple Stochastic Games with Few Random Nodes Faster Using Bland's Rule
David Auger, Pierre Coucheney, and Yann Strozecki 9:1–9:16

Distributed Coloring of Graphs with an Optimal Number of Colors
Étienne Bamas and Louis Esperet 10:1–10:15

On the Descriptive Complexity of Color Coding
Max Bannach and Till Tantau 11:1–11:16

Bounding Quantum-Classical Separations for Classes of Nonlocal Games
Tom Bannink, Jop Briët, Harry Buhrman, Farrokh Labib, and Troy Lee 12:1–12:11

Token Sliding on Split Graphs
Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora 13:1–13:17

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Building Strategies *into* QBF Proofs
Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan 14:1–14:18

Tight Analysis of the Smartstart Algorithm for Online Dial-a-Ride on the Line
Alexander Birx and Yann Disser 15:1–15:17

Enumerating Minimal Dominating Sets in Triangle-Free Graphs
Marthe Bonamy, Oscar Defrain, Marc Heinrich, and Jean-Florent Raymond 16:1–16:12

Sparsification of Binary CSPs
Silvia Butti and Stanislav Živný 17:1–17:8

Tractable QBF by Knowledge Compilation
Florent Capelli and Stefan Mengel 18:1–18:16

A Tight Extremal Bound on the Lovász Cactus Number in Planar Graphs
Parinya Chalermsook, Andreas Schmid, and Sumedha Uniyal 19:1–19:14

Average-Case Completeness in Tag Systems
Matthew Cook and Turlough Neary 20:1–20:17

Pairwise Preferences in the Stable Marriage Problem
Ágnes Cseh and Attila Juhos 21:1–21:16

Closure Properties of Synchronized Relations
María Emilia Descotte, Diego Figueira, and Santiago Figueira 22:1–22:17

Resource-Bounded Kolmogorov Complexity Provides an Obstacle to Soficness of Multidimensional Shifts
Julien Destombes and Andrei Romashchenko 23:1–23:17

Constant-Time Retrieval with $O(\log m)$ Extra Bits
Martin Dietzfelbinger and Stefan Walzer 24:1–24:16

Complexity of the Steiner Network Problem with Respect to the Number of Terminals
Eduard Eiben, Dušan Knop, Fahad Panolan, and Ondřej Suchý 25:1–25:17

Space Lower Bounds for the Signal Detection Problem
Faith Ellen, Rati Gelashvili, Philipp Woelfel, and Leqi Zhu 26:1–26:13

Progressive Algorithms for Domination and Independence
Grzegorz Fabiański, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk . 27:1–27:16

Modification to Planarity is Fixed Parameter Tractable
Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos 28:1–28:17

Visibly Pushdown Languages over Sliding Windows
Moses Ganardi 29:1–29:17

Fast and Longest Rollercoasters
Paweł Gawrychowski, Florin Manea, and Radosław Serafin 30:1–30:17

Wealth Inequality and the Price of Anarchy
Kurtuluş Gemici, Elias Koutsoupias, Barnabé Monnot, Christos H. Papadimitriou, and Georgios Piliouras 31:1–31:16

Lean Tree-Cut Decompositions: Obstructions and Algorithms
Archontia C. Giannopoulou, O-joung Kwon, Jean-Florent Raymond, and Dimitrios M. Thilikos 32:1–32:14

Dispersing Obnoxious Facilities on a Graph
Alexander Grigoriev, Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger 33:1–33:11

Reachability in $O(\log n)$ Genus Graphs is in Unambiguous Logspace
Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari 34:1–34:13

Dominating Sets and Connected Dominating Sets in Dynamic Graphs
Niklas Hjuler, Giuseppe F. Italiano, Nikos Parotsidis, and David Saulpic 35:1–35:17

On Kernelization for Edge Dominating Set under Structural Parameters
Eva-Maria C. Hols and Stefan Kratsch 36:1–36:18

Compressed Decision Problems in Hyperbolic Groups
Derek Holt, Markus Lohrey, and Saul Schleimer 37:1–37:16

How to Secure Matchings Against Edge Failures
Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt 38:1–38:16

A Deterministic Polynomial Kernel for Odd Cycle Transversal and Vertex Multiway Cut in Planar Graphs
Bart M. P. Jansen, Marcin Pilipczuk, and Erik Jan van Leeuwen 39:1–39:18

A Characterization of Subshifts with Computable Language
Emmanuel Jeandel and Pascal Vanier 40:1–40:16

Lower Bounds for DeMorgan Circuits of Bounded Negation Width
Stasys Jukna and Andrzej Lingas 41:1–41:17

Depth First Search in the Semi-streaming Model
Shahbaz Khan and Shashank K. Mehta 42:1–42:16

On Finite Monoids over Nonnegative Integer Matrices and Short Killing Words
Stefan Kiefer and Corto Mascle 43:1–43:13

Tight Complexity Lower Bounds for Integer Linear Programming with Few Constraints
Dušan Knop, Michał Pilipczuk, and Marcin Wrochna 44:1–44:15

The Set Cover Conjecture and Subgraph Isomorphism with a Tree Pattern
Robert Krauthgamer and Ohad Trabelsi 45:1–45:15

Algorithmic Properties of Sparse Digraphs
Stephan Kreutzer, Irene Muzi, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz 46:1–46:20

Tree Automata with Global Constraints for Infinite Trees
Patrick Landwehr and Christof Löding 47:1–47:15

Constructive Discrepancy Minimization with Hereditary L2 Guarantees
Kasper Green Larsen 48:1–48:13

Quantum Advantage for the LOCAL Model in Distributed Computing
François Le Gall, Harumichi Nishimura, and Ansis Rosmanis 49:1–49:14

Lifting Theorems for Equality
Bruno Loff and Sagnik Mukhopadhyay 50:1–50:19

Car-Sharing on a Star Network: On-Line Scheduling with k Servers
Kelin Luo, Thomas Erlebach, and Yinfeng Xu 51:1–51:14

Beyond Boolean Surjective VCSPs
Gregor Matl and Stanislav Živný 52:1–52:15

The Containment Problem for Unambiguous Register Automata
Antoine Mottet and Karin Quaas 53:1–53:15

Stabilization Time in Weighted Minority Processes
Pál András Papp and Roger Wattenhofer 54:1–54:15

Finite Sequentiality of Unambiguous Max-Plus Tree Automata
Erik Paul 55:1–55:17

Paging with Dynamic Memory Capacity
Enoch Peserico 56:1–56:18

Random Noise Increases Kolmogorov Complexity and Hausdorff Dimension
Gleb Posobin and Alexander Shen 57:1–57:14

A Unified Approach to Tail Estimates for Randomized Incremental Construction
Sandeep Sen 58:1–58:16

A $ZPP^{NP[1]}$ Lifting Theorem
Thomas Watson 59:1–59:16

Foreword

The International Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an internationally leading forum for original research on theoretical aspects of computer science. Typical areas are:

- algorithms and data structures, including: design of parallel, distributed, approximation, parameterized and randomized algorithms; analysis of algorithms and combinatorics of data structures; computational geometry, cryptography, algorithmic learning theory, algorithmic game theory;
- automata and formal languages, including: algebraic and categorical methods, coding theory;
- complexity and computability, including: computational and structural complexity theory, parameterized complexity, randomness in computation;
- logic in computer science, including: finite model theory, database theory, semantics, specification verification, rewriting and deduction;
- current challenges, for example: natural computing, quantum computing, mobile and net computing, computational social choice.

STACS is held alternately in France and in Germany. This year's conference (taking place March 13–16 in Berlin) is the 36th in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), Lyon (2014), München (2015), Orléans (2016), Hannover (2017), Caen (2018).

The interest in STACS has remained at a very high level over the past years. The STACS 2019 call for papers led to 260 submissions with authors from 39 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. The committee selected 54 papers during a three-week electronic meeting held in November/December 2018. This means an acceptance rate of only 21 %. For the fifth time within the STACS conference series, there was also a rebuttal period during which authors could submit remarks to the PC concerning the reviews of their papers. As co-chairs of the program committee, we would like to sincerely thank all its members and 491 external reviewers for their valuable work. In particular, there were intense and interesting discussions inside the PC committee. The overall very high quality of the submissions made the selection an extremely difficult task. This year, the conference includes two invited tutorials. We would like to express our thanks to the speakers Karl Bringmann (Saarbrücken) and Tobias Friedrich (Potsdam) for their tutorials, as well as to the three invited speakers, Leslie Ann Goldberg (Oxford), Anca Muscholl (Bordeaux), and Petra Mutzel (Dortmund). Special thanks go to the local organizing committee for continuous help throughout the conference organization. In particular, we wish to thank Till Fluschnik for his day-to-day support on the technical side and Christlinde Thielcke for her permanent administrative support. We also thank André Nichterlein for his final assistance in producing the conference proceedings and TUBS GmbH for their handling of financial accounting.

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Moreover, we thank Michael Wagner from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorials. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series. STACS 2019 has received funds and help from the Deutsche Forschungsgemeinschaft (DFG) and support and help from the Technische Universität Berlin (TU Berlin), for which we are very grateful.

Berlin and Montpellier, March 2019 Rolf Niedermeier and Christophe Paul

Conference Organization

Program Committee

Christoph Berkholz	Humboldt-Universität zu Berlin
Benedikt Bollig	LSV & ENS Paris-Saclay, CNRS
Karl Bringmann	Max Planck Institute for Informatics, Saarbrücken
Gerth Stølting Brodal	Aarhus University
Maike Buchin	Ruhr-Universität Bochum
David Eppstein	University of California, Irvine
Serge Gaspers	UNSW Sydney and Data61, CSIRO
Edward Hirsch	St. Petersburg State University
Telikepalli Kavitha	Tata Institute of Fundamental Research, Mumbai
Hartmut Klauck	National University of Singapore
Antonín Kučera	Masaryk University, Brno
K Narayan Kumar	Chennai Mathematical Institute
Dietrich Kuske	TU Ilmenau
Jérôme Lang	CNRS, LAMSADE, Université Paris-Dauphine
Sophie Laplante	IRIF, Université Paris Diderot
Kazuhisa Makino	Kyoto University
Barnaby Martin	Durham University
Cyril Nicaud	Université Paris Est, LIGM
Rolf Niedermeier	TU Berlin (co-chair)
Jakob Nordström	KTH Royal Institute of Technology, Stockholm
Christophe Paul	LIRMM, CNRS, Université de Montpellier (co-chair)
Pascal Schweitzer	TU Kaiserslautern
Shinnosuke Seki	University of Electro-Communications, Tokyo
Michał Skrzypczak	University of Warsaw
Srikanth Srinivasan	Indian Institute of Technology Bombay, Mumbai
Jan Arne Telle	University of Bergen
Denis Trystram	University Grenoble Alpes
Takeaki Uno	National Institute of Informatics, Tokyo
Mikhail Volkov	Ural Federal University, Ekaterinburg
Stefan Woltran	TU Wien

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Local Organizing Committee (TU Berlin)

Matthias Bentert
Robert Bredereck
Till Fluschnik
Vincent Froese
Klaus Heeger
Anne-Sophie Himmel
Andrzej Kaczmarczyk
Dušan Knop
Leon Kellerhals
Hendrik Molter
André Nichterlein
Rolf Niedermeier (chair)
Malte Renken
Christlinde Thielcke
Philipp Zschoche

External Reviewers

Amir Abboud
Peyman Afshani
S. Akshay
Kazuyuki Amano
Bo An
Dmitry Ananichev
Eric Angel
Antonios Antoniadis
Tetsuya Araki
Elena Arseneva Khramtcova
James Aspnes
Albert Atserias
Nathalie Aubrun
David Avis
Haris Aziz
Maxim Babenko
Arturs Backurs
Nikhil Balaji
Jozsef Balogh
Hideo Bannai
Nikhil Bansal
Jérémy Barbay
Nicolas Basset
Julien Baste
Isabel Beckenbach
Djamal Belazzougui
Rémy Belmonte
Alexander Belov
Matthias Bentert
Cédric Bentz
Sebastian Berndt
Vincent Berry
Nathalie Bertrand
Dietmar Berwanger
Amey Bhangale
Umang Bhaskar
Sayan Bhattacharya
Laurent Bienvenu
Johanna Björklund
Thomas Bläsius
Bernhard Bliem
Ivan Bliznets
Achim Blumensath
Ilario Bonacina
Edouard Bonnet
Bartłomiej Bosek
Nicolas Bousquet
Cornelius Brand
Tomas Brazdil
Sabine Broda
Jonah Brown-Cohen
Guido Brückner
Binh-Minh Bui-Xuan
Christina Büsing
Michaël Cadilhac
Alan Cain
Florent Capelli
Francesco Caravelli
Arnaud Carayol
Marco Carmosino
Olivier Carton
André Chailloux
Sourav Chakraborty
Yi-Jun Chang
Steven Chaplick
Vaggos Chatziafratis
Jiehua Chen
Victor Chepoi
Danila Cherkashin
Ashish Chiplunkar
Dmitry Chistikov
Rajesh Chitnis
Ananya Christman
Vincent Cohen-Addad
Thomas Colcombet
Stefano Coniglio
Alessio Conte
Miguel Couceiro
Ágnes Cseh
Wojciech Czerwiński
Mattia D'Emidio
Konrad Kazimierz Dabrowski
Ugo Dal Lago
Peter Damaschke
Bireswar Das
Syamantak Das
Laure Daviaud
Ronald de Haan
Arnaud de Mesmay
Susanna F. de Rezende
Holger Dell
Dariusz Dereniowski
Stéphane Devismes
Benjamin Doerr
Francesco Dolce
Ran Duan
Devdatt Dubhashi
François Durand
Pierre-Francois Dutot
Wolfgang Dvořák
Attila Egri-Nagy
Eduard Eiben
Khaled Elbassioni
Murray Elder
Jan Elffers
Marek Eliáš
Elaine Eschen
Kousha Etessami
Angelo Fanelli
John Fearnley
Carl Feghali
Stefan Felsner
Asaf Ferber
Henning Fernau
Laurent Feuilloley
Johannes Fichte
Hendrik Fichtenberger
Aris Filos-Ratsikas
Arnold Filtser
Jorg Flum
Till Fluschnik
Jacob Focke
Sebastian Forster
Fabrizio Frati
Dominik D. Freydenberger
André Frochaux
Vincent Froese
Matthias Függer
Hiroshi Fujiwara
Eric Fusy
Ariel Gabizon
Jakub Gajarsky
Pietro Galliani
Philippe Gambette
Robert Ganian
Mohit Garg
Naveen Garg
Paul Gastin
Cyril Gavoille
Pawel Gawrychowski
Enrico Gerding
Panos Giannopoulos
Archontia Giannopoulou
Hugo Gilbert
Andreas Göbel
Stephan Gocht
Tomasz Gogacz

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Alexander Göke
Petr Golovach
Alexander Golovnev
Mika Göös
Gramoz Goranci
Daniel Graça
Erich Grädel
Fabrizio Grandoni
Kasper Green Larsen
Elena Grigorescu
Martin Grohe
Rohit Gurjar
Gregory Gutin
Marc Gyssens
Christoph Haase
Vesa Halava
Guangyue Han
Xin Han
Prahladh Harsha
Frédéric Havet
Markus Hecher
Loic Helouet
Petr Hlineny
Markus Holzer
Kaave Hosseini
Chien-Chung Huang
Pavel Hubáček
Tomohiro I
Rasmus Ibsen-Jensen
Shinji Imahori
Dmitry Itsykson
Yoichi Iwata
Lars Jaffke
Petr Jancar
Bart M. P. Jansen
Bruno Jartoux
Ismaël Jecker
Artur Jeż
Łukasz Jeż
Zhengfeng Ji
Matthew Johnson
Martin Jonas
Peter Jonsson
Gwenaël Joret
Vincent Jugé
Marcin Jurdzinski
Andrzej Kaczmarczyk
Naoyuki Kamiyama
Iyad Kanj
Adam Karczmarz
Jarkko Kari
Nikolai Karpov
Yoshiyuki Karuno
Steve Kass
Yasushi Kawase
Jens Keppeler
Thomas Kesselheim
Emanuel Kieronski
Benjamin Kiesl
Shuji Kijima
Eunjung Kim
Sang-Sub Kim
Kei Kimura
Tamás Király
Masashi Kiyomi
Joachim Klein
Kim-Manuel Klein
Alexander Knop
Dušan Knop
Yusuke Kobayashi
Eryk Kopczynski
Dominik Köppl
Arpita Korwar
Dmitry Kosolobov
Robin Kothari
Martin Koutecký
Yiannis Koutis
Lukasz Kowalik
Alexander Kozachinskiy
Jan Kretinsky
Amer Krivosija
Markus Kröll
Manfred Kufleitner
Alexander Kulikov
Amit Kumar
Michal Kunc
Marvin Künnemann
Kazuhiro Kurita
O-Joung Kwon
Sébastien Labbé
Arnaud Labourel
Guillaume Lagarde
Jean Marie Lagniez
Michael Lampis
John Lapinskas
Massimo Lauria
Mathieu Laurière
Tien-Nam Le
Francois Le Gall
Thierry Lecroq
Johannes Lengler
Pascal Lenzner
Avivit Levy
Nathan Lhote
Nutan Limaye
Stratis Limnios
Andrea Lincoln
Zhenming Liu
Christof Löding
Markus Lohrey
Satyanarayana Lokam
Daniel Lokshtanov
Sylvain Lombardy
Anna Lubiw
Giorgio Lucarelli
Junjie Luo
Ramanujan M. S.
Jeremy Macdonald
Alexander Mäcker
Meena Mahajan
Mohammad Mahmoody
Cécile Mailler
Adam Malinowski
Jan Maly
Florin Manea
David Manlove
Maurice Margenstern
Andrea Marino
Claire Mathieu
Samuel McCauley
Andrew McGregor
Kitty Meeks
Saeed Mehrabi
Or Meir
Stefan Mengel
Robert Mercas
Wolfgang Merkle
Friedhelm Meyer Auf der Heide
David Mezlaf
Mehdi Mhalla
Othon Michail
Vincent Michielini
Ivan Mikhajlin
Martin Milanič
Dor Minzer
Ilya Mironov
Neeldhara Misra
Valia Mitsou
Shuichi Miyazaki
Matthias Mnich
Xavier Molinero
Hendrik Molter
Tobias Mömke
Clement Mommessin
Nelma Moreira
Antoine Mottet
Amer Mouawad
Grégory Mounié
Aida Mousavifar

Sagnik Mukhopadhyay
Martin Mundhenk
Kengo Nakamura
Anand Natarajan
Jesper Nederlof
Volodymyr Nekrashevych
Jelani Nelson
Stefan Neumann
André Nichterlein
Matthias Niewerth
Sergey Nikolenko
Aleksandar Nikolov
Prajakta Nimbhorkar
Reino Niskanen
Alexandre Niveau
Alexandre Nolin
Petr Novotný
Jan Obdrzalek
Pascal Ochem
Joanna Ochremiak
Alexander Okhotin
Krzysztof Onak
Tim Oosterwijk
Sebastian Ordyniak
Magdalena Ortiz
Yota Otachi
Martin Otto
Shayan Oveis Gharan
Dana Pardubska
Paweł Parys
Ami Paz
Timothée Pecatte
Tomáš Peitl
Vianney Perchet
Simon Perdrix
Sylvain Perifel
Clément Pernet
Fedor Petrov
Elena Petrova
Giovanni Pighizzini
Michał Pilipczuk
G. Michele Pinna
Solon Pissis
Alexandru Popa
Natacha Portier
Igor Potapov
Anupam Prakash
M. Praveen
Nicola Prezza
Elena Pribavkina
Kirk Pruhs
Marcin Przybyłko
Gabriele Puppis
Svetlana Puzynina
Jaikumar Radhakrishnan
Jakub Radoszewski
Benjamin Raichel
Günther Raidl
Venkatesh Raman
Narad Rampersad
Michael Rao
Gaurav Rattan
Bhaskar Ray Chaudhury
Jean-Florent Raymond
Igor Razgon
Damien Regnault
Vojtech Rehak
Daniel Reichman
Felix Reidl
Jan Reimann
Fabian Reiter
Malte Renken
Anja Rey
Pierre-Alain Reynier
Gaétan Richard
David Richerby
Kilian Risse
Cristian Riveros
Robert Robere
Emanuele Rodaro
Jérémie Roland
Jonathan Rollin
Andrei Romashchenko
Adi Rosén
Raphael Rossignol
Thomas Rothvoss
Andrew Ryzhikov
Toshiki Saitoh
Prakash Saivasan
Andrej Sajenko
Ville Salo
Rahul Santhanam
Swagato Sanyal
Ramprasad Saptharishi
Thatchaphol Saranurak
Ignasi Sau
Thomas Sauerwald
David Saulpic
Nitin Saurabh
Saket Saurabh
Nadja Scharf
Kevin Schewior
Markus L. Schmid
Jens M. Schmidt
Nicole Schweikardt
Alexander Shen
Xiangkun Shen
Akiyoshi Shioura
Takeharu Shiraga
Arseny Shur
Sebastian Siebertz
Florian Sikora
Rodrigo Silveira
Sunil Simon
Friedrich Slivovsky
Alexander Smal
Christian Sohler
Dmitry Sokolov
Manuel Sorge
Jiri Srba
A V Sreejith
Abhinav Srivastav
Piyush Srivastava
K V Subrahmanyam
Zhaohong Sun
Anupa Sunny
Akira Suzuki
Joseph Swernofsky
Marek Sys
Kenjiro Takazawa
Navid Talebanfard
Suguru Tamaki
Shin-Ichi Tanigawa
Till Tantau
Alexandre Termier
Sumedh Tirodkar
Sophie Tison
Ioan Todinca
Meng-Tsung Tsai
Ryuhei Uehara
René van Bevern
Jan van den Brand
Rob van Stee
Pascal Vanier
Carmine Ventre
Nikolay Vereshchagin
Sergey Verlan
Marc Vinyals
Caterina Viola
Ben Lee Volk
Moritz von Looz
Magnus Wahlström
Haitao Wang
Kunihiro Wasa
Oren Weimann
Armin Weiss
Mathias Weller
Philip Wellnitz
Josef Widder
Piotr Więcek
Andreas Wiese
Ryan Williams
Sarah Winter
Gerhard J. Woeginger
Dominik Wojtczak

Marcin Wrochna
Mingyu Xiao
Chao Xu
Easton Li Xu
Masaki Yamamoto
Katsuhisa Yamanaka
Koichi Yamazaki
Yu Yokoi
Yuichi Yoshida
Raphael Yuster
Viktor Zamaraev
Meirav Zehavi
Peter Zeman
Thomas Zeume
Jingru Zhang
Yong Zhang
Arjana Žitnik
Stanislav Živný
Philipp Zschoche
Paweł Żyliński

Computational Complexity and Partition Functions

Leslie Ann Goldberg
Department of Computer Science, University of Oxford, Oxford, UK
leslie.goldberg@cs.ox.ac.uk

Abstract

This paper is an extended abstract of my STACS 2019 talk "Computational Complexity and Partition Functions".

2012 ACM Subject Classification Theory of computation

Keywords and phrases partition functions, approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.1

Category Invited Talk

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 334828. The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

1 Extended Abstract and Annotated Bibliography

A partition function is a polynomial which summarises properties of physical systems such as spin models (for example – the Ising model, the Potts model, the hard core model, or the monomer-dimer model). The talk does not assume knowledge of these models. The goal in the research area is to develop good approximation algorithms for partition functions, and to understand, in terms of the parameters of the models, when the partition functions can be (approximately) evaluated. Typically, a partition function is represented implicitly by a (small) graph, but it has exponentially many terms, so evaluation is non-trivial.

This talk is a survey, designed to introduce results, methods, and open problems in the area. An emerging trend is that, in order to really understand partition functions, it is useful to work over complex numbers, rather than just over the reals. This written extended abstract does not discuss methods – it is meant to serve more as an annotated bibliography – helping somebody who attended the talk to find the relevant papers.

The talk will start by introducing the independence polynomial of a graph, which is the partition function of the hard core model. This is a model of a gas whose particles occupy the vertices of a graph. Particles have non-negligible size (so cannot be adjacent). The model has a parameter λ. The partition function is given by $Z_G(\lambda) = \sum_{I \in \mathcal{I}_G} \lambda^{|I|}$, where $\mathcal{I}_G$ is the set of independent sets of a graph G and $|I|$ is the number of vertices in the independent set I. We will consider the problem of approximating $Z_G(\lambda)$, given a graph G with maximum degree at most Δ.

When λ is a real number, much is known. In particular, the complexity of approximating $Z_G(\lambda)$ is completely captured by two real-valued thresholds λ^* and λ_c which depend on Δ and satisfy $0 < \lambda^* < \lambda_c$. In the positive direction, $Z_G(\lambda)$ is efficiently approximable if λ is in the interval $(-\lambda^*, \lambda_c)$ [8, 11, 15, 16].

Outside of the interval $[-\lambda^*, \lambda_c]$, subject to suitable complexity-theoretic assumptions, no efficient approximation algorithm exists [7, 14]. The talk will discuss the relevance of

the thresholds $-\lambda^*$ and λ_c. In fact, these are the two real points on the boundary of a cardiod-shaped region in the complex plane [12] which is the region in which the "occupation ratio" at the root of a Δ-regular tree converges, as the height of the tree grows. (This means that, for parameters λ inside this region, when we consider the model on a Δ-regular tree, the contribution to the partition function from independent sets in which the root is occupied converges to some fixed fraction of the value of the total partition function.) Peters and Regts naturally asked whether this cardiod coincides with the region where efficient approximation is possible. It is now known [5] that efficient approximation is impossible outside of the cardiod (subject to complexity-theoretic assumptions). Inside the cardiod, there are many regions where we do have efficient approximation algorithms [4, 8, 11, 12] but the full picture is not fully resolved, so there are plenty of open questions.

Another interesting partition function is the matching polynomial of a graph, which is the partition function of the monomer dimer model. A "monomer" is an unmatched vertex in a matching, and a "dimer" is a pair of matched vertices. The model has a parameter γ and the partition function is given by $Z_G(\gamma) = \sum_{M \in \mathcal{M}_G} \gamma^{|M|}$, where $\mathcal{M}_G$ is the set of matchings of a graph G and $|M|$ is the number of edges in a matching M. When γ is a positive real, there is an efficient *randomised* approximation algorithm based on Monte Carlo simulation [10]. The applicability of *deterministic* approximation is fully resolved for bounded-degree graphs. It turns out that this is possible for any γ except (subject to complexity assumptions) when γ is a real number less than $-1/(4(\Delta - 1))$ [3, 6, 11]. The reason that approximation is impossible for these values of γ (and possible elsewhere) is to do with the fact that this is where the zeroes of the polynomial are located [9]. In general, the method of Barvinok [1, 2] leads to efficient approximation algorithms in complex domains where there are no roots, but in turn, the existence of roots sometimes leads to provable intractability. The approximability question has also been investigated when the bounded-degree requirement is relaxed to a notion of average degree, namely the connective constant. For graphs with bounded connective constant, it turns out that efficient approximation is possible everywhere except when γ is a negative real. Moreover, there is a set of values which is dense on the negative real axis where efficient approximation is impossible (subject to complexity assumptions) [6, 13].

There are many interesting partition functions, such as the partition functions of the Ising model and the Potts model, that will not be discussed in the talk. The chosen examples are designed to give a taste of results, methods and open problems in this research area.

References

1 A. Barvinok. *Combinatorics and Complexity of Partition Functions*. Algorithms and Combinatorics. Springer International Publishing, 2017.

2 A. I. Barvinok. Computing the Permanent of (Some) Complex Matrices. *Foundations of Computational Mathematics*, 16(2):329–342, 2016.

3 M. Bayati, D. Gamarnik, D. A. Katz, C. Nair, and P. Tetali. Simple deterministic approximation algorithms for counting matchings. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 122–127, 2007.

4 Ferenc Bencs and Péter Csikvári. Note on the zero-free region of the hard-core model. *arXiv e-prints*, page arXiv:1807.08963, July 2018. `arXiv:1807.08963`.

5 I. Bezáková, A. Galanis, L. A. Goldberg, and D. Štefankovič. Inapproximability of the independent set polynomial in the complex plane. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1234–1240, 2018. Full version available from `https://arxiv.org/abs/1711.00282`.

6 Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, and Daniel Stefankovic. The complexity of approximating the matching polynomial in the complex plane. *CoRR*, abs/1807.04930, 2018. arXiv:1807.04930.

7 Andreas Galanis, Leslie Ann Goldberg, and Daniel Štefankovič. Inapproximability of the Independent Set Polynomial Below the Shearer Threshold. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 28:1–28:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.28.

8 Nicholas J. A. Harvey, Piyush Srivastava, and Jan Vondrák. Computing the Independence Polynomial: from the Tree Threshold down to the Roots. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1557–1576, 2018. doi:10.1137/1.9781611975031.102.

9 O. J. Heilmann and E. H. Lieb. Theory of monomer-dimer systems. *Communications in Mathematical Physics*, 25(3):190–232, 1972.

10 M. Jerrum and A. Sinclair. Approximating the Permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

11 Viresh Patel and Guus Regts. Deterministic Polynomial-Time Approximation Algorithms for Partition Functions and Graph Polynomials. *SIAM J. Comput.*, 46(6):1893–1919, 2017. doi:10.1137/16M1101003.

12 Han Peters and Guus Regts. On a conjecture of Sokal concerning roots of the independence polynomial. *CoRR*, abs/1701.08049, 2017. arXiv:1701.08049.

13 A. Sinclair, P. Srivastava, D. Štefankovič, and Y. Yin. Spatial mixing and the connective constant: optimal bounds. *Probability Theory and Related Fields*, 168(1):153–197, 2017.

14 Allan Sly and Nike Sun. Counting in two-spin models on d-regular graphs. *Ann. Probab.*, 42(6):2383–2416, November 2014. doi:10.1214/13-AOP888.

15 Piyush Srivastava. Approximating the hard core partition function with negative activities. Manuscript, available at www.its.caltech.edu/~piyushs/, April 2015.

16 Dror Weitz. Counting Independent Sets Up to the Tree Threshold. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 140–149, New York, NY, USA, 2006. ACM. doi:10.1145/1132516.1132538.

The Many Facets of String Transducers

Anca Muscholl
LaBRI, University of Bordeaux, France

Gabriele Puppis
CNRS, LaBRI, France

Abstract

Regular word transductions extend the robust notion of regular languages from a qualitative to a quantitative reasoning. They were already considered in early papers of formal language theory, but turned out to be much more challenging. The last decade brought considerable research around various transducer models, aiming to achieve similar robustness as for automata and languages. In this paper we survey some older and more recent results on string transducers. We present classical connections between automata, logic and algebra extended to transducers, some genuine definability questions, and review approaches to the equivalence problem.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases String transducers, complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.2

Category Invited Talk

Funding Work supported by ANR DeLTA (ANR-16-CE40-0007).

Acknowledgements We would like to thank our colleagues, in particular D. Figueira, O. Gauwin, F. Mazowiecki and I. Walukiewicz, for their comments on a draft version of this paper.

1 Introduction

Since the early times of computer science, the notion of transduction has played a fundamental role, as computers typically process data and transform it between different formats. Numerous fields of computer science are ultimately concerned with transformations, ranging from databases to image processing, and an important issue is to perform transformations with low cost, whenever possible.

The most basic form of transformations is realized by processing inputs and producing outputs using finite memory. Such machines are called finite-state transducers. Finite-state string transducers were considered in very early work in formal language theory [22, 73, 41, 54, 67, 1, 40, 26, 14], and it was soon clear that achieving a good understanding of transducers would be much more challenging than for the classical finite-state automata. There are many aspects that change from automata to transducers, in particular non-determinism and the capability to process the input in both directions strictly increase the expressive power of transducers, while this not the case for automata [69, 77]. A further difference is that some fundamental questions, such as the equivalence problem, are undecidable for transducers.

We consider in this survey string transducers, as non-deterministic finite-state transducers that compute relations (or functions) over finite words. It turns out that for string-to-string functions, (single-valued) two-way transducers nicely capture the notion of regularity: Engelfriet and Hoogeboom showed in [42] that two-way transducers have the same expressive power as Courcelle's monadic second-order logic definable graph transductions [28], restricted to words. This equivalence supports thus the notion of "regular" functions, in the spirit of classical results on regular word languages from automata theory and logic (due to

Büchi, Elgot, Trakhtenbrot, Rabin, and others). Recently, Alur and Cerný [3] fostered new interest into this topic by introducing streaming string transducers, that have the same expressiveness as the two previous models, but allow for more flexibility in extending the model of string-to-string transductions to other quantitative models.

This survey is about classical connections between automata, logic and algebra extended to transducers, some genuine definability questions for transducers, and the main approaches to the equivalence problem. For space reasons we had to leave out several interesting topics, for example regular expressions for transductions [6, 33, 20], transductions over infinite strings [5, 45], visibly pushdown transducers [31, 49], or multi-tape transducer models [63, 25, 23]. Finally, we acknowledge the inspiration provided by a recent survey by Filiot and Reynier [50].

2 String transducers: the setting

As it happens for languages, relations on strings can be specified using many different formalisms. Following a long-lasting tradition, classes of relations are defined using equational, descriptive, or operational formalisms, e.g. algebras, logics, or automata. Unlike for languages, however, different specification formalisms for relations have often different expressive powers, as well as different closure and algorithmic properties. Another important distinction is whether we consider arbitrary relations or functions. For simplicity, hereafter we use the term *function* to generically refer to a partial function, and we say that a relation is *single-valued* if it is a (partial) function. Similarly, we say that a relation is *k-valued* (resp. *finite-valued*) is it is a union of k (resp. finitely many) partial functions.

Below, we present a few common formalisms based on logics and automata that received most of the attention in the literature.

MSO transductions

Within the realm of logics, monadic second-order (MSO) logic is one of the most expressive formalisms for specifying functions, but also relations on words. The approach, as originally proposed by Courcelle for graphs [28], consists of logically interpreting an output on a given input (or copies of it), where both input and output are seen as relational structures. More precisely, an MSO interpretation consists of a family of MSO formulas φ_{dom}, $\varphi_{\text{pos}}(x)$, $\varphi_{c\text{-lab}}(x)$ (one for every letter c in the target alphabet), and $\varphi_{\text{succ}}(x, y)$, that describe, respectively, when the output string is defined, which positions from the input become positions of the output, the labels of the output positions, and the successor relation. An *MSO transduction* can be seen as an MSO interpretation preceded by an operation that copies the input a fixed number of times, annotating each copy with a distinguished color. For example, Figure 1 gives a very simple example of MSO transduction that removes all occurrence of b from the input, and replaces every a by a c.

$$\varphi_{\text{pos}}(x) \;:=\; a(x)$$

$$\varphi_{c\text{-lab}}(x) \;:=\; \texttt{true}$$

$$\varphi_{\text{succ}}(x,y) \;:=\; x < y \;\wedge\; a(x) \;\wedge\; a(x) \;\wedge\; \forall z \;\; x < z < y \rightarrow b(z)$$

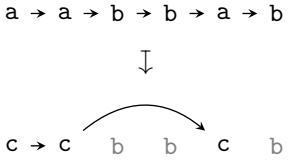


Figure 1 An MSO transduction with formulas defining positions, labels, and successor relation.

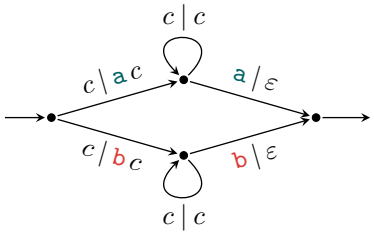


Figure 2 A 1NFT. An arrow labeled by $c \mid v$ represents a transition that consumes an input letter $c \in \{a, b\}$ and appends the word v to the output.

In order to define multi-valued relations, one can first annotate the input with arbitrary colors, and then perform an MSO transduction on the annotated input: this allows a certain degree of freedom in the possible outputs associated with the original input. The latter type of relation is called *NMSO transduction*, the 'N' standing for non-deterministic. There are of course restricted variants of (N)MSO transductions, where, for instance, monadic second-order quantification is forbidden (*FO transductions*), or where the order on output positions is directly inherited from the order on input positions (*order-preserving MSO transductions*). For example, the transduction of Figure 1 is an order-preserving FO transduction.

While the formalism of string-to-string (N)MSO transductions is already quite expressive, automata are certainly the most versatile tool for defining relations in an operational way. The literature offers many different models of automata for relations (a.k.a. transducers), which can roughly be distinguished based on the amount of non-determinism and on how the input and the associated output are consumed and produced, respectively. For instance, one can distinguish between models in which the input head moves only from left to right (one-way) or in either direction (two-way).

One-way transducers

A rather simple model is that of *one-way non-deterministic finite-state transducer* (abbreviated as 1NFT), describing the so-called *rational relations*. This is basically a non-deterministic automaton in which every transition consumes at most one letter from the input and appends a word of any length to the output. Figure 2 gives an example of a 1NFT realizing the cyclic rotation f defined by $f(ua) = au$ and $f(ub) = bu$, for all $u \in \{a, b\}^*$.

1NFT are readily seen to be equivalent to the formalism of order-preserving NMSO transductions. They can also be equally presented as languages. For example, Nivat [67] observed that the set of runs of a 1NFT can be presented as a language of interleavings of input and output letters, called synchronization language. As a consequence, every rational relation $R \subseteq \Sigma^* \times \Gamma^*$ can be presented by at least one regular synchronization language (possibly more) over $\Sigma \uplus \Gamma$.

The deterministic variant of 1NFT, denoted 1DFT and defining the so-called of *sequential functions*, has a deterministic underlying automaton and at most one transition (p, a, q, u), for every pair of states p, q and input letter a. In addition, every final state is associated with an output word that is produced at the end of the run[1].

[1] This feature is necessary to realize functions that are not monotone w.r.t. prefix order. According to the classical terminology, these functions should be called subsequential, whereas sequential was originally

Figure 3 A DSST that updates two registers, named x, y, based on whether the input letter c is the blank space or not.

As a matter of fact, interesting subclasses of rational relations can be obtained by restricting the forms of interleavings of input and output letters in the synchronization languages. For example, synchronization languages contained in $(\Sigma\Gamma)^*(\Sigma^* \cup \Gamma^*)$, which enforce a strict alternation between input and output letters, capture the class of automatic relations. This latter class is closed under all boolean operations (union, intersection, complement), and has a decidable equivalence problem and several other nice algorithmic properties [71]. The idea of defining classes of relations based on synchronization languages was investigated in detail in a series of works [43, 47, 36, 35].

Two-way transducers

The *two-way* variant of a non-deterministic finite-state transducer (abbreviated as 2NFT) allows the input head to move in any direction, to the left or to the right. This gives a more powerful model than 1NFT, which captures e.g. the relation $\{(u, u^n) \ : \ u \in \Sigma^*, n \in \mathbb{N}\}$.

When 2NFT are restricted to be single-valued, they turn out to be be equivalent to MSO transductions, so to their deterministic variant 2DFT as well [42]. This is a striking difference compared to the one-way models, where single-valued 1NFT are strictly more powerful than 1DFT.

Streaming string transducers

More recently, a third transducer model, called *streaming string transducer* (abbreviated as DSST or NSST, depending on whether it is deterministic or not), was proposed by Alur and Cerný in [3]. In this model the input is processed from left to right, while storing partial outputs in a finite set R of write-only registers. Transitions consume one input letter at a time, and can append words to the left and to the right of a register, as well as concatenate registers together. In particular, register updates correspond to substitutions $\sigma : R \to (R \cup \Gamma)^*$, where Γ is the output alphabet. As an example, the streaming transducer in Figure 3 reformats author names in bibliographies by transforming strings of the form `first-name␣second-name␣surname` into strings of the form `surname,␣first-name␣second-name`.

An SST is called *copyless* if no register occurs more than once in the right hand–=-sides of the update rules. According to the usual nomenclature we refer to copyless SST just as SST, and to unrestricted SST as *copyful SST*. Copyful DSST are close to an old formalism for grammars called HDT0L, a special family of Lindenmayer systems. Strictly speaking, HDT0L systems define word languages, as images of an initial word via sequences of morphisms, that is, words of the form $h_{i_1} \circ \cdots \circ h_{i_n}(w)$ for some indexes $i_1, \ldots, i_n$ and for a fixed tuple of morphisms $h_1, \ldots, h_k$ and a fixed word w. Such grammars can be extended naturally so as to define functions from an indexing sequence $i_1, \ldots, i_n$ (the input) to $h_{i_1} \circ \cdots \circ h_{i_n}(w)$

reserved to prefix-monotone functions realized by 1DFT without the final output rule. We prefer the generic name “sequential function”.

(the output). Copyful DSST define precisely those functions that can be obtained from the previous ones by restricting their domains to regular languages, and by possibly applying a final additional morphism [51].

In the single-valued case, NSST can be determinized and are expressively equivalent to 2DFT and MSO transductions [3]. Determinization incurs an exponential blow-up, and so does the transformation from DSST to 2DFT. Surprisingly, the reverse transformation from DSST to 2DFT turned out to be doable in quadratic time [32], through a construction based on reversible (i.e. deterministic and co-deterministic) transducers. Based on the equivalence between 2DFT, DSST, and MSO transductions, one often calls the induced class of string-to-string functions *regular*, in the spirit of classical results on regular languages, relating automata theory and logics. Unfortunately, this correspondence cannot be fully generalized to the relational case, where the transducer models 2NFT and NSST turn out to be incomparable. For example, the relation $\{(u, u^n) \ : \ u \in \Sigma^*, n \in \mathbb{N}\}$ is 2NFT-definable, but not NSST-definable, whereas the relation $\{(uv, vu) \ : \ u, v \in \Sigma^*\}$ is NSST-definable but not 2NFT-definable. It is however the case that NSST and NMSO transductions are equally expressive even in the relational case, as shown in [4]. Moreover, it is possible to capture the latter class of relations by a variant of two-way transducers, enhanced with the so-called *common guess* [19]. Formally, a two-way transducer (input-deterministic or not) has common guess if, before starting the computation, it can annotate the input with arbitrary colors, so that the same color is read each time the head revisits a position. This model is naturally closed under projections on the input, and easily simulates NSST. As a consequence, 2DFT with common guess are equivalent to NSST and to NMSO transductions:

▶ **Proposition 1.** *NMSO transductions, NSST, and 2DFT with common guess define the same class of relations.*

In the following sections we will consider two families of decision problems on relations: the class-membership and the equivalence problems. These are very natural problems, that cover most of the difficulties of reasoning with relations. As we will see, the rule of thumb in this context is that most problems turn out to be undecidable as soon as they involve non-trivial properties of *rational relations*. However, decidability can be recovered in restricted settings, notably in the single-valued, and possibly finite-valued, case. An alternative idea that is often used for recovering decidability of those problems is the origin semantics [17], that we discuss in Section 5.

3 Class-membership problems

Given any two classes $\mathcal{C}_1, \mathcal{C}_2$ of relations, the following class-membership (or characterization) problem arises naturally: *given a relation $R \in \mathcal{C}_1$, decide whether $R \in \mathcal{C}_2$*. Clearly, the decidability status and complexity of the above problem depends on the choice of the classes $\mathcal{C}_1, \mathcal{C}_2$, and on the formalisms used to represent their relations. Before discussing in more detail a few decidable cases, and their complexity, we give a necessary, and rather strict criterion for avoiding undecidability. The criterion is based on the following undecidability result for the universality problem of rational relations:

▶ **Theorem 2** (Fischer and Rosenberg [52]). *It is undecidable whether a given 1NFT realizes the universal relation $\Sigma^* \times \Gamma^*$.*

Proof. We give a simple proof of this undecidability result due to Ibarra [60], showing that undecidability even holds for a unary output alphabet. The proof reduces the Post

Correspondence Problem: given two word morphisms $f, g : \Omega^* \to \Delta^*$, with Ω and Δ finite alphabets, decide whether there is a non-empty word $w \in \Omega^+$ so that $f(w) = g(w)$. Given such f and g, let R_+ be the set of all pairs $(w\,u, c^n)$ such that $n = |u|$ and $u = f(w) = g(w)$ – here we assume w.l.o.g. that the alphabets Ω and Δ are disjoint. Intuitively R_+ consists of correct encodings of solutions of the PCP instance. Further let $R_- = (\Sigma^* \times \Gamma^*) \setminus R_+$ be the complement relation, where $\Sigma = \Omega \uplus \Delta$ and $\Gamma = \{c\}$. In particular, R_- contains pairs of the form $(w\,u, c^n) \in (\Omega^+\Delta^*) \times \Gamma^*$ such that either $n \neq |u|$, or $u \neq f(w)$, or $u \neq g(w)$. Note that R_- is universal if and only if there is no solution to the PCP instance.

Now, it suffices to verify that R_- is a rational relation. The pairs outside $(\Omega^+\Delta^*) \times \Gamma^*$ can be easily realized by a 1NFT. Similarly, the pairs $(w\,u, c^n)$ satisfying $n \neq |u|$ can be obtained by consuming the suffix u of the input while producing an output c^n of length strictly smaller or greater than $|u|$. The remaining pairs satisfying $u \neq f(w) \wedge n = |u|$ (resp. $u \neq g(w) \wedge n = |u|$) are covered using the following strategy. One guesses factorizations of w and u of the form $w_1\,a\,w_2$ and $u_1\,u_2$, so that u_2 does not begin with $f(a)$ (resp. $g(a)$), and $|u_1| = |f(w_1)|$ (resp. $|u_1| = |g(w_1)|$). Accordingly, one outputs $c^{|f(w_1)|}$ (resp. $c^{|g(w_1)|}$) while reading w_1, and $c^{|u_2|}$ while reading u_2. ◀

When considering a class-membership problem from a class $\mathcal{C}_1$ that contains all rational relations to a proper subclass $\mathcal{C}_2$ of it that contains at least the universal relation, one can sometimes adapt the proof above to show that the considered problem is undecidable. For example, following a result from [7], if $\mathcal{C}_1$ is the class of 2NFT-definable relations and $\mathcal{C}_2$ is the class of rational relations, then, given a PCP instance (f, g), one can modify the relation R_- in the previous proof by replacing the pairs $(w\,u, c^n)$ with pairs of the form $(w\,u, w\,c^n)$, where the first part w of the input is copied once to the output. These pairs can be easily produced by a two-way transducer that reads the input twice. It can be shown that, when the PCP instance has a solution, the second pass on the input is necessary for producing the correct number of c's. Otherwise, if the PCP instance has no solutions, then the relation becomes 1NFT-definable: a transducer can read the input prefix w to copy it onto the output, and then cover the arbitrary remaining parts of the input and the output. From this it follows that the class-membership problem from 2NFT to 1NFT is undecidable.

Theorem 2 exploits in a crucial way relations that associate arbitrarily many outputs with the same input, and in particular the existence of the universal relation. This suggests that decidability of universality and class-membership problems can be recovered by restricting to classes of functions. [2] We present below a few cases in which this approach succeeds, notably, for 1NFT/2NFT/NSST vs. single-valued 1NFT/2NFT/NSST (single-valuedness), for 1NFT vs. 1DFT (sequentiality), for 2NFT vs. single-valued 1NFT (one-way definability), and for 2NFT vs. FO transductions (FO-definability).

Single-valuedness

The class-membership problem from the class $\mathcal{C}_1$ of rational relations to the class $\mathcal{C}_2$ of rational functions boils down to testing single-valuedness of 1NFT. This property was first shown to be decidable by Schützenberger [75]. Later, polynomial-time algorithms were given in [58, 81, 12]. The precise complexity is NLOGSPACE-complete, which can be shown using a rather simple reduction to emptiness of one-counter automata, as explained below. Using

[2] Hereafter, for simplicity, we forbid the use of ε-moves in all models of one-way and two-way transducers (SST already forbid these moves by definition). This assumption does not affect the expressiveness of the models, nor the complexity of the considered problems, since in the single-valued case one can efficiently remove ε-moves.

a similar technique, one can prove that single-valuedness remains decidable for 2NFT and NSST, but now with PSPACE complexity (for 2NFT decidability was first shown in [30], and for NSST was shown in [4]).

▶ **Theorem 3.** *The single-valuedness problem is*
1. NLOGSPACE*-complete for 1NFT.*
2. PSPACE*-complete for 2NFT.*
3. *in* PSPACE *for NSST.*

Proof. Given a 1NFT T, one guesses an input word, together with two runs of T on it, and checks that the respective outputs are different. The test for different outputs can be done with a counter, that verifies on a given input that either (1) the lengths of the two outputs are different, or (2) there is some position in the two outputs where the respective symbols differ. In both cases the question can be reduced to emptiness of a one-counter automaton of quadratic size, which yields the claimed complexity. More precisely, the counter is updated on the basis of the number of output symbols produced by the transitions in the two runs, one contributing positively and the other contributing negatively; checking the same position in the two outputs amounts at checking that the value of counter is zero.

The result for single-valuedness of a 2NFT T follows the same idea as above, except that now the (one-way) one-counter automaton is obtained through a crossing sequence construction [77], and thus has exponential size. More precisely, one follows two runs of T on the same input by guessing tuples of states crossing each position. The length of every tuple can be bounded by $2n$, where n is the number of states of T, since to detect a violation of single-valuedness it suffices to consider only runs that visit the same position with the same state at most twice. This reduces single-valuedness of T to emptiness of a one-counter automaton of exponential size, which can be checked in PSPACE. Then PSPACE-hardness follows by reducing emptiness of intersection of finite-state automata [62].

Similarly, for NSST, the one-counter automaton that checks different outputs is exponential in the number of registers, since it needs to guess, for instance, which registers have a non-empty content that eventually appears in the final output [4]. To the best of our knowledge no matching lower bound is known in this case. ◀

Sequentiality

Let us now consider the problem of testing sequentiality of 1NFT, namely, the class-membership problem from rational to sequential functions. The problem was first shown decidable by Choffrut [26]. Later the complexity was shown to be in PTIME [11]. In fact, a close inspection to those proofs shows that the problem is NLOGSPACE-complete:

▶ **Theorem 4.** *The sequentiality problem for 1NFT is* NLOGSPACE*-complete.*

Proof. The theorem is established by first giving a topological characterization of sequential functions as those rational functions that are *Lipschitz w.r.t. the prefix distance* d; more precisely, such that there is a uniform constant k satisfying $d(f(u), f(v)) \leq k \cdot d(u, v)$ for all $u, v \in dom(f)$, where $d(u, v) = |u| + |v| - 2|u \wedge v|$ and $u \wedge v$ is the longest common prefix of u and v. For example, the function $f : cu \mapsto uc$ ($c \in \{a, b\}$, $u \in \{a, b\}^*$) is Lipschitz with constant $k = 1$, while its inverse $f^{-1} : uc \mapsto cu$ is not.

If one starts with a 1NFT T, the above characterization can be rephrased in terms of a structural property of the squared transducer T^2, which has for states the pairs of states of T and associates with any input u a pair of possible outputs (v_1, v_2) of T on u. Formally

[11], one proves that the function realized by T is Lipschitz w.r.t. the prefix distance if and only if T^2 satisfies the so-called *twinning property*:

- for every path $(q_0, q_0') \xrightarrow{u|v_1,v_2} (q, q') \xrightarrow{v|w_1,w_2} (q, q')$ in T^2, with (q_0, q_0') initial state, either $w_1 = w_2 = \varepsilon$ or $|w_1| = |w_2| \ \wedge \ v_1 \cdot w_1^\omega = v_2 \cdot w_2^\omega$ (in particular, the latter condition implies that w_1, w_2 are conjugated).

For non-empty w_1, w_2 as above, one can check the condition $|w_1| = |w_2|$ of the twinning property with a one-counter automaton of quadratic size. As for the second condition $v_1 \cdot w_1^\omega = v_2 \cdot w_2^\omega$, it boils down to checking that the trimmed part of T, seen as a transducer on infinite words with a trivial Büchi condition (all states set to be final), is single-valued. The latter problem is easily shown to be in NLOGSPACE. ◀

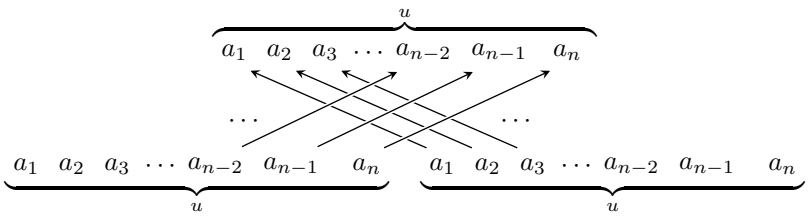
Some generalizations of the above result were given for transducers on infinite ω-words [11] and for variants of SST with updates of the form $x := y \cdot v$, where x, y are registers and v is a word (possibly over an infinitary group) [34].

One-way definability

Here we explain how to decide which regular functions are rational, or, equally, we solve the class-membership problem from single-valued 2NFT to 1NFT. The problem was first shown decidable, with non-elementary complexity, by Filiot et al. in [46]. In [9, 10], the complexity was improved to 2-EXPSPACE, together with an EXPSPACE lower bound. A refinement of a pumping argument in [10] due to I. Jecker, shows that the problem is EXPSPACE-complete.

▶ **Theorem 5.** *One-way definability of 2NFT is* EXPSPACE*-complete.*

The key notion underlying the above result is that of *inversion* of a run, which roughly corresponds to having long factors of the output that are generated without following the left-to-right order. The minimum length of factors forming an inversion is chosen as a function of the number of states of the transducer, so as to enable suitable pumping and combinatorial arguments. The characterization shows that a single-valued 2NFT is equivalent to some 1NFT if and only if every inversion in every successful run delimits a factor of the output that is periodic, with uniformly bounded period. As an example, consider a 2NFT T that realizes the function $f(u) = uu$, e.g. as follows (we draw arrows to represent some dependency relationships between positions in the output and in the input):



Every run of T that is long enough must have an inversion, essentially because one can find many output positions in the first copy of u and many output positions in the second copy of u for which the origin arrows are pairwise crossing. Based on the previous characterization, we know that f cannot be defined by a 1NFT. Observe however, that if the domain of f is restricted to a periodic language, e.g. $dom(f) = (ab)^*$, then f becomes definable by a 1NFT, e.g. one that produces ab at each position of the input.

In [8] a result similar to Theorem 5 is obtained, that characterizes effectively the functions definable by sweeping transducers, i.e., 2NFT with head reversals occurring only at the extremities of the input. This latter characterization can be used to minimize the number of

passes performed by a sweeping transducer. Moreover, [8] shows a correspondence between the number of passes of sweeping transducers and the number of registers of equivalent *concatenation-free* NSST, namely, NSST in which the right hand-side of every update rule contains at most one register (in particular, updates of the form $x := \dots y \dots z \dots$ are forbidden). Based on this correspondence one derives a minimization procedure for the number of registers of concatenation-free NSST:

▶ **Theorem 6** ([8]). *One can compute in 2-*Expspace *the minimum number of registers needed to implement a function given as a concatenation-free NSST. The complexity is* Expspace *if the given NSST is unambiguous.*

FO-definability

We finally turn towards FO-definability of regular functions. The reader may recall the deep theorem of Schützenberger about the equivalence of star-free and aperiodic word languages [74]. This theorem gives a decision procedure for knowing whether a regular language is first-order expressible (or equivalently, [64]), using semigroup properties (see also [68, 82, 38, 39] for some more recent presentations). The general idea is to lift the characterization of FO-definable regular languages to regular functions.

A first trivial observation is that FO-definability of the domain language is a necessary, but not a sufficient condition: for instance, the function $f(w) = w(2)w(4)\cdots w(2\lfloor \frac{|w|}{2} \rfloor)$ has domain Σ^*, which is a star-free language, but it is not an FO transduction. We will see that the natural and correct choice is rather to test aperiodicity on the automaton underlying the transducer – intuitively, aperiodicity means that any two inputs u^n and u^{n+1}, for large enough n, induce the same transformation in the underlying automaton.

The FO-definability problem for regular functions is still open in its full generality, and thought to be difficult. Non-effective characterizations were obtained for 2NFT [24] and for DSST [48]. In both cases the characterization is expressed in terms of the aperiodicity property of a suitable transition monoid – for 2NFT, this is just the standard transition monoid of the underlying automaton, while for DSST, the transition monoid is defined on the basis of the underlying automaton and the register updates.

▶ **Theorem 7** ([24, 48]). *The string-to-string FO transductions are precisely*
- *the functions realized by 2NFT with an aperiodic transition monoid,*
- *the functions realized by DSST with an aperiodic transition monoid.*

▶ **Open question 8.** *Is it decidable whether a transduction given by a 2NFT/DSST is first-order definable?*

There are essentially two ways by means of which one can simplify the FO-definability problem and establish decidability: by either moving from classical semantics to origin semantics, or by restricting further the class of functions. The first approach was followed in [17], where the problem of deciding whether a given 2DFT is origin-equivalent to some FO transduction was shown to be decidable. We will discuss this approach later, in Section 5, while here we focus on the second approach, specifically by considering the subclass of rational (rather than regular) functions. We point out that both approaches, in the end, give a notion of 'minimal' object that is used as a canonical representative of the given function.

For the sake of presentation, we begin by considering an even more restricted case, that of sequential (i.e. 1DFT-definable) functions. As already explained, the goal here is to test aperiodicity on the automaton underlying the minimal 1DFT, where minimality is intended

in a strong categorical sense, namely, as an object to which every other equivalent object can be homomorphically mapped to:

▶ **Theorem 9** ([27]). *For every sequential function f, there is a transducer T_f that realizes f and such that for every equivalent trimmed transducer T, there is a unique morphism from T to T_f. Moreover, T_f can be constructed in* PTIME *from a 1DFT realizing f.*

Before sketching the construction of the minimal transducer T_f, it is worth spending a few more words on the notion of transducer morphism. For this notion, it is tempting to adopt the classical definition of graph morphism, that preserves transitions as labeled edges. However, there are 1NFT that realize the same function using a minimal number of states, but that are not isomorphic in the usual sense, e.g.:

This simple observation suggests that the correct notion of transducer morphism should take into account the possible different 'speeds' at which equivalent transducers may produce their outputs. This can be done by pairing the morphism application with a mapping from states to elements of the free group $(\Gamma \cup \Gamma^{-1})^*$ that encode output lags.

Proof of Theorem 9. The main ingredient underlying the definition of T_f is a notion of congruence for sequential functions, very similar to the Nerode congruence. The additional crux is to correctly identify words that induce the same behavior w.r.t. the produced output. This requires a normalization step, that guarantees that outputs are produced *as early as possible*, based only on the finite information that comes from the consumed part of the input. Formally, given a function f, let $\hat{f}$ be the function that has as domain the prefix closure of the domain of f, and such that $\hat{f}(u) = \bigwedge\{f(uv) \ : \ uv \in dom(f)\}$ is the longest common prefix of all $f(uv)$. Using $\hat{f}$, one defines the equivalence $\equiv_f$ by $u \equiv_f v$ if and only if for all words w, either $f(uw)$ and $f(vw)$ are undefined, or they are both defined and, once the respective prefixes $\hat{f}(u)$ and $\hat{f}(v)$ are removed, they result in the same word. It is not difficult to see that $\equiv_f$ is a right congruence, and that it has finite index if and only if f is sequential. Finally, one defines[3] the minimal transducer T_f that has as states the $\equiv_f$-equivalence classes $[u]$ ($u \in \Sigma^*$) and transitions of the form $[u] \xrightarrow{a|v} [ua]$, where v is obtained from $\hat{f}(ua)$ by removing the prefix $\hat{f}(u)$. ◀

Based on Theorem 7 and Theorem 9, one concludes that f is an FO transduction if and only if the underlying automaton of T_f is finite and aperiodic. Moreover, it is easy to see that if f is defined by a 1DFT, then T_f is finite, and aperiodicity can be tested in PSPACE. This shows that the FO-definability problem for 1DFT is in PSPACE.

Recently, Filiot et al. [44] generalized the characterization of FO-definability from 1DFT to 1NFT (in fact, to the equivalent, but less succinct model of bimachines):

▶ **Theorem 10** ([44]). *A rational function f is an FO transduction if and only if its canonical bimachine is aperiodic. Moreover, if f is given by a bimachine (resp. by a 1NFT), then the property can be decided in* PSPACE *(resp. 2-*EXPTIME*).*

[3] Technically speaking, the outlined definition of T_f misses the production of the initial prefix $\hat{f}(\varepsilon)$ of the output. To fix this problem one can equip transducers with an initial production rule, very similar to the final production rule that they already have (cf. [27]).

We discuss the terminology and the main ideas underlying the decidability of the above problem (for more details, we refer the reader to [44, 50]). First, the model of bimachine, which was originally introduced in [74], is conveniently seen as a 1DFT enhanced with *regular look-ahead*. Here by regular look-ahead we mean a finite-state automaton that deterministically processes the input from right to left (so it is co-deterministic). The transitions of a 1DFT T enhanced with the look-ahead automaton A depend on the current state of T, the input symbol, and the state reached by A after processing the suffix of the input up to the current position. In [41], it is shown that 1NFT are exactly as expressive as bimachines (or equally, 1DFT with regular look-ahead), with a doubly-exponential time translation from the former to the latter – one exponent is due to the co-deterministic look-ahead, and the other exponent is due to the determinization of the transducer parametrized by the look-ahead.

The existence of a canonical bimachine realizing a rational function f essentially relies on the possibility of computing a minimal look-ahead automaton based only on f, and then using this to reduce to the previous minimization problem for simple 1DFT without look-ahead. More precisely, in [70] it is shown that, given any function f, one can define a co-deterministic (possibly infinite) automaton A_f such that f is rational if and only if

- A_f is finite,
- the function $f[A_f]$ that maps every input $u \in dom(f)$, annotated with the unique run of A on u, to the output $f(u)$, is sequential.

For example, the co-deterministic automaton A_f can be obtained from a left congruence $\sim_f$ defined by $u \sim_f v$ if and only if there is a bound k (depending on u, v) such that, for all words w, either $f(wu), f(wv)$ are undefined, or they are both defined and $d(f(wu), f(wv)) \leq k$, where d is the prefix distance defined on page 7.

Summing up, assuming that f is rational, one constructs the minimal co-deterministic automaton A_f and the minimal 1DFT T_f for the sequential function $f[A_f]$. Pairing A_f and T_f results in a minimal and canonical bimachine realizing f. It is then routine to prove that f is an FO transduction if and only if both A_f and T_f are aperiodic.

We conclude by observing that the complexity in Theorem 10 is optimal when we are concerned with functions described by bimachines (PSPACE-hardness follows from the usual reduction from universality of finite-state automata). However, when the function is given by a 1NFT, the FO-definability problem is only known to be between PSPACE and 2-EXPTIME.

4 The equivalence problem

Historically, string transducers have been first studied within their deterministic, or unambiguous, one-way machine models. One reason for this is that many fundamental problems about rational relations are undecidable, in particular the equivalence problem (recall Theorem 2).

This section reviews the status of the equivalence problem for various types of string transducers, together with a selection of the main technical arguments for proving decidability. One of the first references for decidability in the deterministic case, and specifically for length-preserving 1DFT, is due to Moore [66]. The result was then extended progressively. Equivalence for 1DFT was shown decidable by Blattner and Head [16]. The same authors also showed that it is decidable whether a 1NFT is single-valued (see also [75]), and that equivalence of single-valued 1NFT is decidable [15]. For two-way transducers, Gurari showed that equivalence is PSPACE-complete in the deterministic case [56], by reducing it to emptiness of bounded-reversal counter machines [59], a polynomial-time solvable problem [57]. Interestingly, the complexity remains the same for single-valued 2NFT, even though, to the best of our knowledge there is no reference available for this latter result.

The above decidability and complexity results can be presented in a uniform way using reductions to equivalence of finite-state automata and to single-valuedness of relational transducers (see Section 3):

▶ **Theorem 11.** *The equivalence problem $T_1 \stackrel{?}{=} T_2$ is*
1. NLOGSPACE*-complete if T_1, T_2 are 1DFT.*
2. PTIME *if T_1, T_2 are unambiguous 1NFT.*
3. PSPACE*-complete if T_1, T_2 are 2DFT, or single-valued 2NFT or NSST.*

Proof. The idea is to reduce the equivalence problem for T_1 and T_2 to testing equivalence of the domains and single-valuedness of the union of T_1 and T_2. For example, when T_1, T_2 are 1DFT, testing equivalence of the underlying deterministic automata is in NLOGSPACE, and the same for testing single-valuedness of the 1NFT $T_1 \cup T_2$ (cf. Theorem 3).

For single-valued 1NFT, it is the complexity of the equivalence problem for the domains that dominates, since this is in general PSPACE-complete [62]. However, if the transducers are unambiguous (which is not a restriction, see e.g. [40, 76]), then $T_1 \stackrel{?}{=} T_2$ can be solved in PTIME [58], since equivalence of unambiguous automata has a polynomial-time solution [78].

Finally, when T_1, T_2 are 2NFT (or NSST), single-valuedness of $T_1 \cup T_2$ is in PSPACE, again by Theorem 3. We need only to show that domain equivalence can also be solved in PSPACE. While this is easy for NSST, it is perhaps not completely obvious that we can check equivalence of two-way, non-deterministic automata in PSPACE. This is indeed the case using e.g. a construction due to Vardi [79]: for every two-way, non-deterministic automaton A of size n, one can construct two deterministic, one-way automata of size $2^{O(n^2)}$, recognizing respectively $L(A)$ and its complement. As usual, PSPACE-hardness for 2NFT follows from emptiness of intersection of finite-state automata [62]; for NSST it follows from universality of finite-state automata [65]. ◀

It is worth noting that the exact complexity for the equivalence problem of DSST is unknown, since the problem is shown to be in PSPACE and NLOGSPACE-hard:

▶ **Open question 12.** *What is the exact complexity of the equivalence problem for DSST?*

A powerful tool that can be used to establish decidability of equivalence problems on transducers is the *Ehrenfeucht conjecture*. It was originally stated as a conjecture about formal languages: for every language $L \subseteq \Sigma^*$, there is a finite subset $F \subseteq L$ such that for every homomorphisms $f, g : \Sigma^* \to \Delta^*$,

$$\forall u \in L \quad f(u) = g(u) \qquad \text{if and only if} \qquad \forall u \in F \quad f(u) = g(u).$$

Such a set F is called a *test set* for L. There is an equivalent formulation of Ehrenfeucht conjecture in terms of word equations [61]. Let Σ and Ω be two alphabets, where the elements in Ω are variables. A word equation is a pair $(u, v) \in \Omega^* \times \Omega^*$, and a solution is a homomorphism $\sigma : \Omega \to \Sigma^*$ such that $\sigma(u) = \sigma(v)$. The Ehrenfeucht conjecture then says that any system of equations over Ω has a finite, equivalent subsystem, where equivalence means that the solution sets are the same. The proof of Ehrenfeucht conjecture is based on encodings of the free monoid and Hilbert's basis theorem [2, 55].

An elegant application of Ehrenfeucht conjecture is due to Culik and Karhumäki, who established decidability of the equivalence problem for k-valued 1NFT [29]. Their decidability result does not come with any complexity upper bound, however the following stronger result leads to an algorithm of elementary complexity:

▶ **Theorem 13** ([80, 72])**.** *Every k-valued 1NFT can be effectively decomposed into a union of k unambiguous 1NFT of exponential size.*

Since k-valuedness of 1NFT can be checked in PTIME [58], this yields:

▶ **Corollary 14.** *The equivalence problem for k-valued 1NFT is in* EXPTIME *(for fixed k).*

Let us come back to the argument used by Culik and Karhumäki for the equivalence of k-valued 1NFT [29]. The proof consists of two parts. First the Ehrenfeucht conjecture is used to show that there is a finite test set for any two k-valued transducers with at most n states (for any fixed n). In this case, a test set is a language F such that *any* two transducers with at most n states are equivalent if and only if they are equivalent on inputs from F. In the second part it is shown how to find effectively a finite test set for the class of k-valued transducers with at most n states. This step amounts to determine whether two finite systems of equations are equivalent, which reduces to solvability of word equations, and can be solved by Makanin's algorithm (see e.g. [37]). This proof idea extends to k-valued 2NFT:

▶ **Theorem 15** ([29])**.** *The equivalence problem for k-valued 2NFT is decidable.*

Proof. We sketch the proof of [29] for one-way transducers, then explain how it adapts to two-way transducers. We tacitly assume that all transducers are trimmed.

For the existence of a test set one uses the formulation of Ehrenfeucht conjecture with word equations. As mentioned, the set we are looking for will be a test set for *any* pair of k-valued transducers with at most n states (and fixed alphabets). An important observation is that for k-valued transducers (one-way or two-way), the following holds: for every pair of states p, q and input letter a, at most k different output words can label the transition (p, a, q). Thanks to this, any k-valued transducer with at most n states can be described by a partial mapping $\Delta : \{1, \dots, n\}^2 \times \Sigma \times \{1, \dots, k\} \to \Omega$ into a *finite* set Ω of variables (a *schema*), paired with an interpretation $\sigma : \Omega \to \Gamma^*$. The (uninterpreted) output of a run that respects a schema Δ is then a word over Ω. For instance, below we depict a transducer (on the left) with the corresponding schema (on the right):

Note that the transducer is 3-valued, and its schema satisfies for instance the equation $x_1 x_3 x_5^m = x_2 x_3 x_5^m$, relating two runs with the same output and over the input $u = a^{m+2}$.

Now, for every pair of concrete k-valued transducers T_1, T_2 with at most n states, and for every input u, the runs of T_1 and respectively T_2, over u, can be partitioned into at most k groups, each associated with an output among the k possible ones. For each group of runs, we can write word equations over Ω as expected. So two transducers as above, being interpretations of some schemata Δ_1, Δ_2, are equivalent only if they satisfy a formula φ_n of the form $\bigwedge_{u \in \Sigma^*} \bigvee_\pi S_\pi$, where π ranges over the possible Δ_1, Δ_2 and partitions the set of runs of Δ_1 and respectively Δ_2, over the input u, into at most k groups, and S_π is the associated system of word equations. Ehrenfeucht conjecture is used in [29] to show that any φ_n as above is equivalent to some *finite* formula $\varphi_{n,m} = \bigwedge_{u \in \Sigma^{\le m}} \bigvee_\pi S_\pi$.

For the effectiveness part, let us assume that for some m, the formulas $\varphi_{n,m}$ and $\varphi_{n,m+1}$ are equivalent, i.e., they have the same sets of solutions (recall that testing the latter equivalence reduces to solvability of word equations). The goal is to prove that $\Sigma^{\le m}$ is a test set, namely, for all $r > m$ and all k-valued transducers T_1, T_2 of size at most n,

$$T_1 \equiv_r T_2 \qquad \text{if and only if} \qquad T_1 \equiv_m T_2$$

where $T_1 \equiv_m T_2$ stands for equivalence relativized to inputs of length at most m. The above property is proved by induction on r, as follows. Suppose for the moment that, given any

transducer T and any input letter a, one can construct a transducer T_a with the same number of states as T and such that $T_a(u) = T(au)$. Clearly, $T_1 \equiv_{r+1} T_2$ is equivalent to $T_{1,a} \equiv_r T_{2,a}$ for every $a \in \Sigma$, and $T_1 \equiv_0 T_2$ (the latter being abbreviated as $(*)$ below). Therefore,

$$\begin{array}{ccccc} T_1 \equiv_{r+1} T_2 & \Leftrightarrow & T_{1,a} \equiv_r T_{2,a} \ (\forall a \in \Sigma) \text{ and } (*) & & T_1 \equiv_m T_2. \\ & & \Updownarrow \text{ (ind. hyp.)} & & \Updownarrow \\ & & T_{1,a} \equiv_m T_{2,a} \ (\forall a \in \Sigma) \text{ and } (*) & \Leftrightarrow & T_1 \equiv_{m+1} T_2 \end{array}$$

This shows that $\Sigma^{\leq m}$ is a test set. According to the first part, we are guaranteed to find such an m.

If T is a 1NFT, one can build T_a while preserving the number of states, just by a shortcut of the transitions departing from the initial state, which can be assumed to have no incoming transition. If T is a 2NFT, a similar idea works, but now the shortcut is more complex since the first input position can be read several times. Still we can shortcut the transitions involving the first input position, assuming, w.l.o.g., that all our transducers have the extra possibility to check whether the current position is the first one. ◀

Recall that single-valued NSST and 2NFT are equivalent transducer models (actually equivalent to DSST and 2DFT). It is natural to ask whether this equivalence extends to k-valued transducers. One direction is an easy generalization of the deterministic case:

▶ **Proposition 16.** *For every k-valued 2NFT, there is an equivalent k-valued NSST.*

▶ **Open question 17.** *Are NSST strictly more expressive than 2NFT in the k-valued case?*

The approach of Culik and Karhumäki [29] does not appear to generalize to NSST (and not even to the equivalent model of 2NFT with common guess, cf. Section 2). The main difficulty is that it is not clear how to construct the transducer T_a from T while preserving the number of states. This difficulty, however, can be overcome for k-valued, 1-register NSST, which thus turn out to have a decidable equivalence problem:

▶ **Theorem 18.** *The equivalence problem for k-valued, 1-register NSST is decidable.*

The above result also follows from [53], where, in analogy with Theorem 13, it is shown that k-valued, 1-register NSST can be effectively decomposed into k unambiguous NSST. Not surprisingly, a generalization of the decomposition theorem for k-valued NSST with multiple registers faces the same difficulties as the approach of Culik and Karhumäki. We conjecture however that Theorem 18 generalizes to concatenation-free NSST:

▶ **Conjecture 19.** *The equivalence problem for k-valued, concatenation-free NSST is decidable.*

The difficulty in proving this conjecture is combinatorial, since it requires to determine if for two partial runs of an NSST, there is some extension that makes their outputs equal.

We conclude this section by presenting a different approach to show equivalence of transducers, that was recently applied to copyful DSST:

▶ **Theorem 20** ([51])**.** *The equivalence problem for copyful DSST is decidable.*

Proof. The original proof of [51] shows that copyful DSST are equivalent to HDT0L systems, and then applies [29]. An alternative proof was presented in [13], translating copyful DSST into so-called polynomial automata, and showing that the zeroness problem for such automata

is decidable. Recently, Bojańczyk reformulated the proof in terms of polynomial grammars [18]. We briefly sketch his proof below.

The starting idea is to encode words by values computed by polynomials. Such encodings were known even before Matiyasevich' negative solution to Hilbert's 10th problem and Makanin's algorithm for word equations. For example, [18] represents a binary word w by the pair $enc(w) = (bin(w), 2^{|w|})$, where $bin(w)$ is the integer with binary representation w. Word concatenation becomes then a polynomial operation: if w_i is encoded by $A_i = (A_{i,1}, A_{i,2})$, then $w_1 w_2$ is encoded by the pair of integers $A_1 \odot A_2 := (A_{1,1}A_{2,2} + A_{2,1},\ A_{1,2}A_{2,2})$. A copyful DSST easily translates into a context-free grammar that generates pairs of integers, a so-called polynomial grammar [18].

For example, for the DSST of Figure 3, the grammar has one non-terminal $A = (A_x, A_y)$ for the unique state, plus a starting symbol S for the final output. The rules are as follows:

$$\begin{array}{rcll} S & \to & A_x \odot enc(\texttt{,}\,\sqcup) \odot A_y & \\ (A_x, A_y) & \to & (enc(\varepsilon),\ A_y \odot enc(\sqcup) \odot A_x) & \\ (A_x, A_y) & \to & (A_x \odot enc(c),\ A_y) & (\forall c \neq \sqcup). \end{array}$$

The question of whether two copyful DSST T, T' are equivalent reduces to asking whether the difference $S - S'$ of the starting symbols of the associated grammars generates only the null vector. Using the decidability of the first-order of reals, there is a semi-algorithm to know if a polynomial grammar can generate some non-zero value, by enumerating derivations. Using Hilbert's basis theorem, there is also a semi-algorithm for proving that a polynomial grammar generates only the null vector, by enumerating finite sets of polynomials and checking that (1) they induce a solution for the grammar, and (2) they produce only zero (see [18] for details). This shows that one can decide whether a polynomial grammar generates only the null vector, hence the equivalence of copyful DSST. ◀

5 Transducers with origins

Transducers with origin information have been introduced by Bojańczyk in [17] as a way to recover the algebraic world that appears to get lost when switching from regular string languages to regular string transductions. We recall that for 2DFT or DSST there is no canonical (e.g. minimal) model on which properties like aperiodicity can be tested. As we saw in Section 3, the situation is slightly better for one-way transducers, that do have canonical models – the minimal transducer in the deterministic case, and the canonical bimachine in the general case.

In the standard semantics, a string transduction is simply a relation $R \subseteq \Sigma^* \times \Gamma^*$ that consists of pairs of words. In the origin semantics, a transduction is a set of pairs from $\Sigma^* \times (\Gamma \times \mathbb{N})^*$ such that the output from $(\Gamma \times \mathbb{N})^*$ also records at which position of the input it was generated. Every formalism used for describing transductions, be it transducers, logic or expressions, can be enriched by the origin information in a natural way. For example, the origin semantics for 1NFT is the same as a synchronization language – recall from Section 2 that this is a language of interleavings of input and output letters, and that there could be many synchronization languages representing the same relation. For 2NFT, the origin semantics is conveniently described by *origin graphs*, which are special graphs consisting of two total orders (for the input and the output), together with edges from output positions to input positions (the origin mapping). For example, the two origin graphs of Figure 4 are different, although they have the same input/output pair.

Recall that a string-to-string function is called regular if it is realized by one of the following, equivalent formalisms: 2DFT, DSST, MSO transductions. The main contribution

Figure 4 Two origin graphs; the upper line is the input, the lower line the output.

of [17] was to show a Myhill-Nerode theorem for regular functions with origin semantics, and give an effective characterization for the subclass of FO transductions. The result amounts to define left and right congruences, as follows. Recall that the Nerode (right) congruence for a language K defines $u \equiv v$ if $u^{-1}K = v^{-1}K$. Similarly, for a function $f : \Sigma^* \to \Gamma^*$, we define a right congruence by $u \equiv_R v$ if $f_{u\_} = f_{v\_}$, where $f_{u\_}(w)$ is obtained from $f(uw)$ by replacing all maximal non-empty factors with origins inside u by a special marker $\bullet$. For example, for $f(w) = ww$ there are two possibilities for $f_{u\_}$, depending on whether u is empty or not: one is $v \mapsto vv$, and the other is $v \mapsto \bullet\, v \bullet v$. Symmetrically, one defines a left congruence $\equiv_L$ using $f_{\_u}(w)$, that is obtained from $f(wu)$ by replacing with $\bullet$ all maximal non-empty factors with origins inside u.

▶ **Theorem 21** ([17]). *Let f be a string-to-string function with origins.*

1. *f is regular if and only if the associated congruences $\equiv_L$ and $\equiv_R$ have finite index.*
2. *A regular f is origin-equivalent to an FO transduction if and only if all classes of the associated congruences $\equiv_L$ and $\equiv_R$ are FO-definable languages.*

An immediate consequence of the above theorem is that one can decide FO-definability in the origin semantics. Using that the congruences $\equiv_L$ and $\equiv_R$ can be tested in Pspace we get:

▶ **Corollary 22.** *The problem whether given 2DFT or DSST is origin-equivalent to some FO transduction is* Pspace*-complete.*

Register minimization of DSST is another problem that can be solved under the origin semantics, as stated below. The main ingredient of the characterization is the notion of *k-crossing*. Formally, an output position y is said to cross an input position x if the origin of y is x or to the left of x, and the origin of $y + 1$ is to the right of x (if $y + 1$ is defined). An origin graph is k-crossing if every input position is crossed by at most k output positions.

▶ **Theorem 23** ([19]). *A set $\mathcal{G}$ of origin graphs is realizable by a k-register DSST if and only if it satisfies all conditions below:*

1. *every graph in $\mathcal{G}$ is k-crossing,*
2. *there is a constant c such that, for all graphs in $\mathcal{G}$, every input position is the origin of at most c output positions,*
3. *$\mathcal{G}$ is MSO-definable.*

Taking the right hand-side graph of Figure 4 as an example, one sees that $c = k = 1$, and the MSO formula defining such graphs says, among other things, that the origin of the output position $y + 1$ is the predecessor of the origin of y.

Another quite interesting phenomenon is that the equivalence problem becomes decidable in the origin semantics, even for non-deterministic, two-way transducers with common guess:

▶ **Theorem 24** ([21]). *The origin-equivalence problem for 2NFT (possibly enhanced with common guess) is* Pspace*-complete.*

The previous result is consistent with the intuition that under the origin semantics, one does not reason anymore on two separate objects (input and output), but on one single object (the origin graph). In this perspective, it becomes natural to ask whether some amount of information can be removed from the origin semantics, while still preserving decidability of equivalence. The argument below reveals that not much information can be removed.

Consider a modified notion of origin, that instead of mapping output positions to input positions, defines an equivalence on the output so that any two positions are equivalent when they have the same origin in the input. A small modification of the proof of Theorem 2 show that decidability of equivalence is lost under this weaker semantics. Indeed, one can replace the pairs (wu, c^n) that encode non-solutions of the PCP instance by pairs of the form $(w'u', c^n)$, where $w'u'$ is obtained from wu by inserting a fixed dummy word $\sqcup \cdots \sqcup$ between every two consecutive positions, so that it becomes possible to produce at most one output letter at each input position. In this case, the equal-origin information becomes vacuous, and the universality problem turns out to be undecidable.

6 Conclusions

We have reviewed some of the fundamental questions on string transducers, mostly concerning characterization and equivalence problems. Some of the problems were shown decidable, notably in the single-valued and in the finite-valued case, but a few important problems remain open.

This is the case, for instance, for the FO-definability problem for regular functions: given a 2DFT or a DSST, is it equivalent to some FO transduction? As a possible first attempt, one may try to look at FO-definability for restricted variants of 2DFT and DSST (e.g. sweeping 2DFT and concatenation-free DSST), trying to lift the known characterization for 1DFT.

Another challenging question that remains unanswered is whether k-valued NSST are as expressive as k-valued 2NFT, or strictly more expressive. The related problem of testing equivalence of k-valued NSST is also open. A solution to both problems might stem from a decomposition theorem, that is, from a proof that every k-valued NSST can be decomposed into an equivalent finite union of DSST. This latter result however seems rather difficult to get, due to the underlying word combinatorics. Also in this case, it might be helpful to consider first the problem for the restricted class of concatenation-free NSST.

We finally recall a couple of open problems related to equivalence and single-valuedness of SST. We have seen that equivalence for single-valued NSST is PSPACE-complete, but what about DSST? Is the equivalence problem still PSPACE-hard, or is it possible to exploit determinism to lower the complexity? Similarly, the problem of testing singe-valuedness on NSST is shown to be in PSPACE, but no matching lower bound is known. Finally, the problem of minimizing the number of registers in a DSST is also open.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. A general theory of translation. *Math. Syst. Theory*, 3(3):193–221, 1969.

2 M.H. Albert and J. Lawrence. A proof of Ehrenfeucht's conjecture. *Theor. Comput. Sci.*, 41(1):121–123, 1985.

3 Rajeev Alur and Pavel Cerný. Expressiveness of streaming string transducer. In *IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

4 Rajeev Alur and Jyotirmoy Deshmukh. Nondeterministic streaming string transducers. In *International Colloquium on Automata, Languages and Programming (ICALP'11)*, volume 6756 of *LNCS*. Springer, 2011.

5 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular Transformations of Infinite Strings. In *Proc. of Annual IEEE Symposium on Logic in Computer Science (LICS'12)*, pages 65–74. IEEE, 2012.

6 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *Joint meeting of Annual Conference on Computer Science Logic (CSL) and Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LIPIcs, pages 9:1–9:10. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2014.

7 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way Definability of Sweeping Transducers. In *IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 178–191. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

8 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing Resources of Sweeping and Streaming String Transducers. In *International Colloquium on Automata, Languages, and Programming (ICALP'16)*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

9 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Untwisting two-way transducers in elementary time. In *ACM/IEEE Symposium on Logic in Computer Science (LICS'17)*. IEEE Computer Society, 2017.

10 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of two-way word transducers. *Logical Methods in Computer Science*, 14(4):1–54, 2018.

11 Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theor. Comput. Sci.*, 289(1):225–251, 2002.

12 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292:45–63, 2003.

13 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'17)*, pages 1–12. IEEE, 2017.

14 Jean Berstel. *Transductions and context-free languages*. Teubner Studienbücher Stuttgart, 1979.

15 Meera Blattner and Tom Head. Single-valued a-transducers. *J. Comput. and System Sci.*, 15:310–327, 1977.

16 Meera Blattner and Tom Head. The decidability of equivalence for deterministic finite transducers. *J. Comput. and System Sci.*, 19:45–49, 1979.

17 Mikolaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages and Programming (ICALP'14)*, number 8572 in LNCS, pages 26–37. Springer, 2014.

18 Mikolaj Bojańczyk. The Hilbert Method for Transducer Equivalence. *ACM SIGLOG News*, January 2019.

19 Mikolaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which Classes of Origin Graphs Are Generated by Transducers? In *International Colloquium on Automata, Languages and Programming (ICALP'17)*, volume 80 of *LIPIcs*, pages 114:1–114:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

20 Mikolaj Bojańczyk, Laure Daviaud, and Shankara Narayanan Krishna:. Regular and First-Order List Functions. In *Proc. of Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 125–134. ACM, 2018.

21 Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-Equivalence of Two-Way Word Transducers Is in PSPACE. In *IARCS Annual Conference on Foundations of*

Software Technology and Theoretical Computer Science (FSTTCS'18), volume 122 of *LIPIcs*, pages 1–18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

22 Julius Richard Büchi. Weak Second-order Arithmetic and Finite Automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

23 Olivier Carton, Christian Choffrut, and Serge Grigorieff. Decision problems among the main subfamilies of rational relations. *ITA*, 40(2):255–275, 2006.

24 Olivier Carton and Luc Dartois. Aperiodic two-way transducers and FO-transductions. In *Proc. of EACSL Annual Conference on Computer Science Logic (CSL'15)*, LIPIcs, pages 160–174. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

25 Olivier Carton, Léo Exibard, and Olivier Serre. Two-Way Two-Tape Automata. In *Proc. in Developments in Language Theory (DLT'17)*, number 10396 in LNCS, pages 147–159. Springer, 2017.

26 Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5:325–338, 1977.

27 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(131-143), 2003.

28 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

29 Karel Culik II and Juhani Karhumäki. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theor. Comput. Sci.*, 47:71–84, 1986.

30 Karel Culik II and Juhani Karhumäki. The Equivalence Problem for Single-Valued Two-Way Transducers (on NPDT0L Languages) is Decidable. *SIAM J. Comput*, 16(2):221–230, 1987.

31 Luc Dartois, Emmanuel Filiot, Pierre-Alain Reynier, and Jean-Marc Talbot. Two-Way Visibly Pushdown Automata and Transducers. In *Proc. of Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*, pages 217–226. ACM, 2016.

32 Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On Reversible Transducers. In *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP'17,)*, number 113 in LIPIcs, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

33 Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular Transducer Expressions for Regular Transformations. In *Proc. of Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 315–324. ACM, 2018.

34 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A Generalised Twinning Property for Minimisation of Cost Register Automata. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*, pages 857–866. ACM, 2016.

35 María Emilia Descotte, Diego Figueira, and Santiago Figueira. Closure properties of synchronized relations. In *International Symposium on Theoretical Aspects of Computer Science (STACS'19)*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. to appear.

36 María Emilia Descotte, Diego Figueira, and Gabriele Puppis. Resynchronizing Classes of Word Relations. In *International Colloquium on Automata, Languages, and Programming (ICALP'18)*, volume 123 of *LIPIcs*, pages 1–13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

37 Volker Diekert. Makanin's Algorithm. In M. Lothaire, editor, *Algebraic combinatorics on words*. Cambridge University Press, 2002.

38 Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on Small Fragments of First-Order Logic over Finite Words. *International Journal of Foundations of Computer Science*, 19(3):513–548, 2008.

39 Volker Diekert and Manfred Kufleitner. A Survey on the Local Divisor Technique. *Theor. Comput. Sci.*, 610:13–23, 2016.

40 Samuel Eilenberg. *Automata, Languages and Machines*. Academic Press, New York, 1976.

41 Calvin C. Elgot and Jorge E. Mezei. On Relations Defined by Generalized Finite Automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.

42 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.

43 Diego Figueira and Leonid Libkin. Synchronizing Relations on Words. *Theory Comput. Syst.*, 57(2):287–318, 2015.

44 Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Aperiodicity of Rational Functions Is PSPACE-Complete. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'16)*, volume 65 of *LIPIcs*, pages 13:1–13:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

45 Emmanuel Filiot, Olivier Gauwin, Nathan Lhote, and Anca Muscholl. On Canonical Models for Rational Functions over Infinite Words. In *Proc. of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*, volume 30 of *LIPIcs*, pages 1–17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

46 Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From Two-Way to One-Way Finite State Transducers. In *ACM/IEEE Symposium on Logic in Computer Science (LICS'13)*, pages 468–477, 2013.

47 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On Equivalence and Uniformisation Problems for Finite Transducers. In *Proc. of nternational Colloquium on Automata, Languages, and Programming (ICALP'16)*, number 125 in LIPIcs, pages 1–14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

48 Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. First-order Definable String Transformations. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'14)*, LIPIcs, pages 147–159. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

49 Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Visibly pushdown transducers. *J. Comput. Syst. Sci.*, 97:147–181, 2018.

50 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News*, pages 4–19, 2016.

51 Emmanuel Filiot and Pierre-Alain Reynier. Copyful Streaming String Transducers. In *International Workshop on Reachability Problems (RP'17)*, number 10506 in LNCS, pages 75–86. Springer, 2017.

52 Patrick C. Fischer and Arnold L. Rosenberg. Multi-tape one-way nonwriting automata. *J. Comput. and System Sci.*, 2:88–101, 1968.

53 Paul Gallot, Anca Muscholl, Gabriele Puppis, and Sylvain Salvati. On the Decomposition of Finite-Valued Streaming String Transducers. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS'17)*, volume 66 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

54 Seymour Ginsburg and Gene F. Rose. A characterization of machine mappings. *Canad. J. Math.*, 18:381–388, 1966.

55 Victor S. Guba. Equivalence of infinite systems of equations in free groups and semigroups to finite subsystems. *Mat. Zametki*, 40(3):688—-690, 1986.

56 Eitan M. Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM Journal of Computing*, 448–452, 1982.

57 Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *J. Comput. and System Sci.*, 16(1):61–66, 1981.

58 Eitan M. Gurari and Oscar H. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Math. Syst. Theory*, 16(1):61–66, 1983.

59 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *JACM*, 1978.

60 Oscar H. Ibarra. The unsolvability of the equivalence problem for e-free NGSM's with unary input (output) alphabet and applications. *SIAM J. of Comput.*, 7(4):524–532, 1978.

61 Juhani Karhumäki. The Ehrenfeucht conjecture: a compactness claim for finitely generated free monoids. *Theor. Comput. Sci.*, 29:285–308, 1984.

62 Dexter Kozen. Lower bounds for natural proof systems. In *Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 254–266. IEEE, 1977.

63 Christof Löding and Christopher Spinrath. Decision Problems for Subclasses of Rational Relations over Finite and Infinite Words. *Discrete mathematics and theoretical computer science*, 2019.

64 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

65 Albert R. Meyer and Larry J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society, 1972.

66 Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

67 Maurice Nivat. Transduction des langages de Chomsky. *Annales de l'Institut Fourier*, 18:339–455, 1968.

68 Jean-Eric Pin. Logic, Semigroups and Automata on Words. *Annals of Mathematics and Artificial Intelligence*, 16:343–384, 1996.

69 Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, pages 114–125, 1959.

70 Christophe Reutenauer and Marcel-Paul Schützenberger. Minimization of rational word functions. *SIAM Journal of Computing*, 20(4):669–685, 1991.

71 Sasha Rubin. Automata Presenting Structures: A Survey of the Finite String Case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

72 Jacques Sakarovitch and Rodrigo de Souza. Lexicographic decomposition of K-valued transducers. *Theory Comput. Sci.*, 47:758–785, 2010.

73 Marcel-Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4(2-3):185–196, 1961.

74 Marcel-Paul Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8:190–194, 1965.

75 Marcel-Paul Schützenberger. Sur les relations rationnelles. In *Proc. of 2nd GI conference, Automata Theory and Formal Languages*, number 33 in LNCS, pages 209–213. Springer, 1975.

76 Marcel-Paul Schützenberger. Sur les relations rationnelles entre monoïdes libres. *Theor. Comput. Sci.*, 3(2):243–259, 1976.

77 John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

78 Richard E. Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, grammars and automata. In *Annual Symposium on Foundations of Computer Science (FOCS'81)*, pages 74–81, 1981.

79 Moshe Y. Vardi. A Note on the Reduction of Two-Way Automata to One-Way Automata. *Information Processing Letters*, 30:261–264, 1989.

80 Andreas Weber. Decomposing A k-Valued Transducer into k Unambiguous Ones. *RAIRO-ITA*, 30(5):379–413, 1996.

81 Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, pages 327–340, 1995.

82 Thomas Wilke. Classifying Discrete Temporal Properties. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *LNCS*, pages 32–46. Springer, 1999.

Algorithmic Data Science

Petra Mutzel
TU Dortmund, Department of Computer Science, Otto-Hahn-Str. 14, 44221 Dortmund, Germany
https://ls11-www.cs.tu-dortmund.de
petra.mutzel@cs.tu-dortmund.de

Abstract

The area of algorithmic data science provides new opportunities for researchers in the algorithmic community. In this paper we will see examples that demonstrate that algorithm engineering is the perfect basis for algorithmic data science. But there are also many open interesting questions for purely theoretically interested computer scientists. In my opinion, these opportunities should be taken because this will be fruitful for both areas, algorithmics as well as data sciences. I like to call for more participation in algorithmic data science by our community. Now we have the opportunity to shape this new emerging field.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Complexity theory and logic; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Algorithmic Data Science, Graph Similarity, Weisfeiler-Leman

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.3

Category Invited Talk

Funding This work has been supported by the German Science Foundation (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Data Analysis", project A6 "Resource-efficient Graph Mining".

1 Introduction

"Data is the new oil. It's valuable, but if unrefined it cannot really be used." – You may have read this quote, originally phrased by the Mathematician Clive Humby in 2006, several times. However, more than 12 years later, it is still relevant.

The ongoing digital transformation of business and society also affects science. The incoming amount of data that is stored and communicated is still increasing. Autonomously driving cars are not only recording data by various sensors but also sending their data (e.g., location, speed) to other cars that will be analysed in order to avoid critical situations. Smart home sensors are not only convenient but are very helpful for senior citizens who prefer to spend their life at home instead of institutional care. There are many other application domains, e.g., social media interaction, web mining, and video streaming systems. Concerning science, data analysis already became essential in many areas, e.g., biology, chemistry, medicine, neuroscience, linguistics, and geography.

Data science is the field responsible for extracting information out of (unstructured) data. This includes data integration, data cleaning, mathematical modelling, data analysis, and visualisation. Originally settled in the field of statistics, with increasing data sizes, also computer science and applied mathematics became increasingly relevant to the field. However, in contrast to common believe, for the analysis of the data, not only machine learning knowledge is needed. The area of data science is very broad, and not all the tasks need statistical concepts or machine learning. There are many problems for which fundamental and deep knowledge of theoretical aspects of computer science is needed.

In this paper we will see that data analysis provides many opportunities for researchers based in theoretical computer science as well as in algorithmic theory and algorithm engineering. However, there is only a very small overlap of researchers having published in both, leading theoretical computer science venues and data mining venues. The author is convinced that the field of data science will need more researchers from the theoretical computer science community. Experience shows that when co-authors from both communities join their forces, the outcome is quite profitable. For example, very recently, the mixed team Ben-David, Hrubeś, Moran, Shpilka, and Yehudayoff [8] has shown that simple scenarios of learnability are undecidable using the standard axioms of mathematics. For their proof they used the equivalence between learnability and compression. The purpose of this paper is to call for more activity of our community concerning data science tasks. In order to be more visible and to make our algorithmic basis clear, we could, e.g., start an initiative and make the name *algorithmic data science* our own.

The paper is organised as follows. Section 2 motivates the need for algorithmic data science performed by theoretical computer scientists and provides a brief introduction into data science and its main analysis tasks. Section 3 is more specific and gives an introduction to the wide area of graph similarity. The author has chosen this topic, since it is quite important in data science and machine learning, and it has many relations to different fields, since it connects graph algorithms with graph theory, algebra, descriptive logic, pebble games, linear programming relaxations, and functional analysis. Last but not least, there are plenty of interesting open questions.

2 Motivation

The vast majority of publications in data mining conferences and journals is applied. Similarly to algorithm engineering, in the top ranked publication venues also some kind of theory is mandatory. The area of algorithm engineering is relatively close to algorithmic data science, since it is about the design, theoretical analysis, implementation, and experimental evaluation of algorithms and data structures. The focus is on solving a real problem with realistic data provided by practitioners. Algorithm engineering researchers are always interested in the application of their research results. In order to be successful (in publishing at high ranked algorithmic venues) a solid basis on fundamental theoretical algorithmic knowledge is needed. It is important to be able to assess the limits of what is feasible and, if this does not meet the practical requirements, to be able to extend feasibility by means of guaranteed approvals. Alternatively, the structure of the data may help for making an approach efficient or sometimes also the change of the mathematical model of the problem. Sometimes this even leads to better practical and theoretical methods (see, e.g., subsection 3.1). Algorithmic data science essentially asks for the same. The aim is to develop new algorithms and tools for the analysis of the given data. Also for these tasks, a solid and deep knowledge of algorithmic concepts is needed. In both areas, close cooperation with the respective domain scientists is necessary.

Topics of interest for theoretical computer scientists

Blum, Hopcroft, and Kannan [9] recently published a book titled “Foundations of Data Science”, since they were convinced that the “emergence of the web and social networks as central aspects of daily life presents both opportunities and challenges for theory” ([9], p. 9). They state that in their book they “cover the theory we expect to be useful in the next 40 years” ([9], p. 9). This includes high-dimensional geometry, singular value decomposition,

random walks, sampling, sketching, streaming, clustering, graphical models, and foundations of machine learning.

In his book "Data Mining: The Textbook" [2] Aggarwal argues that the following four problems are fundamental to the process of analysing data: clustering, classification, association pattern mining, and outlier detection. However, an important basis for all of these are distances and similarity.

Clustering is the task to group the given data points so that items within the same group are more similar to each other than to outside items. Already from this definition it becomes clear that there are many possibilities to formally define the clustering problem. Clustering has been studied for a long time in the statistics community and in practical computer science (e.g., data bases, machine learning). Only recently, there is increasing interest by researchers in theoretical computer science (see, e.g., [1, 18]). Clustering has become even more popular during the big data hype, since it can be used for data sparsification and sampling approaches (e.g., [18]). Also for big data approaches on graphs, clustering plays an important role and became a research object of its own in the area of sublinear algorithms (e.g., [14]).

In contrast to clustering which belongs to so-called unsupervised learning, *classification* is a supervised learning approach. In classification a training set of data is provided whose items are labelled by its classes (groups). The aim is to train a classifier so that a new incoming item will be assigned to its correct class. Similar to clustering, also classification problems differ widely and have many applications in practice, e.g., in medicine for making decisions or predictions. Other popular applications are in spam detection. Classification is mostly studied in the machine learning community, while the related *regression problems*, where the labels are continuous, are mostly studied in statistics. However, this separation increasingly disappears. Popular approaches are support vector machines (e.g., in combination with kernel functions), k-nearest neighbour methods, logistic regression, Bayes classifiers, decision tree methods, and deep learning.

A popular data mining task is to find *frequent patterns* in a data set. A given parameter defines the *minimum support* s. An item is said to be frequent if it occurs at least s times. For example, in a graph G the set of (frequent) subgraphs (often restricted to a certain size) are those appearing at least s times in G. In a (temporal) sequence of data the considered patterns are subsequences. The aim is to find inherent regularities in the data. This is useful for deriving similarity measures on given data sets, which is important for clustering, classification, and outlier detection. *Association pattern mining* denotes a generalisation of frequent pattern mining, since it does not only rely on absolute frequencies but also on other statistical quantifications leading to association rules from a statistical perspective. The *confidence of a rule* $A \Rightarrow B$ is defined as the fraction of data points containing A that also contain B. The algorithmic aspects of frequent pattern mining problems have been studied widely in the early data mining literature. In their book [3], Aggarwal and Han provide a great survey on the current state of this topic.

Outlier detection is the problem of finding data items that are different from the majority of the given data set. Outliers may reflect errors in the data or they may belong to certain interesting rare events (e.g., in physics). Applications include fraud detection, network intrusion detection, finding unusual symptoms in medicine, or measuring errors from sensors. Also for this task, many models and approaches have been developed so far. There are statistical models (e.g., statistical tests), models based on spatial proximity (e.g., k-nearest neighbour), density-based techniques, ensemble techniques, and many more. The three above mentioned data analysis tasks clustering, classification, and association pattern mining can be used for finding outliers.

All of the above tasks rely on a notion of similarity or dissimilarity. For vectorial data or spatial data it is natural to use distance functions (two points are close to each other if their distance is small), whereas for graphs the notion of similarity is more common. In this context, also fundamental algorithmic techniques and data structures such as finding nearest neighbours and locality-sensitive hashing are useful.

It is important to develop special methods for certain data domains. For text data, pattern mining algorithms as well as compressed data structures are relevant. Of increasing interest are sequence data and time-series data recorded by sensors. For these data types, topics like sequence similarity, time series forecasting, classification, motifs, clustering, and outlier detection are of interest. Streaming algorithms are essential for dealing with continuous data streams.

Spatial data appears in the public health sector, energy and water supplies, smart cities, and many other domains. Examples are, e.g., remotely-sensed satellite images for weather forecast, climate, or crops. In summer 2018, the University Consortium for Geographic Information Science (UCGIS) published a call to action for bringing the geospatial perspective into data science degrees and curricula [47].

A natural representation for linked data sets are graphs. Some applications lead to sequences of large graphs, such as the web graph, while others to large numbers of small static graphs as in chemical molecule databases. One example of successful algorithmic data science where both communities, the data mining and the algorithmic community merged, is the computation of distances in a graph, which is important for social network analysis (e.g., centrality metrics). Other topics such as connectivity, matchings, network design and partitioning problems have not yet been brought to data science attention, although there will be potential. Concerning the classical basic graph algorithms, Skiena states: "I have not seen these tools applied as generally in data science as I think they should be" ([45], p. 323). A possible reason for this may be that the graphs tend to be quite large. On the other hand, this maybe overcome with graph sketching, graph sparsification, graph sampling, sublinear graph algorithms, and network summarisation. All of these are topics studied in the theoretical computer science community and of interest to data science.

3 Graph Similarity

The basis for many data analysis tasks for graphs and networks like clustering, classification and outlier analysis is graph similarity. One can think of many different models and definitions for this depending on the current applications. Although some similarity notions have been studied from practical and theoretical perspectives already, there are still many open questions.

3.1 Isomorphism based Approaches

Graph similarity is closely related to graph isomorphism. In the data mining literature, the notion of *exact graph matching* is devoted to find strict correspondences between two graphs being matched, or at least their subgraphs.

Graph isomorphism

Obviously, two graphs are most similar if they are isomorphic to each other.

▶ **Definition 1.** *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two simple graphs. A bijective map $\pi : V_G \to V_H$ is a* graph isomorphism *if the following holds:*

$\forall\, v, w \in V_G : (v, w) \in E_G \iff (\pi(v), \pi(w)) \in E_H$.

The graph isomorphism problem *asks the question if a graph isomorphism between two given graphs exists.*

The complexity of the graph isomorphism problem is still open. NP-completeness of the problem would result in the collapse of the polynomial time hierarchy to its second level [41]. There are quite a few special graph classes for which the graph isomorphism problem is known to be solvable in polynomial time such as planar graphs [26], bounded degree graphs [33], and graphs of bounded tree width [10]. A quite general result by Grohe and Marx shows that the graph isomorphism problem can be solved in $O(n^{f(|H|)})$ time for H-topological-minor-free graphs [22], where f denotes a function. For a long time, fixed-parameter-tractability of the graph isomorphism problem concerning the parameter tree width was open. Recently, Lokshtanov et al. [32] solved the problem by suggesting a graph canonisation approach leading to a $2^{O(k^5 \log k)} n^5$ time algorithm that either solves the graph isomorphism problem for two given graphs or concludes that one of the graphs does not have tree width bounded by k. Grohe et al. [23] have improved this result to an isomorphism test for graphs with tree width k with running time $O(2^{k\ \mathrm{polylog}(k)}\mathrm{poly}(n))$.

The theoretically best algorithm for general graphs known is the quasi-polynomial time algorithm by Babai [6]. Surprisingly, most practical instances on general graphs can be solved quite fast. However, most of the graph pairs provided for graph analysis are not isomorphic to each other and we are interested in their similarity.

Maximum common subgraph

A natural extension of graph isomorphism to graph similarity between G and H is to search for the largest subgraph that is contained in G and H. This gives rise to the *maximum common subgraph problem.*

▶ **Definition 2.** *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two simple graphs. A graph C is called a* common subgraph *if there exist two vertex sets $R \subseteq V_G$ and $S \subseteq V_H$ so that the induced subgraphs $G[R]$ and $H[S]$ are isomorphic to C. The common subgraph of largest size is called the* maximum common subgraph.

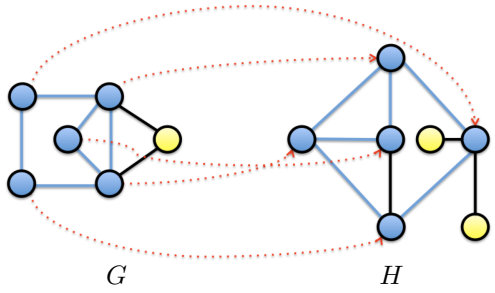


Figure 1 Two graphs and their maximum common subgraph displayed with blue vertices. The red darts demonstrate the mapping between the two isomorphic subgraphs.

Figure 1 shows two graphs and their maximum common subgraph. In contrast to the graph isomorphism problem, the maximum common subgraph problem is well-known to be NP-hard. Also here many variants are possible by asking for non-induced subgraphs

or introducing weights to the vertices and edges. Kann studied the complexity of several variations of the maximum common subgraph problem including approximation [27]. In practice, researchers often use the relationship between the maximum common subgraph problem and the maximum clique problem in the product graph of $G = (V_G, E_G)$ and $H = (V_H, E_H)$. If the graphs are small, then the algorithm by Bron and Kerbosch [12] can be used which enumerates the set of all maximal cliques in a graph. For larger graphs, this can be combined with branch-and-bound techniques. The theoretical fastest clique algorithm by Robson [40] leads to running time $O(2^{0.249|V_G||V_H|})$. However, this algorithm has not been published so far and is quite involved. Therefore, the result by Fomin, Grandoni, and Kratsch [19] is of interest for practitioners, since it introduces a simple algorithm with running time $O(2^{0.288|V_G||V_H|})$. In [30] Kriege suggested an algorithm based on graph canonisation with running time $O(2^{V_H+V_G^{1/2+o(1)}})$ assuming that $|V_G| \leq |V_H|$.

Concerning special graph classes, the maximum common subgraph problem remains NP-hard even if both given graphs are trees [11]. However, if the output is restricted to be connected, then this problem can be solved in polynomial time [35]. Akutsu and Tamura [4] have shown that the maximum common connected subgraph problem is NP-hard in vertex-labeled partial 11-trees of bounded degree. Kriege et al. [28] have shown that the problem remains NP-hard in biconnected partial 2-trees with all but one vertex of degree three or less.

Figure 2 The structure of the caffeine molecule and its molecule graph with vertex labels (atoms) and edge attributes (single or double bonds).

In practice, this problem is highly relevant for chemical molecule databases used for drug design. Molecule graphs are often outerplanar, almost all of them are planar. They have bounded tree width and vertex degree. Very important are the vertex and edge labels corresponding to atoms and activity attributes (see Fig. 2). Chemists want to find small molecules having a similar function as a given molecule or they want to conduct high-throughput screening in order to find promising candidates. Graph similarity approaches work well for answering these type of questions, since there is a direct connection between the structure (atoms and their bonds) and the effect of a molecule. When studying the chemical problem, it turned out that a restricted version of the maximum common subgraph problem which preserves the blocks (maximal biconnected components) and the bridges of the input graphs, is even of more relevance to the chemists. Luckily, this restricted version can be solved in polynomial time [28] in contrast to the original stated problem. For outerplanar bounded degree graphs, the block-and-bridge preserving maximum common connected subgraph can be computed in quadratic time [16]. These examples show that looking at the practical data as well as analysing the given practical problem may often help to find theoretically and practically useful algorithms. A new direction relevant to molecular graphs is to further relax the restriction of isomorphism and instead only require a homeomorphism (see, e.g., [17]). Recently, Kriege, Humbeck and Koch [29] have provided a survey on chemical similarity and substructure search in the area of drug design.

3.2 Distance based Approaches

Distance based approaches for graph comparison have been used widely in the machine learning community and in certain application domains like bioinformatics.

Frobenius distance

A natural distance measure between two graphs G and H is to search a permutation of the vertex set of G so that the number of edge mismatches is minimised. The *Frobenius distance* between two graphs investigated in [25] takes up this idea. Here, a permutation π of the rows and columns of the adjacency matrix A_G is searched that minimises the Frobenius norm of the matrix $A_G^\pi - A_H$. Although this problem has been extensively studied in the machine learning literature (also called *graph matching*) only few theoretical results are known. Recently, Grohe, Rattan, and Woeginger [25] have investigated the complexity of this and related problems. They have shown that this graph similarity problem is NP-hard even if both input graphs are trees or if one input graph is a path. On the other hand, the authors show that in the case that the two graphs are a path and a tree, the problem can be solved in polynomial time. On the positive side, they also show that the weighted version (taking weights of the edges into account) of the graph similarity problem related to the Frobenius norm is tractable if both adjacency matrices are positive semidefinite and have bounded rank, and where one of the matrices has a bounded clustering number. For details, please see [25]. Many problems in this area are worth further studying. It would be of interest if the problem or some restrictions can be approximated in polynomial time. Also results concerning fixed-parameter-tractability would be of great interest for algorithmic data science.

Graph edit distance

A more general distance is the *graph edit distance* in which the goal is to transform graph G into graph H by adding, substituting or deleting vertices and edges with the smallest total cost. This very general problem is often used in bioinformatics (e.g., for sequence comparisons), since it has the advantage that arbitrary vertex and edge attributes (e.g., labels) and many different cost functions and edit operations can be taken into account. However, this flexibility also comes with costs. The graph edit distance is a generalisation of the maximum common subgraph problem, which is NP-complete and hard to approximate with given guarantees [27]. Practical approaches for computing the graph edit distance of two given graphs are often based on backtracking or tree search algorithms and work for small graphs only. Recently, a binary linear programming formulation for computing the graph edit distance has been proposed (Lerouge et al., 2017), which allows to compare graphs of moderate size using state-of-the art general purpose solvers.

3.3 Weisfeiler-Leman Approaches

Despite the fact that the complexity of the graph isomorphism problem is open, experience shows that the problem can be solved quite fast in practice for most instances. The basis for many of the practical algorithms (e.g., nauty [36]) as well as for the quasi-polynomial time algorithm by Babai [6] is the vertex colouring algorithm by Weisfeiler-Leman[1]. The hypothesis

[1] In the literature found as Lehman as well as Leman

that similar graphs tend to have similarly coloured vertex sets led to its usage for classification tasks. For many practical tasks, Weisfeiler-Leman based classification (also called *colour refinement*) successfully competes with or even dominates the best state-of-the-art methods.

Basic Weisfeiler-Leman

The *Weisfeiler-Leman algorithm* (WL) simultaneously colours the vertices of the two given graphs iteratively. In the beginning, all vertices get the same colour c. In each iteration, the vertex sets of each colour class are further separated. This is done by looking at the neighbours of each vertex. E.g., if a vertex v has three neighbours of a colour c while vertex w has only two neighbours of colour c, then v and w will get different colours. The algorithm stops if the colour classes do not change anymore. If now the two given graphs have different colour classes, we know that these graphs cannot be isomorphic to each other.

Figure 3 shows two graphs and their colouring after the second iteration. By looking at the colour histograms of each graph, which are different for our example, it becomes clear that the two graphs cannot be isomorphic to each other. So we do not even need to compute further iterations. We say that the Weisfeiler-Leman algorithm *distinguishes the two graphs* if the colour patterns are different.

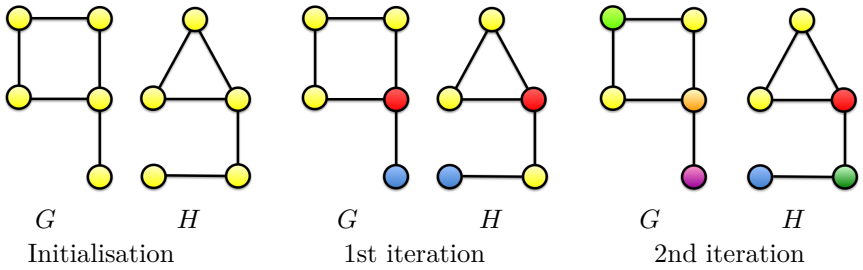
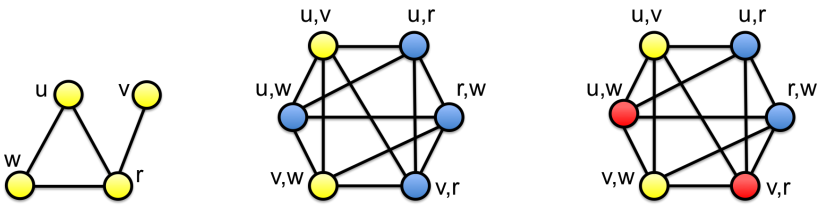


Figure 3 The first iterations of the WL-algorithm for two graphs G and H.

In the case that the colour classes are identical after the final round of the Weisfeiler-Leman algorithm, we cannot be sure if both graphs are isomorphic to each other. Consider, for instance, a k-regular graph (all vertices have degree k). The algorithm would stop after the first round, since every vertex has the same number of neighbours of colour c. Hence all vertices get the same colour. Hence, the WL algorithm can be used as a heuristic with one-sided error for solving the graph isomorphism problem.

WL has the nice property that two random graphs will end up with different colour classes with high probability [7]. In data analysis, the algorithm is been used for solving classification problems via graph kernels. Graph kernels have been used with established learning algorithms such as support vector machines and have proven to be a key technique for solving classification and prediction tasks on graphs [43].

A graph kernel is a similarity measure between graphs, which can be represented as a dot product between feature vectors obtained from the graphs. The colour histograms after every round of the WL-algorithm directly provide such a feature vector. E.g., in our example the feature vector after the first round for G would be $(3, 1, 1)$ (three yellow, one red, one blue) and after the second round $(2, 0, 0, 1, 1, 1, 0)$ (because there is no red, blue or dark green coloured vertex in G). In order to get one feature vector, we can simply concatenate the two feature vectors, hence we get $(3, 1, 1, 2, 0, 0, 1, 1, 1, 0)$. We could also scale some of these vectors up or down. The similarity measure is then given by the dot product of these feature vectors. By doing this for all pairs of graphs in a given data set, we get a similarity function that can be plugged into a learning algorithm, such as a support vector machine.



G initialisation 2-WL 1st iteration 2-WL

Figure 4 An example for the first rounds of 2-WL (set-based) for a graph G.

Higher dimensional Weisfeiler-Leman approaches

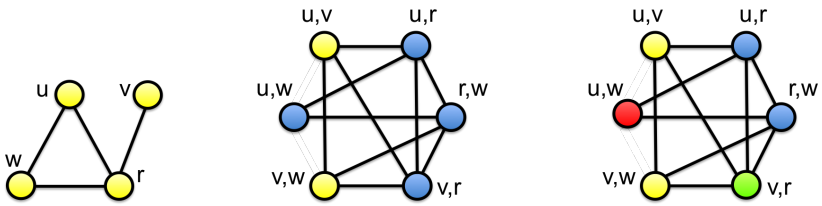
We get a stronger version of the WL algorithm by colouring the set of all k-tuples or k-sets of vertices. There are many different possibilities to generalise the WL approach to k dimensions by defining the neighbourhood of the elements. So be aware about the different definitions in the literature, in particular, if results from other papers concerning the k-WL are used. Most results may be true for the various definitions, but not all of them.

For simplicity we discuss the version in which we consider the set of all unordered k-sets instead of tuples. Then we say that two k-sets of vertices are neighbours if they differ in exactly one element. Initially, the k-sets R and S get the same colour if the induced graphs $G[R]$ and $H[S]$, respectively, are isomorphic to each other. In each iteration two k-sets R and S of the same colour get different colours, if there exists a colour c for which R and S have a different number of neighbours coloured c. Figure 4 shows an example for $k = 2$.

It turns out that, in general, the k-WL algorithm is stronger than the original WL-algorithm in the sense that it is able to distinguish two non-isomorphic graphs (answer "not isomorphic") whenever the original is able to do so. Moreover, for large enough k, the generalised approach would be able to solve the graph isomorphism problem. However, note that already the initialisation phase for $k = n$ asks for solving the graph isomorphism problem. Babai in his quasi-polynomial algorithm uses the k-tuple WL for $k = O(\log n)$ and many other involved algebraic techniques [6]. Cai, Führer, and Immerman [13] have shown that for every k there exist 3-regular graphs G_k and H_k of size $O(k)$ that are not distinguishable by the k-WL. Altogether we can say that the k-WL for $k \geq 2$ is quite strong, but it is also slow to compute.

Because the k-WL algorithm is too slow for using it for classification tasks in data analysis, we have suggested a local version which takes the graph structure into account [37]. Here, we say that two k-sets are neighbours if they differ in exactly one element and there is an edge from a vertex in R (resp. S) to a vertex in S (resp. R). So our new local kernel takes both, local and global graph properties into account. Since for sparse graphs the neighbourhood of a k-set in the local WL is much smaller compared to the neighbourhood in the original k-WL, the algorithm runs much faster on such graphs. Our experiments on several graph classification benchmarks have shown that our kernels often outperform the state-of-the-art in terms of classification accuracies.

Surprisingly, in our experiments, our local version concerning k-sets (we tested for $k = 2, 3$) was at least as strong as the global k-WL and often even stronger. Also the number of colour classes of our local version is in general larger than that of k-WL (see, e.g., Fig. 5). This is nice, since in the best case, every vertex gets a different colour; and then it is easy to distinguish two non-isomorphic graphs. Currently, we are investigating the relationships more deeply; our theoretical as well as new empirical results can be found in [38]. Observe that there are quite a few different definitions of the k-WL in the literature, which differ in their strength. For more information, see, e.g., [21].



G initialisation 2-LWL 1st iteration 2-LWL

Figure 5 An example for the first rounds of the local 2-LWL (set-based) for a graph G.

Linear programming provides a nice relationship between graph isomorphism and the outcome of the Weisfeiler-Leman algorithm. There is a natural integer linear program for the graph isomorphism problem in which the nonnegative integer solutions of the equation system correspond to permutation matrices. Tinhofer [46] has denoted the *nonnegative fractional* solutions of the LP-relaxation of this integer linear program *fractional isomorphisms*. He has shown that two graphs are fractional isomorphic if and only if the Weisfeiler-Leman algorithm is not able to distinguish them.

Weisfeiler-Leman and its relation to descriptive complexity

Cai, Han, and Führer [13] have revealed interesting connections between the k-WL and descriptive complexity. They provide a linear lower bound for the number of variables needed for first-order logic with counting to distinguish a sequence of pairs of graphs G_n and H_n. The authors prove the equivalence of the k-variable language with counting and the $(k-1)$-dimensional WL. Then they introduce combinatorial pebble games and prove that they are logically equivalent in the considered languages.

Atserias and Maneva [5] provide an equation system related to the generalisation of the Weisfeiler-Leman algorithm to k-tuples which is in close relation to the level-k Sherali-Adams relaxation of Tinhofer's equation system. More precisely, they show that the levels of the Sherali-Adams hierarchy of linear programming relaxations applied to Tinhofer's equation system interleave with the levels of k-dimensional Weisfeiler-Leman, and in addition with the levels of indistinguishability in a logic with counting quantifiers and bounded number of variables. The former results have also been obtained by Malkin [34] using polyhedral arguments. Grohe and Otto [24] have further simplified the arguments and strengthened the above result using a modified k-pebble counting game.

Counting homomorphisms

In the graph mining community, graph similarity is often measured by counting small subgraphs. For example, for a given input graph, the number of triangles, paths of length 4, and subtrees of certain sizes and structure are counted. These counts provide the input values for a feature vector for this graph. The dot product yields a graph kernel which can then be used for graph classification using a support vector machine (e.g., see [44]).

Dell, Grohe, and Rattan [15] suggest to count homomorphism vectors restricted to certain subgraphs instead. Their motivation is a classical result due to Lovász stating that a graph G can be characterised by counting the homomorphisms from the set of all graphs F to G. The authors show that if the homomorphism vectors are restricted to trees, then the feature vectors of the homomorphism counts of two graphs are identical if and only if the Weisfeiler-Leman algorithm does not distinguish the graphs. The LP-relaxation of the natural

integer linear program for the graph isomorphism problem has a rational solution if and only if the two feature vectors of the homomorphism counts are identical [15]. Moreover, the authors generalised their result to the k-dimensional Weisfeiler-Leman algorithm. For this, they restrict the homomorphism counts to the graph class of all graphs of tree width at most k. Then they show that the following three statements are equivalent:

(i) The feature vectors of the homomorphism counts restricted to the graph class of graphs with tree width at most k of two given graphs are identical.
(ii) The k-dimensional Weisfeiler-Leman algorithm does not distinguish both graphs.
(iii) The system of linear equations has a nonnegative solution.

The following results have been proven for one direction only: In the case that the system of linear equations has a real solution, then the feature vectors of the homomorphism counts restricted to the graph class of graphs with path width at most k of two given graphs are identical. It is still open if the converse holds. The authors state that it would be very interesting to study the graph similarity measures induced by the homomorphism vectors [15].

Weisfeiler-Leman and deep learning

Deep neural networks are incredibly popular these days, since they have led to qualitative breakthroughs on a wide variety of tasks. They are among the most successful machine learning approaches for voice recognition and image recognition. Today they are used for almost any application related to sound, text, images, videos, graphs, and time series. Although there is plenty of successful empirical research on a wide variety of applications, there are only few theoretical papers explaining their behaviour.

Gilmer et al. [20] have suggested a *message passing neural network* for graphs with vertex and edge features which covers most of the current *graph neural networks*. Similar to Weisfeiler-Leman, also these graph neural networks are based on neighbourhood aggregation. In contrast to WL, the aggregation functions do not need to be discrete, and the architecture of the T-layered network defines the neighbourhood.

Morris et al. [39] have studied the relationship between Weisweiler-Leman and graph neural networks. The authors have shown that graph neural networks can be viewed as a neural version of the Weisfeiler-Leman algorithm, in which the colours are replaced by continuous feature vectors. The neural networks aggregate over the vertex neighbourhoods. The paper shows that although the graph neural networks are more flexible concerning learning tasks, they are not able to distinguish pairs of non-isomorphic graphs that cannot be distinguished by Weisfeiler-Leman. On the other hand, there exist architectures and parameters so that graph neural networks have the same strength as Weisfeiler-Leman. Based on their observation, the authors have suggested so-called *k-graph neural networks* as well as new hierarchical versions, and prove an equivalence concerning strength to the k-dimensional Weisfeiler-Leman approach. Their experimental study has revealed that their new hierarchical approach is able to dominate the state-of-the-art approaches. This a nice example where theory does lead to innovative approaches improving the practical state-of-the-art.

4 Conclusion

This paper tries to motivate the readers to get interested to the area of algorithmic data science. There are plenty of opportunities for achieving new theoretical results as well as practical impact. The paper should not be seen as a survey on the area of data analysis on graphs but rather as an attempt to get the reader interested in algorithmic data science. If you want to learn a bit more, you can find some suggestions in the following.

Skiena [45] wrote a great book that gives a nice introduction into data science focusing on the skills and principles needed for the whole process including data integration, data cleaning, data analysis and visualisation. In particular, the author provides intuition for the presented statistical and mathematical concepts. The recent books by Blum, Hopcroft, and Kannan [9] as well as Aggarwal [2] dig deeper into theory (see section 2). Leskovec, Rajaraman and Ullman [31] cover certain aspects concerning Web mining including the technology of search engines like link-spam detection and recommendation systems. Other interesting books are, e.g., [48] and [42].

References

1 Marcel R. Ackermann, Johannes Blömer, Daniel Kuntze, and Christian Sohler. Analysis of Agglomerative Clustering. *Algorithmica*, 69(1):184–215, 2014. doi:10.1007/s00453-012-9717-4.

2 Charu C. Aggarwal. *Data Mining: The Textbook*. Springer Publishing Company, Incorporated, 2015.

3 Charu C. Aggarwal and Jiawei Han. *Frequent Pattern Mining*. Springer Publishing Company, Incorporated, 2014.

4 Tatsuya Akutsu and Takeyuki Tamura. A Polynomial-Time Algorithm for Computing the Maximum Common Subgraph of Outerplanar Graphs of Bounded Degree. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *37th International Symposium on Mathematical Foundations of Computer Science, MFCS 2012*, pages 76–87, Berlin, Heidelberg, 2012. Springer.

5 Albert Atserias and Elitza Maneva. Sherali–Adams Relaxations and Indistinguishability in Counting Logics. *SIAM Journal on Computing*, 42(1):112–137, 2013. doi:10.1137/120867834.

6 László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, STOC 2016*, pages 684–697, New York, NY, USA, 2016. ACM.

7 László Babai, Paul Erdös, and Stanley M. Selkow. Random Graph Isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980. doi:10.1137/0209047.

8 Shai Ben-David, Pavel Hrubes, Shay Moran, Amir Shpilka, and Amir Yehudayoff. Learnability can be undecidable. *Nature Machine Intelligence*, 1:44–48, 2019. doi:10.1038/s42256-018-0002-3.

9 Avrim Blum, John Hopcroft, and Ravi Kannan. Foundations of Data Science, 2018. URL: https://www.cs.cornell.edu/jeh/book.pdf.

10 Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990. doi:10.1016/0196-6774(90)90013-5.

11 Franz J. Brandenburg. Subgraph isomorphism problems for k-connected partial k-trees. Unpublished manuscript, 2000.

12 Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. In *CACM*, 1973.

13 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identificationS. *Combinatorica*, 12(4):389–410, 1992.

14 Artur Czumaj, Pan Peng, and Christian Sohler. Testing Cluster Structure of Graphs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 723–732. ACM, 2015. doi:10.1145/2746539.2746618.

15 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:14,

Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.40.

16 Andre Droschinsky, Nils Kriege, and Petra Mutzel. Finding Largest Common Substructures of Molecules in Quadratic Time. In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, editors, *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science*, volume 10139 of *Lecture Notes in Computer Science*, pages 309–321. Springer, 2017. doi:10.1007/978-3-319-51963-0_24.

17 Andre Droschinsky, Nils M. Kriege, and Petra Mutzel. Largest Weight Common Subtree Embeddings with Distance Penalties. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, pages 54:1–54:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.54.

18 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1434–1453. SIAM, 2013. doi:10.1137/1.9781611973105.103.

19 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and Conquer: A Simple $O(2^{0.288N})$ Independent Set Algorithm. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 18–25, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109560.

20 Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research (PMLR)*. PMLR, 2017.

21 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

22 Martin Grohe and Daniel Marx. Structure Theorem and Isomorphism Test for Graphs with Excluded Topological Subgraphs. *SIAM Journal on Computing*, 44(1):114–159, 2015. doi:10.1137/120892234.

23 Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. An Improved Isomorphism Test for Bounded-Tree-Width Graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.67.

24 Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

25 Martin Grohe, Gaurav Rattan, and Gerhard J. Woeginger. Graph Similarity and Approximate Isomorphism. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.20.

26 John E. Hopcroft and Joseph K. Wong. Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, STOC 1974*, pages 172–184, New York, NY, USA, 1974. ACM. doi:10.1145/800119.803896.

27 Viggo Kann. On the approximability of the maximum common subgraph problem. In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, pages 375–388, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

28 Nils Kriege, Florian Kurpicz, and Petra Mutzel. On maximum common subgraph problems in series-parallel graphs. *European Journal of Combinatorics*, 68:79–95, 2018. Combinatorial Algorithms, Dedicated to the Memory of Mirka Miller. doi:10.1016/j.ejc.2017.07.012.

29 Nils M. Kriege, Lina Humbeck, and Oliver Koch. Chemical Similarity and Substructure Searches. In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 640–649. Academic Press, Oxford, 2019. doi:10.1016/B978-0-12-809633-8.20195-7.

30 Nils Morten Kriege. *Comparing Graphs: Algorithms & Applications*. Phd thesis, TU Dortmund, 2015.

31 Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2 edition, 2014. doi:10.1017/CBO9781139924801.

32 Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Fixed-Parameter Tractable Canonization and Isomorphism Test for Graphs of Bounded Treewidth. *SIAM Journal on Computing*, 46(1):161–189, 2017. doi:10.1137/140999980.

33 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

34 Peter N. Malkin. Sherali-Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014.

35 David 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 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978. doi:10.1016/S0167-5060(08)70324-8.

36 Brendan D. McKay and Adolfo Piperno. Practical Graph Isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

37 Christopher Morris, Kristian Kersting, and Petra Mutzel. Glocalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs. In Vijay Raghavan, Srinivas Aluru, George Karypis, Lucio Miele, and Xindong Wu, editors, *IEEE International Conference on Data Mining (ICDM), 2017*, pages 327–336. IEEE Computer Society, 2017. doi:10.1109/ICDM.2017.42.

38 Christopher Morris and Petra Mutzel. Towards a practical k-dimensional Weisfeiler-Leman algorithm. Unpublished manuscript, 2019.

39 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. *CoRR*, abs/1810.02244, 2018. to appear at AAAI 2019. arXiv:1810.02244.

40 John M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université Bordeaux I, 2001.

41 Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988. doi:10.1016/0022-0000(88)90010-4.

42 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. doi:10.1017/CBO9781107298019.

43 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

44 Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 2009. PMLR. URL: http://proceedings.mlr.press/v5/shervashidze09a.html.

45 Steven S. Skiena. *The Data Science Design Manual*. Springer, 2017. URL: https://www.springer.com/de/book/9783319554433.

46 Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2):253–264, 1991.

47 University Consortium for Geographic Information Science. A UCGIS Call to Action: Bringing the Geospatial Perspective to Data Science Degrees and Curricula, 2018. URL: `https://www.ucgis.org/assets/docs/UCGIS-Statement-on-Data-Science-Summer2018.pdf`.

48 Mohammed J. Zaki and Meira Wagner Jr. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

Fine-Grained Complexity Theory

Karl Bringmann
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Abstract

Suppose the fastest algorithm that we can design for some problem runs in time $O(n^2)$. However, we want to solve the problem on big data inputs, for which quadratic time is impractically slow. We can keep searching for a faster algorithm, but maybe none exists. Is there any reasoning that provides evidence against significantly faster algorithms, and thus allows us to *stop searching*? In other words, is there an *analogue of NP-hardness for polynomial-time problems*?

In this tutorial, we will give an introduction to *fine-grained complexity theory*, which allows to rule out faster algorithms by proving *conditional lower bounds* via *fine-grained reductions* from certain *key conjectures*. We will define these terms and show exemplary lower bounds.

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

Keywords and phrases Hardness in P, conditional lower bound, fine-grained reduction

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.4

Category Tutorial

1 Introduction

The traditional way of establishing a problem as intractable is to prove it to be NP-hard. This makes a polynomial-time algorithm unlikely, so even for medium-size instances we cannot expect to solve the problem in reasonable time, at least on worst-case instances.

However, in a modern big data world with inputs such as DNA sequences, social network graphs, or sensor network measurings, even a quadratic-time algorithm can be too slow and essentially only near-linear-time algorithms are feasible. This shift necessitates changes in intractability theory. In order to avoid searching until eternity for a faster algorithm that does not exist, we need intractability tools that establish far-from-linear lower bounds. In other words, we need an analogue of NP-hardness for polynomial-time problems. Unfortunately, P vs. NP is too coarse to even differentiate between running time $O(n)$ and $O(n^{100})$, and no techniques for proving unconditional lower bounds higher than $\Theta(n \log n)$ are known.

Therefore, the modern approach is to prove *conditional lower bounds*. To this end, we start from a widely believed *conjecture*[1] about the time complexity of a key problem, and transfer the conjectured intractability to another problem via a *fine-grained reduction*, yielding a conditional lower bound on how fast the other problem can be solved. An exemplary conjecture is the Strong Exponential Time Hypothesis, which essentially states that any algorithm for Satisfiability requires time $2^{(1-o(1))n}$ in the worst case [31], see Section 2 for details. The resulting area of *fine-grained complexity theory*, also sometimes called *hardness in P*, had initial results in the early 90s [29], was heavily influenced by developments in the fixed-parameter tractability community [23, 27, 33], and started to mature in the last 5 years, with a wealth of publications appearing every year at the topmost theory conferences, see [1–4, 6–15, 17, 18, 20, 21, 25, 30, 34, 35, 39].

[1] Some authors prefer the terminology *hypothesis*.

The development of fine-grained complexity theory in particular enabled the *design of conditionally best-possible algorithms*: On the one hand, we design an efficient algorithm running in some time bound $T(n)$, on the other hand, we prove a conditional lower bound ruling out time $T(n)^{1-\delta}$ for any $\delta > 0$ using fine-grained complexity theory. Together, we determined $T(n)$ as the *best-possible* time complexity of our problem (up to lower order factors and conditional on a plausible complexity-theoretic assumption). This provides strong indication to stop searching for a faster algorithm.

Despite fine-grained complexity theory being a young field of research, conditionally best-possible algorithms have already been found for various problems. The fine-grained approach has been particularly successful for dynamic programming problems. For instance, assuming the Strong Exponential Time Hypothesis, quadratic running time is essentially optimal for similarity measures such as Edit distance [13], Longest Common Subsequence [3, 21], and Fréchet distance [18]. These developments have also fueled algorithmic improvements, e.g., the classic $O(nt)$-time algorithm for Subset Sum[2] from 1957 [16] has been improved to time $O(t \operatorname{polylog}(t))$ [19], which matches a SETH-based lower bound [5].

In this tutorial, we give an introduction to fine-grained complexity theory. An overview of the key conjectures is presented in Section 2. We introduce the notions of fine-grained reduction and conditional lower bound in Section 3. Then, in Sections 4 and 5 we show basic examples of fine-grained reductions. Finally, we conclude in Section 6.

2 Key Conjectures

The four most central conjectures of fine-grained complexity theory are as follows.

Satisfiability

Recall the standard k-SAT problem: We are given a formula on n Boolean variables in k-CNF (i.e., the formula is a conjunction of clauses, where each clause is a disjunction of at most k literals, and each literal is a negated or unnegated variable). The task is to decide whether there is a satisfying assignment (i.e., an assignment of each variable to true or false for which the formula evaluates to true). This problem can be solved naively in time $O(2^n n^k)$, by enumerating all 2^n assignments and checking each of at most $O(n^k)$ clauses. The *Strong Exponential Time Hypothesis* (SETH) states that the naive running time is essentially optimal when k tends to infinity. More formally, for any $\delta > 0$ there is a $k \geq 3$ such that k-SAT has no $O(2^{(1-\delta)n})$-time algorithm [31].

Orthogonal Vectors

Given sets A, B consisting of n vectors in $\{0,1\}^d$, decide whether there are vectors $a \in A, b \in B$ that are orthogonal (i.e., for any $1 \leq i \leq d$ we have $a[i] \cdot b[i] = 0$). This problem can be solved naively in time $O(n^2 d)$, by enumerating all pairs of vectors and checking orthogonality in time $O(d)$. It is conjectured that Orthogonal Vectors has no $O(n^{2-\delta} d^c)$-time algorithm for any $\delta, c > 0$. It is well-known that SETH implies the Orthogonal Vectors conjecture [41].

[2] Given a set X of n positive integers and a target t, does any subset of X sum to exactly t?

All Pairs Shortest Path

Given a graph G with n nodes and positive edge weights, compute for all pairs of nodes their shortest path distance. The classic Floyd-Warshall algorithm solves this problem in time $O(n^3)$ [28, 40]. It is conjectured that All Pairs Shortest Path has no $O(n^{3-\delta})$-time algorithm for any $\delta > 0$.

3SUM

Given a set X of n integers, decide whether there are $a, b, c \in X$ with $a+b+c = 0$. A folklore algorithm solves this problem in time $O(n^2)$. It is conjectured that there is no $O(n^{2-\delta})$-time algorithm for any $\delta > 0$.

We remark that for all of these problems, lower order improvements beyond the stated running times are known. For instance, All Pairs Shortest Path can be solved in time $n^3/2^{\Omega(\sqrt{\log n})}$ [42]. However, these improvements are not enough to falsify the conjectures.

3 Fine-Grained Reductions and Conditional Lower Bounds

For simplicity and since they form the majority of reductions in the literature, here we only consider many-one reductions, also known as Karp reductions. For a fine-grained variant of Turing reductions, see e.g. [37].

▶ **Definition 1.** *For problems P, Q and time bounds t_P, t_Q, a* fine-grained reduction *from (P, t_P) to (Q, t_Q) is an algorithm that, given an instance I of P, computes an instance J of Q such that:*

1. *I is a YES-instance of P if and only if J is a YES-instance of Q,*
2. *for any $\varepsilon > 0$ there is a $\delta > 0$ such that $t_Q(|J|)^{1-\varepsilon} = O(t_P(|I|)^{1-\delta})$, and*
3. *the running time of the reduction is $O(t_P(|I|)^{1-\gamma})$ for some $\gamma > 0$.*

In particular, if there is a fine-grained reduction from (P, t_P) to (Q, t_Q) and there is an algorithm for Q running in time $O(t_Q(n)^{1-\varepsilon})$ for some $\varepsilon > 0$, then by combining the two we can solve any instance I of P in time

$$O(t_P(|I|)^{1-\gamma} + t_Q(|J|)^{1-\varepsilon}) = O(t_P(|I|)^{1-\gamma} + t_P(|I|)^{1-\delta}) = O(t_P(|I|)^{1-\delta'}),$$

for some $\delta' > 0$. In other words, any significant improvement over time $t_Q(n)$ for Q yields a significant improvement over time $t_P(n)$ for P. Or, equivalently, if P cannot be solved significantly faster than in time $t_P(n)$, then Q cannot be solved significantly faster than in time $t_Q(n)$.

Suppose we have a fine-grained reduction from (All Pairs Shortest Path, n^3) to (Q, t_Q). Then problem Q cannot be solved in time $O(t_Q(n)^{1-\varepsilon})$ for any $\varepsilon > 0$ unless All Pairs Shortest Path can be solved in time $O(n^{3-\delta})$ for some $\delta > 0$, meaning that the All Pairs Shortest Path conjecture fails. In this situation, we say that we have proven a *conditional lower bound* of $t_Q(n)^{1-o(1)}$ for problem Q, assuming the All Pairs Shortest Path conjecture.

4 Example I: SETH-Hardness of Orthogonal Vectors

As a first example for a fine-grained reduction, we present the following by-now classic result.

▶ **Theorem 2** ([41]). *The Strong Exponential Time Hypothesis implies the Orthogonal Vectors conjecture.*

Proof. We show a fine-grained reduction from Satisfiability to Orthogonal Vectors. Specifically, given a k-CNF formula ϕ on n variables and m clauses, we will construct sets A, B of $N = 2^{n/2}$ vectors in dimension $D = m$.

Denote the clauses of ϕ by $C_1, \ldots, C_m$. We split the n variables into sets X, Y of size $n/2$. For any assigment α of X, we construct a corresponding vector $a(\alpha) \in A$, by setting its i-th coordinate to 0 if α satisfies clause C_i (i.e., if there is a literal in C_i that is set to true by applying assignment α to X), and setting it to 1 otherwise. Similarly, for any assigment β of Y, we construct a corresponding vector $b(\beta) \in B$, by setting its i-th coordinate to 0 if β satisfies clause C_i, and setting it to 1 otherwise. Observe that $a(\alpha)$ and $b(\beta)$ are orthogonal if and only if (α, β) forms a satisfying assignment of ϕ, since orthogonality means that each clause is satisfied by at least one of the two half-assignments α, β. Thus, we constructed an equivalent Orthogonal Vectors instance.

Note that we indeed constructed sets A, B consisting of $N = 2^{n/2}$ vectors in $\{0,1\}^D$ for $D = m$. Also note that we can construct A, B in time $O(ND)$. Now suppose that the Orthogonal Vectors conjecture fails, i.e., Orthogonal Vectors has an $O(N^{2-\varepsilon}D^c)$-time algorithm for some $\varepsilon, c > 0$. Then by combining this algorithm with our reduction, we can solve k-SAT on n variables and m clauses in time $O(2^{n/2}m + 2^{(2-\varepsilon)n/2}m^c) = O(2^{(1-\delta)n}m^{c'})$ for some $\delta > 0, c' \geq 1$. Since in k-CNF there are $O(n^k)$ different clauses, and we can bound the polynomial $O(n^k)$ by the exponential function $O(2^{\delta n/(2c')})$, we may estimate the factor $m^{c'}$ by $O(2^{\delta n/2})$. This yields a final time bound of $O(2^{(1-\delta/2)n})$ for k-SAT, which contradicts SETH. As contraposition, we obtain that SETH implies the Orthogonal Vectors conjecture. ◀

5 Example II: Regular Expression Pattern Matching

Let us recall the basics of regular expressions. A regular expression R is a search pattern that matches any string in the corresponding language $L(R)$. Here we only consider the following operations over a fixed alphabet Σ. For any symbol $c \in \Sigma$, the regular expression $R = c$ matches the length-1 string c, i.e., $L(R) = \{c\}$. For any regular expressions R_1, R_2, the regular expression $R = R_1|R_2$ matches any string matched by R_1 or R_2, i.e., $L(R) = L(R_1) \cup L(R_2)$. For any regular expressions R_1, R_2, the regular expression $R = R_1 \circ R_2$ matches any string that is a concatenation of a string matched by R_1 with a string matched by R_2, i.e., $L(R) = \{ab \mid a \in L(R_1), b \in L(R_2)\}$.

In the Regular Expression Pattern Matching problem, given a regular expression R and a text string T, the task is to decide whether any substring of T matches R. Denoting the length of T by n and the number of operations that define R by m, there is a classic $O(nm)$-time algorithm for this problem [36]. Here we show the following tight lower bound.

▶ **Theorem 3** ([14]). *Regular Expression Pattern Matching has no $O((n+m)^{2-\varepsilon})$-time algorithm for any $\varepsilon > 0$, unless the Orthogonal Vectors conjecture fails.*

Proof. For a fine-grained reduction from Orthogonal Vectors to Regular Expression Pattern Matching, consider an Orthogonal Vectors instance $A, B \subseteq \{0,1\}^D$ of size N. We construct a regular expression R and a text string T as follows.

On the coordinate level, we define two regular expressions

$$CR(1) := 0 \qquad \text{and} \qquad CR(0) := (0|1).$$

Note that for coordinates $x, y \in \{0,1\}$, regular expression $CR(x)$ matches string y if and only if $x \cdot y = 0$.

On the vector level, for any vector $b \in B$ we construct the regular expression

$$VR(b) := CR(b[1]) \circ CR(b[2]) \circ \ldots \circ CR(b[D]).$$

Note that for vectors $a, b \in \{0,1\}^D$, the regular expression $VR(b)$ matches the string $a[1]a[2]\ldots a[D]$ if and only if a and b are orthogonal.

On the level of sets of vectors, for the set $B = \{b_1, \ldots, b_N\}$ we construct the regular expression

$$R := VR(b_1)|VR(b_2)|\ldots|VR(b_N).$$

Note that for a vector $a \in \{0,1\}^D$, the regular expression R matches the string $a[1]a[2]\ldots a[D]$ if and only if a is orthogonal to some $b \in B$.

Finally, we define the text string T to be the concatenation of all vectors in A, padded by a dummy symbol '#', i.e., for $A = \{a_1, \ldots, a_N\}$ we set

$$T := a_1[1]\ldots a_1[D]\#a_2[1]\ldots a_2[D]\#\ldots\#a_N[1]\ldots a_N[D].$$

Since the dummy symbol does not appear in R, the regular expression R matches text T if and only if there is a vector $a \in A$ such that R matches $a[1]a[2]\ldots a[D]$. Hence, (R, T) is a YES-instance of Regular Expression Pattern Matching if and only if (A, B) is a YES-instance of Orthogonal Vectors.

Note that R and T have size $O(ND)$ and can be constructed in time $O(ND)$. Thus, any $O((n+m)^{2-\varepsilon})$-time algorithm for Regular Expression Pattern Matching for some $\varepsilon > 0$ would yield an algorithm for Orthogonal Vectors in time $O((ND)^{2-\varepsilon} + ND) = O(N^{2-\delta}D^c)$ for some $\delta, c > 0$, which contradicts the Orthogonal Vectors conjecture. Hence, Regular Expression Pattern Matching is not in time $O((n+m)^{2-\varepsilon})$ for any $\varepsilon > 0$ unless the Orthogonal Vectors conjecture fails. ◀

6 Conclusion

In this short introduction to fine-grained complexity theory we focused on the main conjectures and two basic examples. Over the last years, fine-grained complexity theory developed into a widely applicable tool with many interesting directions including extensions to dynamic graph algorithms [9], hardness of approximation [8], compressed data [1], external memory algorithms [26], and many more. Besides the four main conjectures presented here, several other conjectures have been used and relations among conjectures have been explored [4, 22, 24, 32].

For further reading, we refer to the surveys [37, 38]. Lecture material including slides can can be found at `https://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/summer16/poly-complexity/` (or linked on the author's homepage).

References

1 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-Grained Complexity of Analyzing Compressed Data: Quantifying Improvements over Decompress-and-Solve. In *FOCS*, pages 192–203, 2017.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *FOCS*, pages 98–117, 2015.

3 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *FOCS*, pages 59–78, 2015.

4 Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *STOC*, pages 253–266. ACM, 2018.

5 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for Subset Sum and Bicriteria Path. In *SODA*, 2019. To appear. See `https://arxiv.org/abs/1704.04546`.

6 Amir Abboud and Søren Dahlgaard. Popular Conjectures as a Barrier for Dynamic Planar Graph Algorithms. In *FOCS*, pages 477–486, 2016.

7 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends, or: a polylog shaved is a lower bound made. In *STOC*, pages 375–388, 2016.

8 Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. Distributed PCP Theorems for Hardness of Approximation in P. In *FOCS*, pages 25–36, 2017.

9 Amir Abboud and Virginia Vassilevska Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *FOCS*, pages 434–443, 2014.

10 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching Triangles and Basing Hardness on an Extremely Popular Conjecture. In *STOC*, pages 41–50, 2015.

11 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *STOC*, pages 239–252, 2018.

12 Divesh Aggarwal and Noah Stephens-Davidowitz. (Gap/S)ETH hardness of SVP. In *STOC*, pages 228–238, 2018.

13 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *STOC*, pages 51–58, 2015.

14 Arturs Backurs and Piotr Indyk. Which Regular Expression Patterns Are Hard to Match? In *FOCS*, pages 457–466, 2016.

15 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *STOC*, pages 267–280, 2018.

16 R.E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

17 Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. On the Quantitative Hardness of CVP. In *FOCS*, pages 13–24, 2017.

18 Karl Bringmann. Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *FOCS*, pages 661–670, 2014.

19 Karl Bringmann. A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum. In *SODA*, pages 1073–1084. SIAM, 2017.

20 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A Dichotomy for Regular Expression Membership Testing. In *FOCS*, pages 307–318, 2017.

21 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *FOCS*, pages 79–97, 2015.

22 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility. In *ITCS*, pages 261–270, 2016.

23 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

24 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michał Włodarczyk. On Problems Equivalent to (min, +)-Convolution. In *ICALP*, volume 80 of *LIPIcs*, pages 22:1–22:15, 2017.

25 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. In *STOC*, pages 281–288, 2018.

26 Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O Complexity via Reductions: New Lower Bounds, Faster Algorithms, and a Time Hierarchy. In *ITCS*, volume 94 of *LIPIcs*, pages 34:1–34:23, 2018.

27 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

28 Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

29 Anka Gajentaan and Mark H. Overmars. On a Class of $O(n^2)$ Problems in Computational Geometry. *Comput. Geom.*, 5:165–185, 1995.

30 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *STOC*, pages 21–30, 2015.

31 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

32 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In *ICALP*, volume 80 of *LIPIcs*, pages 21:1–21:15, 2017.

33 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

34 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC*, pages 515–524, 2013.

35 Barna Saha. Language Edit Distance and Maximum Likelihood Parsing of Stochastic Grammars: Faster Algorithms and Connection to Fundamental Graph Problems. In *FOCS*, pages 118–135, 2015.

36 Ken Thompson. Regular Expression Search Algorithm. *Commun. ACM*, 11(6):419–422, 1968.

37 Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk). In *IPEC*, volume 43 of *LIPIcs*, pages 17–29, 2015.

38 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *ICM*, 2018. To appear. See `https://people.csail.mit.edu/virgi/eccentri.pdf`.

39 Virginia Vassilevska Williams and Ryan Williams. Subcubic Equivalences between Path, Matrix and Triangle Problems. In *FOCS*, pages 645–654, 2010.

40 Stephen Warshall. A Theorem on Boolean Matrices. *J. ACM*, 9(1):11–12, 1962.

41 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

42 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.

From Graph Theory to Network Science: The Natural Emergence of Hyperbolicity

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
https://hpi.de/friedrich/
friedrich@hpi.de

Abstract

Network science is driven by the question which properties large real-world networks have and how we can exploit them algorithmically. In the past few years, hyperbolic graphs have emerged as a very promising model for scale-free networks. The connection between hyperbolic geometry and complex networks gives insights in both directions:

(1) Hyperbolic geometry forms the basis of a natural and explanatory model for real-world networks. Hyperbolic random graphs are obtained by choosing random points in the hyperbolic plane and connecting pairs of points that are geometrically close. The resulting networks share many structural properties for example with online social networks like Facebook or Twitter. They are thus well suited for algorithmic analyses in a more realistic setting.

(2) Starting with a real-world network, hyperbolic geometry is well-suited for metric embeddings. The vertices of a network can be mapped to points in this geometry, such that geometric distances are similar to graph distances. Such embeddings have a variety of algorithmic applications ranging from approximations based on efficient geometric algorithms to greedy routing solely using hyperbolic coordinates for navigation decisions.

2012 ACM Subject Classification Computing methodologies → Network science

Keywords and phrases Graph Theory, Graph Algorithms, Network Science, Hyperbolic Geometry

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.5

Category Tutorial

1 Introduction

In the field of algorithms, a major discrepancy between theory and practice derives from the fact that the analysis usually assumes worst-case instances. However, real-world instances behave very differently. In fact, numerous NP-hard problems can be solved in reasonable time even on large real-world instances, using techniques such as branch-and-bound [30], branch-and-reduce [5], or reductions to integer linear programming [26], which in itself is an NP-hard problem that is rather well-behaved on practical instances. One approach to bridge this gap between theory and practice is to employ an average-case analysis by bounding the expected run time under the assumption that the input is randomly drawn from a certain distribution. This was already pointed out by Karp [42] in 1983, who noted that: "One way to validate or compare imperfect[1] algorithms for NP-hard combinatorial problems is simply to run them on typical instances and see how often they fail. [...] While probabilistic assumptions are always open to question, the approach seems to have considerable explanatory power." In 1986, Levin [45] laid the foundation for average-case complexity theory by providing an average-case complete problem. For more on this topic, see the survey by Bogdanov and Trevisan [19].

[1] Karp calls an algorithm "imperfect" if it potentially outputs the wrong answer or runs too long.

Probability distributions on graphs

The practical relevance and "considerable explanatory power" [42] of an average-case analysis of course heavily depends on the assumed probability distribution of the input. Thus, when focusing on graph problems, the considered graph model should mimic at least the most important properties of typical real-world networks. Two fundamental properties that have been observed in large real-world networks of many domains are the following.

Heterogeneity. Large real-world networks are highly heterogeneous, i.e., they usually have some high-degree and many low-degree vertices. In fact, many real-world networks are *scale-free*, which means that the number of vertices of degree at most x is roughly proportional to $x^{-\beta}$ for some constant β. One then says that such graphs have a *power-law* degree distribution with *power-law exponent* β, which usually lies between 2 and 3.

Interdependency. Edges in real-world networks are typically not independent, i.e., vertices with a common neighbor tend to be rather similar and thus more likely to be connected than two random vertices. Formally, this can be measured using the so-called *clustering coefficient*, which comes in two flavors, local and global. The global clustering coefficient is the ratio of triangles among triples of vertices that have at least two edges; and the local clustering coefficient of a single vertex is the ratio of connected neighbors among all pairs of neighbors (one then usually considers these local clustering coefficients averaged over all vertices).

We note that some types of real-world networks are homogeneous in the sense that most vertices have roughly the same degree (e.g., the degrees of most vertices in a typical road network lie between 1 and 4 [52]). However, large networks from many domains (social networks, electricity maps, biological networks, co-author graphs, romantic relationships, etc. [41, 29, 46, 47]) are highly heterogeneous. Intuitively, this reflects the fact that the entities represented by nodes typically differ in importance, influence, or popularity.

The interdependency between edges is also not surprising. For example, two autonomous systems in the Internet that both have a direct connection to a third autonomous system are likely to be geographically close, which increases the chance that they also have a direct connection. In a similar fashion, two researchers who collaborated with the same third researcher are probably working on similar topics and are thus more likely to collaborate than two random researchers. One can therefore expect the clustering coefficients of real-world networks to be bounded away from 0 (e.g., the local clustering coefficients of collaboration networks are mostly above 0.5 [52]). In the following we briefly discuss different random graph models with heterogeneity and interdependency in mind.

Random graphs

The earliest and most-studied model is the Erdős-Rényi random graph [31]. In this model, an input graph $G(n, p)$ with n vertices is generated by connecting each vertex pair independently with probability p. Erdős-Rényi graphs are popular among researchers mainly for two reasons. First, for $p = \frac{1}{2}$, it produces each (labeled) graph with n vertices with the same probability, which seems like a desirable property. Second, it is simple, which makes it accessible to rigorous and very detailed mathematical analysis. It is thus not surprising, that, on the one hand, Erdős-Rényi graphs were used in the early days of average-case analysis, and are, on the other hand, still the object of current research. To name two examples, Angluin and Valiant [8] showed in 1977 that on Erdős-Rényi random graphs the NP-hard problem Hamiltonian Circuit can be solved in expected time $O(n \log^2 n)$ if the probability p

is sufficiently large. Also the the W[1]-hard k-Clique problem admits an average-case FPT-algorithm on Erdős-Rényi graphs, bringing together the fields of average-case and parameterized complexity [33].

Unfortunately, graphs generated with the Erdős-Rényi model lack the above-mentioned desired properties. *Asymptotically almost surely* (i.e, with probability $\to 1$ for $n \to \infty$), all vertices have roughly the same degree, leading to almost regular graphs, and their clustering coefficients tend to 0 for $n \to \infty$.

Heterogeneous random graphs

To account for the heterogeneity of real-world networks, different models have been introduced. The Barabási-Albert model [9] (also called *preferential attachment*) adds one vertex at a time, connecting it to already existing vertices with probability proportional to their degree. This model actually has an explanatory character in the sense that a reasonable assumption (namely that already popular nodes are more attractive to new nodes) leads to the power-law degree distribution observed in practice. On the downside, this procedural description of the model introduces strong stochastic dependencies, which makes a mathematical analysis of the resulting graphs rather difficult [21]. The Chung-Lu model [27, 28] produces scale-free graphs by assigning weights to the nodes (following a power law) and connecting every pair of vertices with a probability proportional to the product of their weights [3, 4]. Though the Chung-Lu model cannot explain the emergence of a power-law degree distribution in real-world networks, it is much more accessible to a mathematical analysis due to the fact that edges are chosen independently. Further similar random models are inhomogeneous random graphs by van der Hofstad [40] and Norros-Reittu graphs [48].

However, despite this abundance of theoretical models for power-law networks, all of them fall short in describing real-world networks as their clustering coefficient tends to 0 for $n \to \infty$ while it is bounded away from 0 for most real-world networks.

Random graphs with interdependencies

There are a number of random graph models that lead to graphs with non-vanishing clustering coefficients. An example of such models are *geometric random graphs* [38] (also called *random unit disk graphs*). Such a graph is obtained by assigning random coordinates to each vertex and connecting two vertices if and only if they are close (with respect to Euclidean distance). It is not surprising that the geometric locality (two vertices close to a third vertex are also close to each other) leads to high clustering coefficients in the resulting graphs [49]. Though random geometric graphs are well suited to represent sensor networks [51], they are less suited for many other real-world networks. As in the Erdős-Rényi model, the resulting degree distribution is rather homogeneous [49]. Moreover, two other properties often observed in large real-world networks, namely sparsity and a small diameter, cannot be achieved together by random unit disk graphs for the following reason: If the generated graph is sparse, the vertices have to occupy an area linear in the number of vertices. This inevitably leads to a polynomial diameter (e.g., $\sqrt{n}$ in the 2-dimensional Euclidean plane).

Watts and Strogatz [56] proposed a model leading to sparse graphs with high clustering and logarithmic diameter and coined the term *small-world network* for this type of network. Their model starts with a regular graph with high clustering and randomly rewires edges (i.e., it deletes and adds edges randomly). The resulting graphs inherit the high clustering from the initial graph while obtaining a small diameter due to the edges added independently at random (as in the Erdős-Rényi model). On the downside, this model also leads to a homogeneous degree distribution.

2 Combining heterogeneity and interdependency

None of the aforementioned models fulfill both properties (heterogeneity and interdependency) at the same time. A natural model that leads to graphs with both features are *hyperbolic random graphs* as introduced by Krioukov et al. [44]. We strongly believe that hyperbolic random graphs are an excellent model to describe and study real-world networks. This believe is supported by an empirical analysis of a few hundred real-world networks [14]. In the following we briefly describe the model, discuss previous results establishing additional desirable properties and thereby debate why we believe that hyperbolic random graphs are well suited for representing large real-world networks.

Definition and basic properties

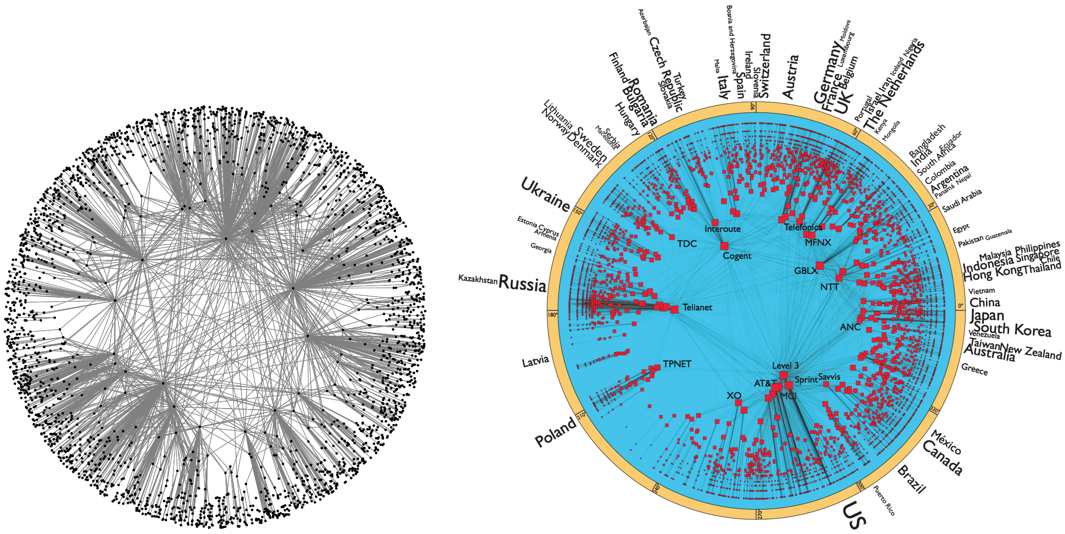
Hyperbolic random graphs are generated in the same way as geometric random graphs, replacing the Euclidean with the hyperbolic plane, i.e., the vertices are assigned to random positions within a disk of the hyperbolic plane and two vertices are connected if and only if their hyperbolic distance is small. As in the Euclidean case, the geometry leads to a non-vanishing clustering coefficient [44, 39, 25]. However, the hyperbolic plane expands exponentially, i.e., the area and the circumference of a disk grows exponentially in its radius. Thus, when distributing the vertices evenly within a disk in the hyperbolic plane, most vertices will end up close to the boundary of the disk; see Figure 1a. As distances are much larger between vertices close to the boundary, these vertices have low degree, while the few vertices close to the center have high degree. In fact, this way of distributing the vertices leads to a power-law degree distribution with power-law exponent $\beta = 3$ [44, 39]. Moreover, one can obtain arbitrary exponents β, with $\beta > 2$, by assuming a hyperbolic space with different negative curvature when sampling the radial coordinates of the vertices [18]. Thus, the power-law exponent is a parameter of the model. Similarly, the expected average degree of the vertices can be controlled by changing the threshold distance, below which vertices are still connected.

Beyond these two fundamental properties of a heterogeneous degree distribution and interdependency between edges, hyperbolic random graphs show other desirable properties. In contrast to the Euclidean plane, the exponential expansion of hyperbolic space makes it possible to have a graph with low average degree and only logarithmic diameter (intuitively speaking, spreading the vertices over a region with linear area no longer implies that this region has polynomial diameter as in the Euclidean plane). In fact, hyperbolic random graphs have polylogarithmic diameter [35, 36, 43] and the average distance between pairs of vertices is $\Theta(\log \log n)$ [1].

Such a more realistic random graph model now opens up the possibility to explain why real-world instances tend to be algorithmically well-behaved, by performing an average-case analysis. There is not much work in this direction. One particular example is bidirectional breadth-first search. Borassi and Natale [22] observed that bidirectional search performs sublinear in practice. This was the motivation for Bläsius et al. [12], who proved that the runtime of bidirectional search is sublinear runtime with high probability for hyperbolic random graphs.

Component structure and connectivity

Note that there is no explicit mechanism ensuring that a hyperbolic random graph is connected, and in fact, it usually consists of multiple connected components. As some applications are only interested in connected graphs, it is good to know that hyperbolic



(a) Due to the exponential expansion of the hyperbolic plane, most vertices lie close to the disk's boundary. Nodes close to the center are rare, but highly connected, leading to the power-law degree distribution.

(b) The embedding of the Internet by Boguná et al. [20] into the hyperbolic plane enables greedy routing with high success ratio and low stretch, i.e., for 97% of the vertex pairs, greedy routing succeeds and resulting paths are on average only 10% longer than the shortest paths.

Figure 1 A hyperbolic random graph (a) and an embedding of the Internet into the hyperbolic plane (b).

random graphs (at least for $2 < \beta < 3$) have a so-called *giant component* [17], i.e., a connected component with a linear number of vertices, while all other components have only polylogarithmic size [43].

Related to the component structure is the question of how densely connected each component is. Despite the fact that we assume real-world networks to have constant average degree, we still expect to find large highly connected subgraphs forming communities. For hyperbolic random graphs with power-law exponent $2 < \beta < 3$, it has been shown that the largest clique has polynomial size $\Theta(\sqrt{n^{3-\beta}})$ and that there are $\sqrt{n^{(3-\beta)k}} \cdot \Theta(k)^{-k}$ cliques of size k [34, 15]. On the other hand, different parts of the graph are loosely connected in the sense that hyperbolic random graphs have small balanced separators. To be more precise, if $2 < \beta < 3$, there exists a hierarchy of balanced separators each having size $\Theta(\sqrt{n^{3-\beta}})$ [10], leading to sublinear treewidth of $\Theta(\sqrt{n^{3-\beta}})$ (which is tight due to the matching bound for the clique size). On the one hand, this result enables faster algorithms by using dynamic programming on the tree decomposition. On the other hand, it gives structural insights distinguishing hyperbolic random graphs further from the Barabási-Albert model having linear treewidth [37]. We note that real-world networks typically have rather small treewidth [2, Tables I and V], which gives another indication that hyperbolic random graphs are well suited for representing large real-world networks.

Generating algorithms and related models

Algorithmically, hyperbolic random graphs can naively be generated in $\Theta(n^2)$ time [7], which has been improved to $O(n^{1.5} \log n)$ [55] and even down to linear [24, 54, 50, 23] by using geometric data structures. The algorithm of Bringmann et al. [24, 23] additionally makes use of the relationship between hyperbolic random graphs and so-called geometric inhomogeneous

random graphs (GIRGs), which can be seen as a combination of the Chung-Lu model with a geometry. In a similar way, hyperbolic random graphs have been related to Barabási-Albert graphs additionally equipped with a geometry [32].

The hyperbolic metric of real-world networks

Beyond the very promising hyperbolic random graph model (showing properties that one expects in large real-world networks of many domains), the metric defined by most networks appears to be very similar to the metric of the hyperbolic plane. To support this observation, we name three examples. Boguná [20] embedded the Internet network into the hyperbolic plane by assigning a hyperbolic coordinate to every autonomous system; see Figure 1b. They observed that greedy routing solely based on these coordinates is almost maximally efficient, i.e., it finds short paths between almost any two pairs of vertices. Verbeek and Suri [53] show that graphs with low quasi-cyclicity (which appears to be low for many networks) admit a metric embedding into a hyperbolic space of constant dimension with constant additive distortion. Finally, Albert et al. [6] computed the so-called Gromov hyperbolicity (which basically measures how close a graph metric is to the metric of the hyperbolic plane) for several biological and social networks. They found out that these networks are indeed hyperbolic in this sense, i.e., their Gromov hyperbolicity is small. Though all three results relate networks to hyperbolic geometry in different ways, they all support the above claim that the metric of networks is similar to the hyperbolic metric.

Embeddings in the hyperbolic plane

A common application of a random model describing real-world instances reasonably well is the possibility to perform a meaningful average-case analysis. However, acknowledging hyperbolic random graphs as a reasonable model for real-world networks opens up another line of research: viewing a given real-world network as a hyperbolic random graph but without known coordinates. It is then a natural question, whether we can retrieve the missing geometric information, i.e., whether we can embed the graph into the hyperbolic plane such that most edges are short and most non-adjacent vertices are far apart. There are a number of algorithms for embedding a network into the hyperbolic space. Algorithms with quasilinear runtime are known for maximum likelihood embeddings [11, 16] and for optimizing greedy routing [13].

References

1 Mohammed Amin Abdullah, Michel Bode, and Nikolaos Fountoulakis. Typical Distances in a Geometric Model for Complex Networks. *CoRR*, abs/1506.07811, 2015.

2 Aaron B. Adcock, Blair D. Sullivan, and Michael W. Mahoney. Tree Decompositions and Social Graphs. *Internet Mathematics*, 12(5):315–361, 2016. doi:10.1080/15427951.2016.1182952.

3 William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *32nd STOC*, pages 171–180, 2000. doi:10.1145/335305.335326.

4 William Aiello, Fan Chung, and Linyuan Lu. A Random Graph Model for Power Law Graphs. *Experimental Mathematics*, 10(1):53–66, 2001. doi:10.1080/10586458.2001.10504428.

5 Takuya Akiba and Yoichi Iwata. Branch-and-Reduce Exponential/FPT Algorithms in Practice: A Case Study of Vertex Cover. *Theoretical Computer Science*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

6 Réka Albert, Bhaskar DasGupta, and Nasim Mobasheri. Topological Implications of Negative Curvature for Biological and Social Networks. *Physical Review E*, 89:032811, 2014. doi:10.1103/PhysRevE.89.032811.

7 Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic graph generator. *Computer Physics Communications*, 196:492–496, 2015. doi:10.1016/j.cpc.2015.05.028.

8 D. Angluin and L.G. Valiant. Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings. *Journal of Computer and System Sciences*, 18(2):155–193, 1979. doi:10.1016/0022-0000(79)90045-X.

9 Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.

10 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Hyperbolic Random Graphs: Separators and Treewidth. In *24th ESA*, pages 15:1–15:16, 2016. doi:10.4230/LIPIcs.ESA.2016.15.

11 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient Embedding of Scale-Free Graphs in the Hyperbolic Plane. In *24th ESA*, pages 16:1–16:18, 2016. doi:10.4230/LIPIcs.ESA.2016.16.

12 Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. Efficient Shortest Paths in Scale-Free Networks with Underlying Hyperbolic Geometry. In *45th ICALP*, pages 20:1–20:14, 2018.

13 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, and Anton Krohmer. Hyperbolic Embeddings for Near-Optimal Greedy Routing. In *20th ALENEX*, pages 199–208, 2018.

14 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Anton Krohmer, and Jonathan Striebel. Towards a Systematic Evaluation of Generative Network Models. In *15th WAW*, pages 99–114, 2018.

15 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Cliques in Hyperbolic Random Graphs. *Algorithmica*, 80:2324–2344, 2018.

16 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient Embedding of Scale-Free Graphs in the Hyperbolic Plane. *IEEE/ACM Transactions on Networking*, 26(1-57570FN):920–933, 2018.

17 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. On the Largest Component of a Hyperbolic Model of Complex Networks. *Electronic Journal of Combinatorics*, 22(3):P3.24, 2015.

18 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. The probability of connectivity in a hyperbolic model of complex networks. *Random Structures & Algorithms*, 49(1):65–94, 2016. doi:10.1002/rsa.20626.

19 Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1):1–106, 2006. doi:10.1561/0400000004.

20 Marián Boguná, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the Internet with Hyperbolic Mapping. *Nature Communications*, 1:62, 2010. doi:10.1038/ncomms1063.

21 Béla Bollobás and Oliver Riordan. The Diameter of a Scale-Free Random Graph. *Combinatorica*, 24:5–34, 2004. doi:10.1007/s00493-004-0002-2.

22 Michele Borassi and Emanuele Natale. KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation. In *24th ESA*, volume 57, pages 20:1–20:18, 2016.

23 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Sampling Geometric Inhomogeneous Random Graphs in Linear Time. In *25th ESA*, pages 20:1–20:15, 2017.

24 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *Theoretical Computer Science*, 2019.

25 Elisabetta Candellero and Nikolaos Fountoulakis. Clustering and the Hyperbolic Geometry of Complex Networks. *Internet Mathematics*, 12(1-2):2–53, 2016. doi:10.1080/15427951.2015.1067848.

26 Markus Chimani, Maria Kandyba, Ivana Ljubić, and Petra Mutzel. Obtaining Optimal k-Cardinality Trees Fast. In *20th ALENEX*, pages 27–36, 2008. doi:10.1137/1.9781611972887.3.

27 Fan Chung and Linyuan Lu. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics*, 6(2):125–145, 2002. doi:10.1007/PL00012580.

28 Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002.

29 Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661–703, 2009. doi:10.1137/070710111.

30 Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck. Exact Combinatorial Branch-and-Bound for Graph Bisection. In *14th ALENEX*, pages 30–44, 2012. doi:10.1137/1.9781611972924.3.

31 P. Erdős and A. Rényi. On Random Graphs I. *Publicationes Mathematicae*, 6:290–297, 1959.

32 L. Ferretti, M. Cortelezzi, and M. Mamino. Duality between preferential attachment and static networks on hyperbolic spaces. *Europhysics Letters*, 105(3):38001, 2014.

33 Nikolaos Fountoulakis, Tobias Friedrich, and Danny Hermelin. On the Average-Case Complexity of Parameterized Clique. *Theoretical Computer Science*, 576:18–29, 2015. doi:10.1016/j.tcs.2015.01.042.

34 Tobias Friedrich and Anton Krohmer. Cliques in Hyperbolic Random Graphs. In *34th INFOCOM*, pages 1544–1552, 2015.

35 Tobias Friedrich and Anton Krohmer. On the Diameter of Hyperbolic Random Graphs. In *42nd ICALP*, pages 241–252, 2015.

36 Tobias Friedrich and Anton Krohmer. On the Diameter of Hyperbolic Random Graphs. *SIAM Journal on Discrete Mathematics*, 32(2):1314–1334, 2018.

37 Yong Gao. Treewidth of Erdős-Rényi Random Graphs, Random Intersection Graphs, and Scale-Free Random Graphs. *Discrete Applied Mathematics*, 160(4–5):566–578, 2012. doi:10.1016/j.dam.2011.10.013.

38 E. Gilbert. Random Plane Networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961. doi:10.1137/0109045.

39 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: degree sequence and clustering. In *39th ICALP*, pages 573–585, 2012.

40 Remco van der Hofstad. *Random Graphs and Complex Networks:*, volume 1 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2016. doi:10.1017/9781316779422.

41 Matthew O. Jackson. *Social and Economic Networks*. Princeton University Press, 2010.

42 Richard M. Karp. The Probabilistic Analysis of Combinatorial Optimization Algorithms. In *Proceedings of the International Congress of Mathematicians*, pages 1601–1609, 1983.

43 Marcos Kiwi and Dieter Mitsche. A Bound for the Diameter of Random Hyperbolic Graphs. In *12th ANALCO*, pages 26–39, 2015. doi:10.1137/1.9781611973761.3.

44 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010. doi:10.1103/PhysRevE.82.036106.

45 Leonid A. Levin. Average Case Complete Problems. *SIAM Journal on Computing*, 15(1):285–286, 1986. doi:10.1137/0215020.

46 Michael Mitzenmacher. A Brief History of Generative Models for Power Law and Lognormal Distributions. *Internet Mathematics*, 1(2):226–251, 2003. doi:10.1080/15427951.2004.10129088.

47 M. E. J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005. doi:10.1080/00107510500052444.

48 Ilkka Norros and Hannu Reittu. On a conditionally Poissonian graph process. *Advances in Applied Probability*, 38(1):59–75, 2006.

49 M. Penrose. *Random Geometric Graphs*. Oxford scholarship online. Oxford University Press, 2003.

50 Manuel Penschuck. Generating Practical Random Hyperbolic Graphs in Near-Linear Time and with Sub-Linear Memory. In *16th SEA*, pages 26:1–26:21, 2017.

51 G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000. doi:10.1145/332833.332838.

52 Ryan Rossi and Nesreen Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *29th AAAI*, pages 4292–4293, 2015. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9553`.

53 Kevin Verbeek and Subhash Suri. Metric Embedding, Hyperbolic Space, and Social Networks. In *30th SoCG*, pages 501–510, 2014. `doi:10.1145/2582112.2582139`.

54 M. von Looz, M. S. Özdayi, S. Laue, and H. Meyerhenke. Generating Massive Complex Networks with Hyperbolic Geometry Faster in Practice. In *20th IEEE HPEC*, pages 1–6, 2016. `doi:10.1109/HPEC.2016.7761644`.

55 Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating Random Hyperbolic Graphs in Subquadratic Time. In *26th ISAAC*, pages 467–478, 2015.

56 Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 939:440–442, 1998. `doi:10.1038/30918`.

The Semialgebraic Orbit Problem

Shaull Almagor
Department of Computer Science, Oxford University, UK
shaull.almagor@cs.ox.ac.uk

Joël Ouaknine
Max Planck Institute for Software Systems, Germany
Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

James Worrell
Department of Computer Science, Oxford University, UK
jbw@cs.ox.ac.uk

Abstract

The *Semialgebraic Orbit Problem* is a fundamental reachability question that arises in the analysis of discrete-time linear dynamical systems such as automata, Markov chains, recurrence sequences, and linear while loops. An instance of the problem comprises a dimension $d \in \mathbb{N}$, a square matrix $A \in \mathbb{Q}^{d\times d}$, and semialgebraic source and target sets $S, T \subseteq \mathbb{R}^d$. The question is whether there exists $x \in S$ and $n \in \mathbb{N}$ such that $A^n x \in T$.

The main result of this paper is that the Semialgebraic Orbit Problem is decidable for dimension $d \leq 3$. Our decision procedure relies on separation bounds for algebraic numbers as well as a classical result of transcendental number theory – Baker's theorem on linear forms in logarithms of algebraic numbers. We moreover argue that our main result represents a natural limit to what can be decided (with respect to reachability) about the orbit of a single matrix. On the one hand, semialgebraic sets are arguably the largest general class of subsets of $\mathbb{R}^d$ for which membership is decidable. On the other hand, previous work has shown that in dimension $d = 4$, giving a decision procedure for the special case of the Orbit Problem with singleton source set S and polytope target set T would entail major breakthroughs in Diophantine approximation.

2012 ACM Subject Classification Computing methodologies → Algebraic algorithms; Theory of computation → Logic and verification

Keywords and phrases linear dynamical systems, Orbit Problem, first order theory of the reals

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.6

Related Version A full version of the paper is available at `https://arxiv.org/abs/1901.11023`.

Funding *Joël Ouaknine*: Supported by ERC grant AVS-ISS (648701), and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Projektnummer 389792660 – TRR 248.
James Worrell: Supported by EPSRC Fellowship EP/N008197/1.

1 Introduction

This paper concerns decision problems of the following form: given $d \in \mathbb{N}$, a square matrix $A \in \mathbb{Q}^{d\times d}$, and respective *source* and *target* sets $S, T \subseteq \mathbb{R}^d$, does there exist $n \in \mathbb{N}$ and $x \in S$ such that $A^n x \in T$? One way to categorise such problems is according to the types of sets allowed for the source and target (e.g., polytopes or semialgebraic sets). We collectively refer to the various problems that arise in this way as *Orbit Problems*. Orbit Problems occur naturally in the reachability analysis of discrete-time linear dynamical systems, including Markov chains, automata, recurrence sequences, and linear loops in program analysis (see [5, 11, 9] and references therein).

In order to describe the main result of this paper in relation to existing work, we identify three successively more general types of Orbit Problems. In the *point-to-point* version both the source and target are singletons with rational coordinates; in the *Polytopic Orbit Problem* the source and target S and T are polytopes (i.e., sets defined by conjunctions of linear inequalities with rational coefficients); in the *Semialgebraic Orbit Problem* S and T are semialgebraic sets defined with rational parameters.

The question of the decidability of the point-to-point Orbit Problem was raised by Harrison in 1969 [10]. The problem remained open for ten years until it was finally resolved in a seminal paper of Kannan and Lipton [11], who in fact gave a polynomial-time decision procedure.

The Polytopic Orbit Problem is considerably more challenging than the point-to-point version, and its decidablity seems out of reach for now. Indeed the special case in which S is a singleton and T is a linear subspace of $\mathbb{R}^d$ of dimension $d-1$ is a well-known decision problem in its own right, called the *Skolem Problem*, whose decidability has been open for many decades [20]. In contrast to the point-to-point case the only positive decidability results for the Polytopic Orbit Problem are in the case of fixed dimension d. For the Skolem Problem, decidability is known for $d \leq 4$ [14, 22]. In case S and T are allowed to be arbitrary polytopes, decidability is known in case $d \leq 3$ [1] (see also [4]). While Kannan and Lipton's decision procedure in the point-to-point case mainly relied on algebraic number theory (e.g., separation bounds between algebraic numbers and prime factorisation of ideals in rings of algebraic integers), the decision procedures for the Skolem Problem and the Polytopic Orbit Problem additionally use results about transcendental numbers (specifically Baker's theorem about linear forms in logarithms of algebraic numbers). It was shown in [4] that the existence of a decision procedure for the Polytopic Orbit Problem in dimension $d = 4$ would entail computability of the Diophantine approximation types of a general class of transcendental numbers (a problem considered intractable at present). Not only does this suggest that the use of transcendental number theory is unavoidable in analysing the Polytopic Orbit Problem, it also indicates that further progress beyond the case $d = 3$ is contingent upon significant advances in the field of Diophantine approximation.

In this paper we remain in dimension $d = 3$ and consider a generalisation of previous work by allowing the source and target sets to be semialgebraic, that is, defined by Boolean combinations of polynomial equalities and inequalities. This allows us to handle three-dimensional source and target sets in much greater geometrical generality than polytopes. In applications to program analysis and dynamical systems, semialgebraic sets are indispensable in formulating sufficiently expressive models (e.g., to describe initial conditions and transition guards) and in model analysis (e.g., in synthesising invariants and barrier certificates and approximating sets of reachable states) [15, 12].

The Semialgebraic Orbit Problem could be reduced to the polytopic case in a fairly straightforward fashion by increasing the dimension d according to the degree of the polynomials appearing in the semialgebraic constraints. However such a general approach is doomed to failure in view of the obstacles to obtaining decidability in the polytopic case beyond dimension 3 and instead we develop specific techniques for the semialgebraic case that are considerably more challenging than in the Polytopic Problem. As in previous work on the Skolem Problem and on the Polytopic Orbit Problem, Baker's Theorem plays a crucial role in the present development. The main difficulty in generalising from the polytopic case to the semialgebraic case lies in the delicate analytic arguments that are required to bring Baker's Theorem to bear. More precisely: (i) we need to resort to symbolic quantifier elimination (in lieu of explicit Fourier-Motzkin elimination, which had been used in the Polytopic Orbit

Problem), since we are now dealing with non-linear constraints; (ii) we also need to perform spectral calculations symbolically, via the use of Vandermonde methods, instead of the explicit direct approach possible in our earlier work; and (iii) we replace triangulation of polytopes by cylindrical algebraic decomposition of semialgebraic sets into cells, which again necessitates a new symbolic treatment along with a substantially refined analysis based on Taylor approximation of the attendant functions.

In summary, this paper provides a decision procedure for the Orbit Problem in dimension $d = 3$ with semialgebraic source and target sets. The latter appear to be a natural limit to the positive decidability results that can be obtained for this problem, barring major new advances in Diophantine approximation.

At a technical level, our contributions are twofold: in Section 3 we start by analysing the case of the Orbit Problem in which S is a singleton and T a semialgebraic set. We then reduce this problem in Section 3.1 to solving certain systems of polynomial-exponential equalities and inequalities, and in Section 3.2 we show precisely how to solve such systems. The second technical contribution consists in handling the general case of the Semialgebraic Orbit Problem, in Section 4. There, we show how to circumvent problems that arise when quantifying over the set S, and arrive at a system that can ultimately be solved using the techniques and results developed in Section 3.2.

For brevity, some proofs are omitted, and can be found in the full version on the authors' homepages.

2 Mathematical Tools

In this section we introduce the key technical tools used in this paper.

2.1 Algebraic numbers

For $p \in \mathbb{Z}[x]$ a polynomial with integer coefficients we denote by $\|p\|$ the bit length of its representation as a list of coefficients encoded in binary. Note that the *degree* of p, denoted $\deg(p)$ is at most $\|p\|$, and the *height* of p – i.e., the maximum of the absolute values of its coefficients, denoted $H(p)$ – is at most $2^{\|p\|}$.

We begin by summarising some basic facts about algebraic numbers and their (efficient) manipulation. The main references include [3, 6, 19]. A complex number α is *algebraic* if it is a root of a single-variable polynomial with integer coefficients. The *defining polynomial* of α, denoted p_α, is the unique polynomial of least degree, and whose coefficients do not have common factors, which vanishes at α. The *degree* and *height* of α are respectively those of p, and are denoted $\deg(\alpha)$ and $H(\alpha)$. We denote the field of algebraic numbers by $\mathbb{A}$. A standard representation[1] for algebraic numbers is to encode α as a tuple comprising its defining polynomial together with rational approximations of its real and imaginary parts of sufficient precision to distinguish α from the other roots of p_α. More precisely, α can be represented by $(p_\alpha, a, b, r) \in \mathbb{Z}[x] \times \mathbb{Q}^3$ provided that α is the unique root of p_α inside the circle in $\mathbb{C}$ of radius r centred at $a + bi$. A separation bound due to Mignotte [13] asserts that for roots $\alpha \neq \beta$ of a polynomial $p \in \mathbb{Z}[x]$, we have

$$|\alpha - \beta| > \frac{\sqrt{6}}{d^{(d+1)/2} H^{d-1}} \tag{1}$$

[1] Note that this representation is not unique.

where $d = \deg(p)$ and $H = H(p)$. Thus if r is required to be less than a quarter of the root-separation bound, the representation is well-defined and allows for equality checking. Given a polynomial $p \in \mathbb{Z}[x]$, it is well-known how to compute standard representations of each of its roots in time polynomial in $\|p\|$ [3, 6, 17]. Thus given an algebraic number α for which we have (or wish to compute) a standard representation, we write $\|\alpha\|$ to denote the bit length of this representation. From now on, when referring to computations on algebraic numbers, we always implicitly refer to their standard representations.

Note that Equation (1) can be used more generally to separate arbitrary algebraic numbers: indeed, two algebraic numbers α and β are always roots of the polynomial $p_\alpha p_\beta$ of degree at most $\deg(\alpha) + \deg(\beta)$, and of height at most $H(\alpha)H(\beta)$. Given algebraic numbers α and β, one can compute $\alpha + \beta$, $\alpha\beta$, $1/\alpha$ (for $\alpha \neq 0$), $\overline{\alpha}$, and $|\alpha|$, all of which are algebraic, in time polynomial in $\|\alpha\| + \|\beta\|$. Likewise, it is straightforward to check whether $\alpha = \beta$. Moreover, if $\alpha \in \mathbb{R}$, deciding whether $\alpha > 0$ can be done in time polynomial in $\|\alpha\|$. Efficient algorithms for all these tasks can be found in [3, 6].

2.2 First-order theory of the reals

Let $\vec{x} = x_1, \ldots, x_m$ be a list of m real-valued variables, and let $\sigma(\vec{x})$ be a Boolean combination of atomic predicates of the form $g(\vec{x}) \sim 0$, where each $g(\vec{x}) \in \mathbb{Z}[x]$ is a polynomial with integer coefficients over these variables, and $\sim \in \{>, =\}$. A *formula of the first-order theory of the reals* is of the form $Q_1 x_1 Q_2 x_2 \cdots Q_m x_m \sigma(\vec{x})$, where each Q_i is one of the quantifiers $\exists$ or $\forall$. Let us denote the above formula by τ , and write $\|\tau\|$ to denote the bit length of its syntactic representation. Tarski famously showed that the first-order theory of the reals is decidable [21]. His procedure, however, has non-elementary complexity. Many substantial improvements followed over the years, starting with Collins's technique of cylindrical algebraic decomposition [7], and culminating with the fine-grained analysis of Renegar [19]. In this paper, we will use the following theorems [18, 19].

▶ **Theorem 1** (Renegar [18]). *The problem of deciding whether a closed formula τ of the form above holds over the reals is in **2EXP**, and in **PSPACE** if τ has only existential quantifiers.*

▶ **Theorem 2** (Renegar [19]). *There is an algorithm that, given a sentence $\tau(x_1, \ldots, x_m)$ where $x_1, \ldots, x_m$ are free variables, computes an equivalent quantifier-free formula in* disjunctive normal form *(DNF) $\Phi(x_1, \ldots, x_m) = \bigvee_I \bigwedge_J R_{I,J}(x_1, \ldots, x_m) \sim_{I,J} 0$ where $R_{I,J}$ is a polynomial* [2] *and $\sim_{I,J} \in \{>, =\}$. Moreover, the algorithm runs in time $2^{2^{\mathcal{O}(\|\tau\|)}}$, and in particular, $\|\Phi\| = 2^{2^{\mathcal{O}(\|\tau\|)}}$.*

A set $S \subseteq \mathbb{R}^d$ is *semialgebraic* if there exists a formula $\Phi(x_1, \ldots, x_d)$ in the first-order theory of the reals with free variables $x_1, \ldots, x_d$ such that $S = \{(c_1, \ldots, c_d) : \Phi(c_1, \ldots, c_d)$ is true$\}$.

We remark that algebraic constants can also be incorporated as coefficients in the first-order theory of the reals (and in particular, in the definition of semialgebraic sets), as follows. Consider a polynomial $g(x_1, \ldots, x_m)$ with algebraic coefficients $c_1, \ldots, c_k$. We replace every c_j with a new, existentially-quantified variable y_j, and add to the sentence the predicates $p_{c_j}(y_j) = 0$ and $(y_j - (a + bi))^2 < r^2$, where (p_{c_j}, a, b, r) is the representation of c_j. Then, in any evaluation of this formula to True, it must hold that y_j is assigned value c_j.

[2] Technically, the indices should be I, J_I, but we omit the dependency of J on I for simplicity.

3 Almost Self-Conjugate Systems of Inequalities

In this section we lay the groundwork for solving the Semialgebraic Orbit Problem. We do so by initially treating the case where the set S of initial points is a singleton.

3.1 Analysis of the Point-to-Semialgebraic Orbit Problem

The *point-to-semialgebraic Orbit Problem* is to decide, given a matrix $A \in \mathbb{Q}^{3\times 3}$, an initial point $s \in \mathbb{Q}^3$ and a semialgebraic target $T \subseteq \mathbb{R}^3$, whether there exists $n \in \mathbb{N}$ such that $A^n s \in T$.

By Theorem 2, we can compute a quantifier-free representation of T. That is, we can write $T = \{(x,y,z) : \bigvee_I \bigwedge_J R_{I,J}(x,y,z) \sim_{I,J} 0\}$ where $R_{I,J}$ are polynomials with integer coefficients, and $\sim_{I,J} \in \{>,=\}$. For the purpose of solving the point-to-semialgebraic Orbit Problem, we note that it is enough to consider each disjunct separately. Thus, we can assume $T = \{(x,y,z) : \bigwedge_J R_J(x,y,z) \sim_J 0\}$, and it remains to decide whether there exists $n \in \mathbb{N}$ such that $\bigwedge_J R_J(A^n s) \sim_J 0$.

Note that, as per Theorem 2, we have that $\|R_J\| = 2^{2^{\mathcal{O}(\|T\|)}}$. Moreover, the number of terms in the DNF formula above can itself be doubly-exponential in $\|T\|$. Complexity wise, this is the most expensive part of our algorithm.

Consider the eigenvalues of A. Since A is a 3×3 matrix, then either it has only real eigenvalues, or it has one real eigenvalue and two conjugate complex eigenvalues. In particular, if A has complex eigenvalues, then it is diagonalisable.

The case where A has only real eigenvalues is treated in the full version for the general case of the Semialgebraic Orbit Problem, and is considerably simpler.

Henceforth, we assume A has complex eigenvalues, so that $A = PDP^{-1}$ with $D = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \overline{\lambda} & 0 \\ 0 & 0 & \rho \end{pmatrix}$, where λ is a complex eigenvalue, $\rho \in \mathbb{R}$, and P an invertible matrix.

Observe that $A^n = PD^nP^{-1}$. By carefully analysing the structure of P, it is not hard to show that $A^n s = \begin{pmatrix} a_1\lambda^n + \overline{a_1}\overline{\lambda}^n + b_1\rho^n \\ a_2\lambda^n + \overline{a_2}\overline{\lambda}^n + b_2\rho^n \\ a_3\lambda^n + \overline{a_3}\overline{\lambda}^n + b_3\rho^n \end{pmatrix}$ where the a_i and b_i are algebraic and the b_i are also real (see the full version for a detailed analysis).

Thus, we want to decide whether there exists $n \in \mathbb{N}$ such that $R_J(a_1\lambda^n + \overline{a_1}\overline{\lambda}^n + b_1\rho^n, a_2\lambda^n + \overline{a_2}\overline{\lambda}^n + b_2\rho^n, a_3\lambda^n + \overline{a_3}\overline{\lambda}^n + b_3\rho^n) \sim_J 0$ for every J. Since R_J is a polynomial, then by aggregating coefficients we can write

$$R_J(A^n s) = \sum_{0\le p_1,p_2,p_3\le k} \alpha_{p_1,p_2,p_3}\lambda^{np_1}\overline{\lambda}^{np_2}\rho^{np_3} + \overline{\alpha_{p_1,p_2,p_3}}\,\overline{\lambda}^{np_1}\lambda^{np_2}\rho^{np_3} \tag{2}$$

for some $k \in \mathbb{N}$. Note that we treat the (real) coefficients of ρ as a sum of complex conjugate coefficients, but this can easily be achieved by writing e.g., $c\rho^{np} = \frac{c}{2}\rho^{np} + \frac{c}{2}\rho^{np}$.

We notice that for every J, the polynomial $R_J(A^n s)$, consists of conjugate summands. More precisely, $R_J(A^n s)$, when viewed as a polynomial in $\lambda^n, \overline{\lambda}^n$, and ρ^n, has the following property.

▶ **Property 3** (Almost Self-Conjugate Polynomial). *A complex polynomial $Q(z_1,z_2,z_3)$ over $\mathbb{C}^3$ is* almost self-conjugate *if*

$$Q(z_1,z_2,z_3) = \sum_{0\le t_1,t_2,t_3\le \ell} \delta_{t_1,t_2,t_3} z_1^{t_1} z_2^{t_2} z_3^{t_3} + \overline{\delta_{t_1,t_2,t_3}} z_2^{t_1} z_1^{t_2} z_3^{t_3}.$$

That is, if $z_2 = \overline{z_1}$ and z_3 is a real variable, then the monomials in Q appear in conjugate pairs with conjugate coefficients.

We refer to the conjunction $\bigwedge_J R_J(A^n s) \sim_J 0$ as an *almost self-conjugate system*. It remains to show that we can decide whether there exists $n \in \mathbb{N}$ that solves the system.

3.2 Solving Almost Self-Conjugate Systems

Our starting point is now an almost self-conjugate system as described above. In the following, we will consider a single conjunct $R_J(A^n s) \sim_J 0$.

We start by normalising the expression $R_J(A^n s) \sim_J 0$ in the form of (2), as follows. Let $\Lambda = \max\left\{|\lambda^{p_1}\overline{\lambda}^{p_2}\rho^{p_3}| : \alpha_{p_1,p_2,p_3} \neq \emptyset\right\}$, we divide the expression in (2) by Λ^n, and get that $R_J(A^n s) \sim_J 0$ iff

$$\sum_{m=0}^{k} \beta_m \gamma^{nm} + \overline{\beta_m}\overline{\gamma}^{nm} + r(n) \sim_J 0 \tag{3}$$

where the β_m are algebraic coefficients, $\gamma = \frac{\lambda}{|\lambda|}$ satisfies $|\gamma| = 1$ and $r(n) = \sum_{l=1}^{k'} \chi_l \mu_l^n + \overline{\chi_l \mu_l}^n$ with χ_l being algebraic coefficients, and $|\mu_l| < 1$ for every $1 \leq l \leq k'$. Moreover, every μ_l is a quotient of two elements of the form $\lambda^{p_1}\overline{\lambda}^{p_2}\rho^{p_3}$, and thus, by Section 2.1, $\deg(\mu_l) = \|R_J\|^{\mathcal{O}(1)}$ and $H(\mu_l) = 2^{\|R_J\|^{\mathcal{O}(1)}}$. Note that for simplicity, we reuse the number k, although it may differ from k in (2). We refer to Equation (3) as the *normalised expression*.

In the following, we assume that at least one of the β_j is nonzero for $j \geq 1$. Indeed, otherwise we can recast our analysis on $r(n)$, which is of lower order.

We now split our analysis according to whether or not γ is a root of unity. That is, whether $\gamma^d = 1$ for some $d \in \mathbb{N}$.

3.2.1 The case where γ is a root of unity

Suppose that γ is a root of unity. Then, the set $\{\gamma^n : n \in \mathbb{N}\}$ is a finite set $\left\{\gamma^0, \ldots, \gamma^{d-1}\right\}$. Thus, by splitting the analysis of $A^n s$ according to n mod d, we can reduce the problem to d instances which involve only real numbers. In the full version we detail how to handle this case, and comment on its complexity.

3.2.2 The case where γ is not a root of unity

When γ is not a root of unity, the set $\{\gamma^n : n \in \mathbb{N}\}$ is dense in the unit circle. With this motivation in mind, we define, for a normalised expression, its *dominant function* $f : \mathbb{C} \to \mathbb{R}$ as $f(z) = \sum_{m=0}^{k} \beta_m z^m + \overline{\beta_m}\overline{z}^m$. Observe that (3) is now equivalent to $f(\gamma^n) + r(n) \sim_J 0$.

Our main technical tool in handling (3) is the following lemma.

▶ **Lemma 4.** *Consider a normalised expression as in* (3)*. Let $\|\mathcal{I}\|$ be its encoding length, and let f be its dominant function. Then there exists $N \in \mathbb{N}$ computable in polynomial time in $\|\mathcal{I}\|$ with $N = 2^{\|\mathcal{I}\|^{\mathcal{O}(1)}}$ such that for every $n > N$ it holds that*

1. *$f(\gamma^n) \neq 0$,*
2. *$f(\gamma^n) > 0$ iff $f(\gamma^n) + r(n) > 0$,*
3. *$f(\gamma^n) < 0$ iff $f(\gamma^n) + r(n) < 0$.*

In particular, the lemma implies that if $f(\gamma^n) + r(n) = 0$, then $n \leq N$. That is, if $\sim_J$ is "=", then there is a bound on n that solves the system.

▶ Remark 5. In the formulation of Lemma 4, we measure the complexity with respect to $\|\mathcal{I}\|$. However, recall that when the input is T, we actually have $\|\mathcal{I}\| = 2^{2^{\mathcal{O}(\|T\|)}}$. The analysis in Lemma 4 thus allows us to separate the blowup required for analysing the semialgebraic target from our algorithmic contribution. In particular, when the target has bounded description length, we can obtain better complexity bounds.

We prove Lemma 4 in the remainder of this section.

Since $\{\gamma^n : n \in \mathbb{N}\}$ is dense on the unit circle, our interest in f is also about the unit circle. By identifying $\mathbb{C}$ with $\mathbb{R}^2$, we can think of f as a function of two real variables. In this view, $f(x, y)$ is a polynomial with algebraic coefficients, and we can therefore compute a description of the algebraic set $Z_f = \{(x, y) : f(x, y) = 0 \wedge x^2 + y^2 = 1\}$. We start by showing that this set is finite. Define $g : (-\pi, \pi] \to \mathbb{R}$ by $g(x) = f(e^{ix})$. Explicitly, we have $g(x) = \sum_{m=0}^{k} 2|\beta_m| \cos(mx + \theta_m)$ where $\theta_m = \arg(\beta_m)$. Clearly there is a one-to-one correspondence between Z_f and the roots of g.

We present the following proposition, which will be reused later in the proof.

▶ **Proposition 6.** *For every $x \in (-\pi, \pi]$ there exists $1 \leq i \leq 4k$ such that $g^{(i)}(x) \neq 0$, where $g^{(i)}$ is the i-th derivative of g.*

Proof. Assume by way of contradiction that $g'(x) = \ldots = g^{4k}(x) = 0$. For every $1 \leq i \leq 4k$ we have that

$$g^{(i)}(x) = \begin{cases} \sum_{m=1}^{k} m^i 2|\beta_m| \cos(mx + \theta_m) & i \equiv_4 0 \\ \sum_{m=1}^{k} -m^i 2|\beta_m| \sin(mx + \theta_m) & i \equiv_4 1 \\ \sum_{m=1}^{k} -m^i 2|\beta_m| \cos(mx + \theta_m) & i \equiv_4 2 \\ \sum_{m=1}^{k} m^i 2|\beta_m| \sin(mx + \theta_m) & i \equiv_4 3 \end{cases}$$

(note that the summand that corresponds to $m = 0$ is dropped out in the derivative, as it is constant).

Splitting according $i \mod 4$, we rewrite the equations $g^{(i)}(x) = 0$ in matrix form as follows.[3]

$$\text{for } i \equiv_4 0: \quad \begin{pmatrix} 1^4 & 2^4 & \cdots & k^4 \\ 1^8 & 2^8 & \cdots & k^8 \\ \vdots & \vdots & \vdots & \vdots \\ 1^{4k} & 2^{4k} & \cdots & k^{4k} \end{pmatrix} \begin{pmatrix} 2|\beta_1| \cos(x + \theta_1) \\ 2|\beta_2| \cos(2x + \theta_2) \\ \vdots \\ 2|\beta_k| \cos(kx + \theta_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

[3] By splitting modulo 2, we could actually improve the bound in the proposition from $4k$ to $2k$, but this further complicates the proof.

$$\text{for } i \equiv_4 1: \quad \begin{pmatrix} -1^1 & -2^1 & \cdots & -k^1 \\ -1^5 & -2^5 & \cdots & -k^5 \\ \vdots & \vdots & \vdots & \vdots \\ -1^{4k-3} & -2^{4k-3} & \cdots & -k^{4k-3} \end{pmatrix} \begin{pmatrix} 2|\beta_1|\sin(x+\theta_1) \\ 2|\beta_2|\sin(2x+\theta_2) \\ \vdots \\ 2|\beta_k|\sin(kx+\theta_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\text{for } i \equiv_4 2: \quad \begin{pmatrix} -1^2 & -2^2 & \cdots & -k^2 \\ -1^6 & -2^6 & \cdots & -k^6 \\ \vdots & \vdots & \vdots & \vdots \\ -1^{4k-2} & -2^{4k-2} & \cdots & -k^{4k-2} \end{pmatrix} \begin{pmatrix} 2|\beta_1|\cos(x+\theta_1) \\ 2|\beta_2|\cos(2x+\theta_2) \\ \vdots \\ 2|\beta_k|\cos(kx+\theta_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\text{for } i \equiv_4 3: \quad \begin{pmatrix} 1^3 & 2^3 & \cdots & k^3 \\ 1^7 & 2^7 & \cdots & k^7 \\ \vdots & \vdots & \vdots & \vdots \\ 1^{4k-1} & 2^{4k-1} & \cdots & k^{4k-1} \end{pmatrix} \begin{pmatrix} 2|\beta_1|\sin(x+\theta_1) \\ 2|\beta_2|\sin(2x+\theta_2) \\ \vdots \\ 2|\beta_k|\sin(kx+\theta_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Observe that the matrices we obtain are minors of Vandermonde Matrices (up to their sign), and as such are non-singular [8]. It follows that

$$\begin{pmatrix} 2|\beta_1|\sin(x+\theta_1) \\ 2|\beta_2|\sin(2x+\theta_2) \\ \vdots \\ 2|\beta_k|\sin(kx+\theta_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} 2|\beta_1|\cos(x+\theta_1) \\ 2|\beta_2|\cos(2x+\theta_2) \\ \vdots \\ 2|\beta_k|\cos(kx+\theta_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Recall that we assume at least one β_j is nonzero for some $1 \leq j \leq k$, so we have $\cos(jx+\theta_j) = \sin(jx+\theta_j) = 0$, which is clearly a contradiction. We thus conclude the proof. ◀

By Proposition 6, it follows that g is not constant, and therefore $f(x,y)$ is not constant on the curve $x^2+y^2=1$. By Bezout's Theorem, we have that Z_f is finite, and consists of at most $4k$ points. Moreover, f is a semialgebraic function (that is, its graph $\{(x,y,f(x,y)) : x,y \in \mathbb{R}\}$ is semialgebraic set in $\mathbb{R}^3$). Thus, the points in Z_f have semialgebraic coordinates, and we can compute them. By identifying $\mathbb{R}^2$ with $\mathbb{C}$, denote $Z_f = \{z_1, \ldots, z_{4k}\}$.

▶ Remark 7. Since the polynomial f has algebraic coefficients, it is not immediately clear how the degree and height of the points in Z_f relate to $\|f\|$. However, recall that the algebraic coefficients in f are polynomials in the entries of $A^n s$, which are, in turn, algebraic numbers of degree at most 3 whose description is polynomial in that of A and s.

Thus, we can define Z_f with a formula in the first-order theory of the reals with a fixed number of variables. Using results of Renegar [19], we show in the full version that the points in Z_f have semialgebraic coordinates with description length polynomial in $\|f\|$.

We now employ the following lemma from [16], which is itself a consequence of the Baker-Wüstholz Theorem [2].

▶ **Lemma 8** ([16]). *There exists $D \in \mathbb{N}$ such that for all algebraic numbers ζ, ξ of modulus 1, and for every $n \geq 2$, if $\zeta^n \neq \xi$, then $|\zeta^n - \xi| > \frac{1}{n^{(\|\zeta\|+\|\xi\|)^D}}$.*

Since γ is not a root of unity, it holds that $\gamma^{n_1} \neq \gamma^{n_2}$ for every $n_1 \neq n_2 \in \mathbb{N}$. Thus, there exists a computable $N_1 \in \mathbb{N}$ such that $\gamma^n \notin Z_f$ for every $n > N_1$. Moreover, by [5, Lemma D.1], we have that $N_1 = \|f\|^{\mathcal{O}(1)}$. By Lemma 8, there exists a constant $D \in \mathbb{N}$ such that for every $n \geq N_1$ and $1 \leq j \leq 4k$ we have that $|\gamma^n - z_j| > \frac{1}{n^{(\|f\|^D)}}$ (since $\|z_j\| + \|\gamma\| = \mathcal{O}(\|f\|)$). Intuitively, for $n > N_1$ we have that γ^n does not get close to any z_i "too quickly" as a function of n. In particular, for $n > N_1$ we have $f(\gamma^n) \neq 0$. It thus remains to show that for

large enough n, $r(n)$ does not affect the sign of $f(\gamma^n) + r(n)$. Intuitively, this is the case because $r(n)$ decreases exponentially, while $|f(\gamma^n)|$ is bounded from below by an inverse polynomial.

For every $z_j \in Z_f$, let $\varphi_j = \arg z_j$, so that $f(z) = 0$ iff $g(\varphi_j) = 0$. We assume w.l.o.g. that $\varphi_j \in (-\pi, \pi)$ for every $1 \leq j \leq 4k$. Indeed, if $\varphi_j = \pi$ for some j, then we can shift the domain of g slightly so that all zeros are in the interior.

For every $1 \leq j \leq 4k$, let T_j be the Taylor polynomial of g around φ_j such that the degree d_j of T_j is minimal and T_j is not identically 0. Thus, we have $T_j(x) = \frac{g^{(d_j)}(\varphi_j)}{d_j!}(x - \varphi_j)^{d_j}$. By Proposition 6 we have that $d_j \leq 4k$ for every j. In addition, the description of T_j is computable from that of $\|f\|$.

By Taylor's inequality, we have that for every $x \in [-\pi, \pi]$ it holds that $|g(x) - T_j(x)| \leq \frac{M_j |x-\varphi_j|^{d_j+1}}{(d_j+1)!}$ where $M_j = \max_{x\in[-\pi,\pi]} \left\{g^{(d_j+1)}(x)\right\}$ (where g is extended naturally to the domain $[-\pi, \pi]$). By our description of $g^{(d_j+1)}(x)$, we see that M_j is bounded by $M = 4k \max_{1\leq i\leq k} \{|\beta_i|\} k^{4k+1}$.

Let $\epsilon_1 > 0$ be such that the following conditions hold for every $1 \leq j \leq 4k$.

1. $\text{sign}(g'(x))$ does not change in $(\varphi_j, \varphi_j + \epsilon_1)$ nor in $(\varphi_j - \epsilon_1, \varphi_j)$.
2. $|g(x) - T_j(x)| \leq \frac{1}{2}|T_j(x)|$ for every $x \in (\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$.
3. $\text{sign}(g'(x)) = \text{sign}(T_j'(x))$ for every $x \in (\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$.

Note that we can assume $(\varphi_j - \epsilon_1, \varphi_j + \epsilon_1) \subseteq (-\pi, \pi)$, since by our assumption $\varphi_j \in (-\pi, \pi)$ for all $1 \leq j \leq 4k$.

An ϵ_1 as above exists due to the following properties (see Figure 1 for an illustration):

- There are only finitely many points where $g'(x) = 0$,
- $T_j(x)$ is of degree d_j, whereas $|g(x) - T_j(x)|$ is upper-bounded by a polynomial of degree $d_j + 1$, and
- $T_j'(x)$ is the Taylor polynomial of degree $d_j - 1$ of $g'(x)$ around φ_j, so by bounding the distance $|g'(x) - T_j'(x)|$ we can conclude the third requirement.

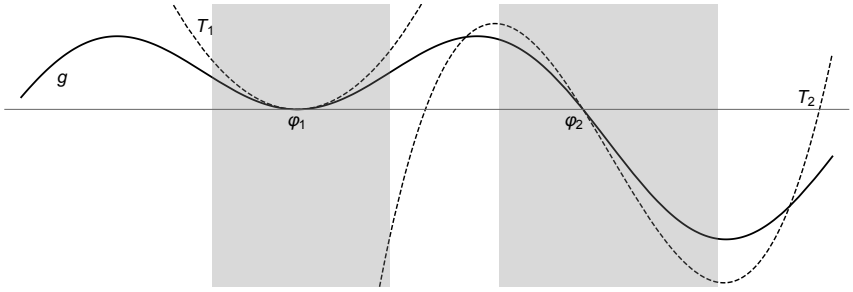


Figure 1 $g(x)$ and two Taylor polynomials: $T_1(x)$ around φ_1 and $T_2(x)$ around φ_2. The shaded regions show where requirements (1)–(3) hold, which determine ϵ_1. Observe that for T_1, the most restrictive requirement is $|g(x) - T_1(x)| \leq \frac{1}{2}T_1(x)$, whereas for T_2 the restriction is the requirement that $T_2(x)$ is monotone.

In order to establish Lemma 4, we must be able to effectively compute ϵ_1. We thus proceed with the following lemma.

▶ **Lemma 9.** *ϵ_1 can be computed in polynomial time in $\|f\|$, and $\frac{1}{\epsilon_1} = 2^{\|f\|^{\mathcal{O}(1)}}$.*

Proof. We compute $\delta_1, \delta_2, \delta_3$ that satisfy requirements 1,2, and 3, respectively. Then, taking $\epsilon_1 = \min\{\delta_1, \delta_2, \delta_3\}$ will conclude the proof.

Condition 1. We compute $\delta_1 > 0$ such that $\text{sign}(g'(x))$ does not change in $(\varphi_j - \delta_1, \varphi_j)$ nor in $(\varphi_j, \varphi_j + \delta_1)$. This is done as follows. Recall that $g(x) = f(e^{ix}) = \sum_{m=0}^{k} \beta_m e^{imx} + \overline{\beta_m} e^{-imx}$. It is not hard to check that $g'(x) = \sum_{m=0}^{k} im\beta_m e^{imx} + \overline{im\beta_m} e^{-imx}$. Let $\widehat{f}(z) : \mathbb{C} \to \mathbb{R}$ be the function $\widehat{f}(z) = \sum_{m=0}^{k} im\beta_m z + \overline{im\beta_m}\overline{z}$, then $g'(x) = \widehat{f}(e^{ix})$ and $\left\|\widehat{f}\right\| = \mathcal{O}(\|f\|)$.

Consider the algebraic set $F = \left\{z : |z| = 1 \wedge \widehat{f}(z) = 0\right\}$, then $\{x : g'(x) = 0\} = \{\arg(z) : z \in F\}$. By similar arguments as those by which we found the roots of f on the unit circle, namely by adapting Proposition 6 to $\widehat{f}$, we can conclude that F contains at most $4k$ points. Thus, it is enough to set δ_1 such that $\left(\bigcup_{j=1}^{4k}(\varphi_j - \delta_1, \varphi_j) \cup (\varphi_j, \varphi_j + \delta_1)\right) \cap F = \emptyset$.

By Equation (1), we have that for $z \neq z' \in F$ it holds that $|z - z'| > \frac{\sqrt{6}}{d^{\frac{d+1}{2}} \cdot H^{d-1}}$ where d and H are the degree and height of the roots of $\widehat{f}(z)$ (see Remark 7). Thus, $1/|z - z'|$ is $2^{\|f\|^{\mathcal{O}(1)}}$, and has a polynomial description. Since $|\arg(z) - \arg(z')| > |z - z'|$, we conclude that by setting $\delta_1 = \min\{|z - z'| : z \neq z' \in F\}/3$, it holds that $\frac{1}{\delta_1}$ has a polynomial description in $\|f\|$, and δ_1 satisfies the required condition.

Condition 2. Next, we compute $\delta_2 > 0$ such that $|g(x) - T_j(x)| \leq \frac{1}{2}|T_j(x)|$ for every $x \in (\varphi_j - \delta_2, \varphi_j + \delta_2)$. Recall that $T_j(x) = \frac{g^{(d_j)}(\varphi_j)}{d_j!}(x - \varphi_j)^{d_j}$. Note that this case is more challenging than Condition 1, as unlike $g(x) = f(e^{ix})$, the polynomial $T_j(x)$ has potentially transcendental coefficients (namely φ_j). For clarity, we omit the index j in the following. Thus, we write T, d, φ instead of T_j, d_j, φ_j, etc.

In order to ignore the absolute value, assume $T(x) \geq g(x) > 0$ in an interval $(\varphi, \varphi + \xi)$ for some $\xi > 0$ (the other cases are treated similarly). Then, the inequality above becomes $g(x) - \frac{1}{2}T(x) \geq 0$. Since the degree of T is d, then by the definition of T, the first $d - 1$ derivatives of g in φ vanish. Define $h(x) = g(x) - \frac{1}{2}T(x)$, then we have $h(\varphi) = 0$, $h'(\varphi) = 0, \ldots, h^{(d-1)}(\varphi) = 0$ and $h^{(d)}(\varphi) = g^{(d)}(\varphi) - \frac{1}{2}g^{(d)}(\varphi) = \frac{1}{2}g^{(d)}(\varphi)$. By our assumption, $T(x) \geq \frac{1}{2}T(x)$ for $x \in (\varphi, \varphi + \xi)$, so $h^{(d)}(\varphi) > 0$. In addition, recall that $|h^{(d+1)}(x)| = |g^{(d+1)}(x)| \leq M$ for every $x \in [-\pi, \pi]$. Thus, by writing the d-th Taylor expansion of $h(x)$ around φ, we have that $h(x) = \frac{h^{(d)}(\varphi)}{d!}(x - \varphi)^d + E(x)$ where $|E(x)| \leq \frac{M}{(d+1)!}(x - \varphi)^{d+1}$. We now have that

$$h(x) \geq \frac{1}{2}\frac{g^{(d)}(\varphi)}{d!}(x - \varphi)^d - \frac{M}{(d+1)!}(x - \varphi)^{d+1}.$$

Taking $x \in (\varphi, \varphi + \frac{g^{(d)}(\varphi)(d+1)}{2M})$, it is easy to check that $h(x) \geq 0$. We can now set $\delta_2 = \frac{g^{(d)}(\varphi)(d+1)}{2M}$, which satisfies the required condition.

Condition 3. Finally, we compute $\delta_3 > 0$ such that $\text{sign}(g'(x)) = \text{sign}(T_j'(x))$ for every $x \in (\varphi_j - \delta_3, \varphi_j + \delta_3)$. Observe that $T_j'(x)$ is the $d_j - 1$-th Taylor polynomial of $g'(x)$ around φ^j. Thus, by following the reasoning used to find δ_2, we can find δ_3 such that $|g'(x) - T_j'(x)| \leq \frac{1}{2}|T_j'(x)|$ for every $x \in (\varphi - \delta_3, \varphi + \delta_3)$, and in particular it holds that $\text{sign}(g'(x)) = \text{sign}(T_j'(x))$ for every $x \in (\varphi_j - \delta_3, \varphi_j + \delta_3)$.

As mentioned above, by setting $\epsilon_1 = \min\{\delta_1, \delta_2, \delta_3\}$, we conclude the proof. ◀

Conditions 1,2, and 3 above imply that within the intervals $(\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$ we have that $|g(x)| \geq \frac{1}{2}|T_j(x)|$, that $g(x)$ and $T_j(x)$ have the same sign, and that they are both decreasing/increasing together.

We now claim that there exists a polynomial $p(n)$ and a number $N_2 \in \mathbb{N}$ such that for every $n > N_2$ it holds that $|g(\arg(\gamma^n))| > \frac{1}{p(n)}$. In order to compute $p(n)$, we compute separate polynomials for the domain $\bigcup_{j=1}^{4k}(\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$ and for its complement. Then, taking their minimum and bounding it from below by another polynomial yields $p(n)$.

We start by considering the case where $\arg(\gamma^n) \in \bigcup_{j=1}^{4k}(\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$. Recall that since γ is not a root of unity, then for every $n > N_1$ it holds that $\gamma^n \notin Z_f = \{z_1, \ldots, z_{4k}\}$. Then, by Lemma 8, for every $1 \leq j \leq 4k$ and every $n \geq N_2 = \max\{N_1, 2\}$ we have $|\gamma^n - z_j| > \frac{1}{n^{(\|f\|^{D})}}$. In addition, $|\gamma^n - z_j| \leq |\arg(\gamma^n) - \varphi_j|$ (since the LHS is the Euclidean distance and the RHS is the spherical distance). Therefore, $|\arg(\gamma^n) - \varphi_j| > \frac{1}{n^{(\|f\|^{D})}}$, so either $\arg(\gamma^n) > \varphi_j + \frac{1}{n^{(\|f\|^{D})}}$ or $\arg(\gamma^n) < \varphi_j - \frac{1}{n^{(\|f\|^{D})}}$. Next, we have that if $\arg(\gamma^n) \in (\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$ for some $1 \leq j \leq 4k$, then $|g(\arg(\gamma^n))| \geq \frac{1}{2}|T_j(\arg(\gamma^n))| \geq \frac{1}{2}\min\left\{|T_j(\varphi_j + \frac{1}{n^{(\|f\|^{D})}})|, |T_j(\varphi_j - \frac{1}{n^{(\|f\|^{D})}})|\right\}$, where the last inequality follows from condition 3 above, which implies that T_j is monotone with the same tendency as g.

Observe that $T_j(\varphi_j - \frac{1}{n^{(\|f\|^{D})}}) = \frac{g^{(d_j)}(\varphi)}{d_j!}\frac{1}{n^{(\|f\|^{D})}}$ and that similarly $T_j(\varphi_j + \frac{1}{n^{(\|f\|^{D})}}) = -\frac{g^{(d_j)}(\varphi)}{d_j!}\frac{1}{n^{(\|f\|^{D})}}$ are both inverse polynomials (in n). Thus, $|g(\arg(\gamma^n))|$ is bounded from below by an inverse polynomial. Moreover, these polynomials can be easily computed in time polynomial in $\|f\|$.

Finally, we note that for $x \notin \bigcup_{j=1}^{4k}(\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)$ we can compute in polynomial time a bound $B > 0$ such that $|g(x)| > B$. Indeed, $B = \min\{|g(x)| : x \in [-\pi, \pi] \setminus \bigcup_{j=1}^{4k}(\varphi_j - \epsilon_1, \varphi_j + \epsilon_1)\}$ (where $g(-\pi)$ is defined naturally by extending the domain), and we have that $|B| > 0$ since we assumed non of the φ_j are exactly at π (in which case we would have had $g(-\pi) = 0$). In particular, we can combine the two domains and compute a polynomial p as required. We remark that we can compute $\|B\|$ in polynomial time, since it is either at least $\frac{1}{2}|T_j(\varphi_j \pm \epsilon_1)|$ for some $1 \leq j \leq 4k$ (and by Lemma 9, $\|\epsilon_1\|$ can be computed in polynomial time), or it is the value of one of the extrema of g, and the latter can be computed by finding the extrema of the (algebraic) function f on the unit circle.

To recap, for every $n > N_2$ it holds that $|g(\arg(\gamma^n))| > \frac{1}{p(n)}$ for a non-negative polynomial p, and both N_2 and p can be computed in polynomial time in the description of the input.

Next, we wish to find $N_3 \in \mathbb{N}$ such that for every $n > N_3$ it holds that $r(n) < \frac{1}{p(n)}$. Recall that $r(n) = \sum_{l=1}^{k'} \chi_l \mu_l^n + \overline{\chi_l \mu_l}^n$ where for every $1 \leq l \leq k'$ we have that μ_l is algebraic with $\deg(\mu_l) = \|f\|^{\mathcal{O}(1)}$ and $H(\mu_l) = 2^{\|f\|^{\mathcal{O}(1)}}$. Observe that $1 - |\mu_l|$ is also an algebraic number. Indeed, $1 - |\mu_l| = 1 - \sqrt{\mu_l \overline{\mu_l}}$. Moreover, we get that $\deg(1 - |\mu_l|) \leq \deg(\mu_l)^4$, as it is the root of a polynomial of degree at most $\deg(\mu_l)^4$, and that $H(1 - |\mu_l|)$ is polynomial in $H(\mu_l)$. Since $|\mu_l| < 1$, by applying Equation (1), we get $1 - |\mu_l| = |1 - |\mu_l|| > \frac{\sqrt{6}}{d^{(d+1)/2}H(\mu_l)^{d-1}}$ where $d = \deg(\mu_l)^{\mathcal{O}(1)}$ and $H(\mu_l) = 2^{\|I\|^{\mathcal{O}(1)}}$. It follows that we can compute $\delta \in (0, 1)$ with $\frac{1}{\delta} = 2^{\|I\|^{\mathcal{O}(1)}}$ such that $1 - |\mu_l| > \delta$, and hence $|\mu|^n < 1 - \delta$. Thus,

$$|r(n)| \leq \sum_{l=1}^{k'} 2|\chi_l||\mu_l|^{mn} \leq \sum_{l=1}^{k'} 2|\chi_l|(1-\delta)^{mn} \leq 2k' \max_{1 \leq l \leq k'} |\chi_l|(1-\delta)^n$$

We can now compute $\epsilon \in (0, 1)$ and $N_3 \in \mathbb{N}$ such that:

1. $\frac{1}{\epsilon} = 2^{\|I\|^{\mathcal{O}(1)}}$
2. $N_3 = 2^{\|I\|^{\mathcal{O}(1)}}$
3. For every $n > N_3$ it holds that $|r(n)| < (1 - \epsilon)^n$

Finally, by taking $N_4 \in \mathbb{N}$ such that $(1 - \epsilon)^n < \frac{1}{p(n)}$ (which satisfies $N_4 = 2^{\|I\|^{\mathcal{O}(1)}}$) for all $n > N_4$, we can now conclude that for every $n > \max\{N_2, N_3, N_4\}$, the following hold.

1. $f(\gamma^n) = g(\arg(\gamma^n)) \neq 0$.
2. If $f(\gamma^n) > 0$, then $g(\arg(\gamma^n)) > 0$, so $g(\arg(\gamma^n)) > \frac{1}{p(n)}$. Since $|r(n)| < \frac{1}{p(n)}$, it follows that $f(\gamma^n) + r(n) = g(\arg(\gamma^n)) + r(n) > \frac{1}{p(n)} - |r(n)| > 0$. Conversely, if $f(\gamma^n) + r(n) > 0$, then $g(\arg(\gamma^n)) + r(n) > 0$, but since $|g(\arg(\gamma^n))| > \frac{1}{p(n)}$ and $|r(n)| < \frac{1}{p(n)}$, then it must hold that $g(\arg(\gamma^n)) > 0$, so $f(\gamma^n) > 0$.

3. If $f(\gamma^n) < 0$, then $g(\arg(\gamma^n)) < 0$, so $g(\arg(\gamma^n)) < -\frac{1}{p(n)}$. Since $|r(n)| < \frac{1}{p(n)}$, it follows that $f(\gamma^n)+r(n) = g(\arg(\gamma^n))+r(n) < -\frac{1}{p(n)}+|r(n)| < 0$. Conversely, if $f(\gamma^n)+r(n) < 0$, then $g(\arg(\gamma^n)) + r(n) < 0$, but since $|g(\arg(\gamma^n))| > \frac{1}{p(n)}$ and $|r(n)| < \frac{1}{p(n)}$, then it must hold that $g(\arg(\gamma^n)) < 0$, so $f(\gamma^n) < 0$.

This concludes the proof of Lemma 4. ◀

We are now ready to use Lemma 4 in order to solve the systems.

▶ **Theorem 10.** *The problem of deciding whether an almost self-conjugate system has a solution is decidable.*

Proof. Consider an almost self-conjugate system of the form $\bigwedge_J R_J(A^n s) \sim_J 0$. For each expression $R_J(A^n s) \sim_J 0$, let f be the corresponding dominant function, as per Lemma 4, and compute its respective bound N. If $\sim_J$ is "=", then by Lemma 4, if the equation is satisfiable for $n \in \mathbb{N}$, then $n < N$.

If all the $\sim_J$ are ">", then for each such inequality compute $\{z : f(z) > 0\}$, which is a semialgebraic set. If the intersection of these sets is empty, then if n is a solution for the system, it must hold that $n < N$. If the intersection is non-empty, then it is an open set. Since γ is not a root of unity, then $\{\gamma^n : n \in \mathbb{N}\}$ is dense in the unit circle. Thus, there exists $n > N$ such that γ^n is in the above intersection, so the system has a solution. Checking the emptiness of the intersection can be done using Theorem 1.

Thus, it remains to check whether there exists a solution $n < N$, which is clearly decidable. ◀

Observe that from Theorem 10, combined with Section 3.1, we can conclude the decidability of the point-to-semialgebraic Orbit Problem. However, as it turns out, we can reuse Theorem 10 to obtain a much stronger result, namely the decidability of the Semialgebraic Orbit Problem.

4 The Semialgebraic Orbit Problem

In [1], we proved that the following problem is decidable: given two polytopes $S, T \subseteq \mathbb{R}^3$ and a matrix $A \in \mathbb{Q}^{3\times 3}$, does there exist $n \in \mathbb{N}$ such that $A^n S \cap T \neq \emptyset$. We now show that the techniques developed here can be used as an alternative solution for this problem, and in fact solve a much stronger variant, where S and T are replaced by semialgebraic sets. That is, given two semialgebraic sets $S, T \subseteq \mathbb{R}^3$ and a matrix $A \in \mathbb{Q}^{3\times 3}$, does there exist $n \in \mathbb{N}$ such that $A^n S \cap T \neq \emptyset$.

▶ **Theorem 11.** *The Semialgebraic Orbit Problem is decidable.*

Proof. Consider semialgebraic sets $S, T \subseteq \mathbb{R}^3$ and a matrix $A \in \mathbb{Q}^{3\times 3}$, as described above. Recall that we can write $S = \{\vec{x} : \bigvee_I \bigwedge_J R_{I,J}(\vec{x}) \sim_{I,J} 0\}$ and similarly for T. Since we want to decide whether some point in S hits T, we can consider each disjunct in the description of S separately. Thus, we henceforth assume $S = \{\vec{x} : \bigwedge_J R_J(\vec{x}) \sim_J 0\}$.

We now turn to characterise the set $A^n S$ for every $n \in \mathbb{N}$. For this purpose, we assume A is invertible. The case where A is not invertible can be reduced to analysis in a lower dimension, and is handled in the full version. For every $n \in \mathbb{N}$, we now have

$$A^n S = \left\{ A^n \vec{x} : \bigwedge_J R_J(\vec{x}) \sim_J 0 \right\} = \left\{ \vec{x} : \bigwedge_J R_J((A^{-1})^n \vec{x}) \sim_J 0 \right\}.$$

We further assume that A has a complex eigenvalue. As in Section 3, the case where all eigenvalues are real is simpler (even if A is not diagonalisable), and is handled in the full version. We can now write $A = PDP^{-1}$ with $D = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \overline{\lambda} & 0 \\ 0 & 0 & \rho \end{pmatrix}$, where λ is a complex eigenvalue, $\rho \in \mathbb{R}$, and P an invertible matrix. We thus have $A^{-1} = PD^{-1}P^{-1}$ where $D^{-1} = \begin{pmatrix} \frac{\overline{\lambda}}{|\lambda|^2} & 0 & 0 \\ 0 & \frac{\lambda}{|\lambda|^2} & 0 \\ 0 & 0 & \rho^{-1} \end{pmatrix}$. We denote $\zeta = \frac{\overline{\lambda}}{|\lambda|^2}$ and $\eta = \rho^{-1}$, so $D^{-1} = \begin{pmatrix} \zeta & 0 & 0 \\ 0 & \overline{\zeta} & 0 \\ 0 & 0 & \eta \end{pmatrix}$. As in Section 3, by analysing the structure of P and P^{-1}, we have that for $\vec{x} = (x_1, x_2, x_3)$, $(A^{-1})^n(\vec{x})_i = \sum_{j=1}^{3}(a_{i,j}\zeta^n + \overline{a_{i,j}}\overline{\zeta}^n + b_{i,j}\eta^n)x_j$ with $a_{i,j} \in \mathbb{A}$ and $b_{i,j} \in \mathbb{A} \cap \mathbb{R}$. That is, each coordinate $1 \leq i \leq 3$, is a linear combination of x_1, x_2, x_3 where the coefficients are of the form above. In particular, the coefficient of every x_j is an almost self-conjugate polynomial (see the full version for a complete analysis).

Consider a monomial of the form $x_1^{s_1}x_2^{s_2}x_3^{s_3}$ in $R_J(\vec{x})$. Replacing $\vec{x}$ with $(A^{-1})^n\vec{x}$, the monomial then becomes $Q(\zeta^n, \overline{\zeta}^n, \eta^n)x_1^{s_1}x_2^{s_2}x_3^{s_3}$, where $Q(z_1, z_2, z_3)$ is an almost self-conjugate polynomial. Indeed, this follows since the coordinates of $(A^{-1})^n\vec{x}$ above are almost self-conjugate, and products of almost self-conjugate polynomials remain almost self-conjugate.

Recall that the polynomials R_J in the description of S have integer (and in particular, real) coefficients. By lifting the discussion about monomials to R_J, we can write

$$R_J((A^{-1})^n(\vec{x})) = \sum_{0 \leq s_1, s_2, s_3 \leq k} Q^J_{s_1,s_2,s_3}(\zeta^n, \overline{\zeta}^n, \eta^n)x_1^{s_1}x_2^{s_2}x_3^{s_3}$$

where $k \in \mathbb{N}$ and the coefficients $Q^J_{s_1,s_2,s_3}$ are almost self-conjugate.

Observe that now, there exists $n \in \mathbb{N}$ such that $A^nS \cap T \neq \emptyset$ iff there exists $n \in \mathbb{N}$ and $\vec{x} \in \mathbb{R}^3$ such that $\vec{x} \in T$ and

$$\bigwedge_J \sum_{0 \leq s_1, s_2, s_3 \leq k} Q^J_{s_1,s_2,s_3}(\zeta^n, \overline{\zeta}^n, \eta^n)x_1^{s_1}x_2^{s_2}x_3^{s_3} \sim_J 0. \tag{4}$$

Intuitively, we now want to eliminate the quantifiers on $\vec{x}$ in the expression above. However, we cannot readily do so, as the expression is also quantified by $n \in \mathbb{N}$. Nonetheless, in the following we manage to circumvent this problem by increasing the dimension of the problem.

Let K be the number of polynomials $Q^J_{s_1,s_2,s_3}$ that appear in the conjunction (4) above, indexed by J, s_1, s_2, s_3. Consider the set

$$U = \left\{ (y_1, \ldots, y_K) \in \mathbb{R}^K : \begin{array}{l} \exists \vec{x} \in \mathbb{R}^3,\ x \in T \wedge \\ \bigwedge_J \sum_{0 \leq s_1, s_2, s_3 \leq k} y^J_{s_1,s_2,s_3} x_1^{s_1}x_2^{s_2}x_3^{s_3} \sim_J 0 \end{array} \right\}$$

That is, U is obtained by replacing each polynomial $Q^J_{s_1,s_2,s_3}$ with a "placeholder" real variable $y^J_{s_1,s_2,s_3}$. U is clearly a semialgebraic set, so by Theorem 2, we can eliminate the quantifier on $\vec{x}$, and write

$$U = \left\{ (y_1, \ldots, y_K) \in \mathbb{R}^K : \bigwedge_J S_J(y_1, \ldots, y_K) \sim_J 0 \right\}$$

where S_J are polynomials with integer coefficients. It is now the case that there exists $n \in \mathbb{N}$ such that $A^nS \cap T \neq \emptyset$ iff there exists $n \in \mathbb{N}$ such that $(Q_1(\zeta^n, \overline{\zeta}^n, \eta^n), \ldots, Q_K(\zeta^n, \overline{\zeta}^n, \eta^n)) \in U$. That is, we need to decide whether there exists $n \in \mathbb{N}$ such that $S_J(Q_1(\zeta^n, \overline{\zeta}^n, \eta^n), \ldots, Q_K(\zeta^n, \overline{\zeta}^n, \eta^n)) \sim_J 0$ for every J.

It is easy to see that since the polynomials Q_i are almost self-conjugate, then so is $S_J(Q_1(\zeta^n, \overline{\zeta}^n, \eta^n), \ldots, Q_K(\zeta^n, \overline{\zeta}^n, \eta^n))$, (when viewed as a polynomial in $\zeta^n, \overline{\zeta}^n, \eta^n$).

Thus, the conjunction

$$\bigwedge_J S_J(Q_1(\zeta^n, \overline{\zeta}^n, \eta^n), \ldots, Q_K(\zeta^n, \overline{\zeta}^n, \eta^n))$$

is an almost self-conjugate system, and by Theorem 10, it is decidable whether it has a solution. This concludes the proof. ◀

5 Discussion

This paper establishes the decidability of the Semialgebraic Orbit Problem in dimension at most three. The class of semialgebraic sets is arguably the largest natural class of subsets of $\mathbb{R}^n$ for which membership is decidable. Thus, our results reach the limit of what can be decided about the orbit of a single matrix. Moreover, our techniques shed light on the decidability (or hardness) of orbit problems in higher dimensions: the techniques we develop for analysing orbits can be applied to any matrix (in any dimension) whose eigenvalues have arguments that are pairwise linearly dependent over $\mathbb{Q}$ (i.e., the arguments of all the eigenvalues are rational multiples of some angle θ). Indeed, it is easy to see that the orbits generated by such matrices can be reduced to solving almost self-conjugate systems (see Section 3). This can be put in contrast to known hardness results [4] in dimension $d \geq 4$, which require a single pair of eigenvalues whose arguments do not satisfy the above property. Thus, we significantly sharpen the border of known decidability, and allow future research to focus on hard instances.

Technically, our contribution uncovers two interesting tools. First, the identification of almost self-conjugate polynomials, and their amenability to analysis (Section 3), and second, the ability to abstract away integral exponents in order to perform quantifier elimination, by increasing the dimension (Section 4). The former arises naturally in the context of matrix exponentiation, while the latter is an obstacle that is often encountered when quantifying over semialgebraic sets in the presence of a discrete operator (e.g., matrix exponentiation). In the future, we plan to further investigate the applications of these directions.

References

1. S. Almagor, J. Ouaknine, and J. Worrell. The Polytope-Collision Problem. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 24:1–24:14, 2017.
2. A. Baker and G. Wüstholz. Logarithmic forms and group varieties. *J. reine angew. Math*, 442(19-62):3, 1993.
3. S. Basu, R. Pollack, and M-F. Roy. *Algorithms in real algebraic geometry*, volume 20033. Springer, 2005.
4. V. Chonev, J. Ouaknine, and J. Worrell. The polyhedron-hitting problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 940–956. SIAM, 2015.
5. V. Chonev, J. Ouaknine, and J. Worrell. On the Complexity of the Orbit Problem. *J. ACM*, 63(3):23:1–23:18, 2016.
6. H. Cohen. *A course in computational algebraic number theory*, volume 138. Springer Science & Business Media, 2013.
7. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pages 134–183. Springer, 1975.

8 F.R. Gantmacher. *The Theory of Matrices*. Number v. 2 in The Theory of Matrices. Chelsea Publishing Company, 1959.

9 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's Problem – On the Border between Decidability and Undecidability. Technical Report 683, Turku Centre for Computer Science, 2005.

10 M. A Harrison. Lectures on linear sequential machines. Technical report, DTIC Document, 1969.

11 R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM (JACM)*, 33(4):808–821, 1986.

12 G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic Reachability Computation for Families of Linear Vector Fields. *J. Symb. Comput.*, 32(3):231–253, 2001.

13 M. Mignotte. Some useful bounds. In *Computer algebra*, pages 259–263. Springer, 1983.

14 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.

15 M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Inf. Process. Lett.*, 91(5):233–244, 2004.

16 J. Ouaknine and J. Worrell. Ultimate Positivity is decidable for simple linear recurrence sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 330–341. Springer, 2014.

17 V. Y. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. *Computers & Mathematics with Applications*, 31(12):97–138, 1996.

18 J. Renegar. A Faster PSPACE Algorithm for Deciding the Existential Theory of the Reals. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 291–295, 1988.

19 J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. Part I: Introduction. Preliminaries. The geometry of semi-algebraic sets. The decision problem for the existential theory of the reals. *Journal of symbolic computation*, 13(3):255–299, 1992.

20 T. Tao. *Structure and randomness: pages from year one of a mathematical blog*. American Mathematical Soc., 2008.

21 A. Tarski. *A decision method for elementary algebra and geometry*. Rand Corporation, 1951.

22 N. K. Vereshchagin. Occurrence of zero in a linear recursive sequence. *Mathematical notes of the Academy of Sciences of the USSR*, 38(2):609–615, 1985.

Best-Of-Two-Worlds Analysis of Online Search

Spyros Angelopoulos
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
Spyros.Angelopoulos@lip6.fr

Christoph Dürr
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
Christoph.Durr@lip6.fr

Shendan Jin
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
Shendan.Jin@lip6.fr

Abstract

In search problems, a mobile searcher seeks to locate a target that hides in some unknown position of the environment. Such problems are typically considered to be of an on-line nature, in that the input is unknown to the searcher, and the performance of a search strategy is usually analyzed by means of the standard framework of the competitive ratio, which compares the cost incurred by the searcher to an optimal strategy that knows the location of the target. However, one can argue that even for simple search problems, competitive analysis fails to distinguish between strategies which, intuitively, should have different performance in practice.

Motivated by the above, in this work we introduce and study measures *supplementary* to competitive analysis in the context of search problems. In particular, we focus on the well-known problem of *linear search*, informally known as the *cow-path problem*, for which there is an infinite number of strategies that achieve an optimal competitive ratio equal to 9. We propose a measure that reflects the rate at which the line is being explored by the searcher, and which can be seen as an extension of the *bijective ratio* over an uncountable set of requests. Using this measure we show that a natural strategy that explores the line aggressively is optimal among all 9-competitive strategies. This provides, in particular, a strict separation from the competitively optimal doubling strategy, which is much more conservative in terms of exploration. We also provide evidence that this aggressiveness is requisite for optimality, by showing that any optimal strategy must mimic the aggressive strategy in its first few explorations.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online computation, search problems, linear search, performance measures

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.7

Funding Supported by ANR OATA, ANR ENERGUMEN, DIM RFSI DACM and Labex Mathématique Hadamard. This research also benefited from the support of the FMJH Program PGMO and from the support of EDF-Thales-Orange.

Acknowledgements We are thankful to Elmar Langetepe for discussions on the literature of online search, as well as to Pascal Schweitzer for comments on an early draft of this paper.

1 Introduction

Searching for a hidden target is an important paradigm in computer science and operations research, with numerous applications. A typical search problem involves an environment, a mobile *searcher* (who may, or may not, have knowledge of the environment) and a *hider* (sometimes also called *target*) who hides at some position within the environment that is oblivious to the searcher. The objective is to define a *search strategy*, i.e., a traversal of the environment, that optimizes a certain efficiency criterion. A standard approach to the latter is by means of *competitive analysis*, in which we seek to minimize the worst-case cost for

locating the target, divided by some concept of "optimal" solution; e.g., the minimum cost to locate the target once its position is known. Even prior to the advent of online computation and competitive analysis, search games had already been studied under such *normalized* measures within operations research [9]. Explicit studies of the competitive ratio and the closely related *search ratio* were given in [7] and [28], respectively, and led to the development of *online searching* [24, 11] as a subfield of online computation. See also [1] for an in-depth treatment of search games, including the role of payoff functions that capture the competitive ratio.

In this work we revisit one of the simplest, yet fundamental search problems, namely the *linear search*, or, informally, *cow-path* problem. The setting involves an infinite (i.e., unbounded) line, with a point O designated as its origin, a searcher which is initially placed at the origin, and an immobile target which is at some position on the line that is unknown to the searcher. More specifically, the searcher does not know whether the hider is at the left branch or at the right branch of the line. The searcher's *strategy* S defines its exploration of the line, whereas the hider's strategy H is determined by its placement on the line. Given strategies S, H, the *cost* of locating the hider, denoted by $c(S, H)$ is the total distance traversed by the searcher at the first time it passes over H. Let $|H|$ denote the distance of the hider from the origin. The competitive ratio of S, denoted by $\mathrm{cr}(S)$, is the worst-case normalized cost of S, among all possible hider strategies. Formally,

$$\mathrm{cr}(S) = \sup_{H} \frac{c(S, H)}{|H|}. \tag{1}$$

It has long been known [8, 20] that the competitive ratio of linear search is 9, and is achieved by a simple *doubling* strategy: in iteration i, the searcher starts from O, explores branch $i \bmod 2$ at a length equal to 2^i, and then returns to O. However, this strategy is not uniquely optimal; in fact, it is known that there is an infinite number of competitively optimal strategies for linear search (see Lemma 6 in Section 3). In particular, consider an "aggressive" strategy, which in each iteration searches a branch to the maximum possible extent, while maintaining a competitive ratio equal to 9. This can be achieved by searching, in iteration i, branch $i \bmod 2$ to a length equal to $(i + 2)2^{i+1}$ (see Corollary 8).

While both `doubling` and `aggressive` are optimal in terms of competitive ratio, there exist realistic situations in which the latter may be preferable to the former. Consider, for example, a search-and-rescue mission for a missing backpacker who has disappeared in one of two (very long) concurrent, hiking paths. Assuming that we select our search strategy from the space of 9-competitive strategies, it makes sense to choose one that is tuned to discovering new territory, rather than a conservative strategy that tends to often revisit already explored areas.

With the above observation in mind, we first need to quantify what constitutes efficiency in exploration. To this end, given a strategy S and $l \in \mathbb{R}^+$, we define $D(S, l)$ as the cost incurred by S the first time the searcher has explored an aggregate length equal to l, combined in both branches. An efficient strategy should be such that $D(S, l)$ is small, for all l. Unfortunately, this criterion by itself is insufficient: Consider a strategy that first searches one branch to a length equal to L, where L is very large. Then $D(S, l)$ is as small as possible for all $l < L$; however, this is hardly a good strategy, since it all but ignores one of the branches (and thus its competitive ratio becomes unbounded as $L \to \infty$).

To remedy this situation, we will instead use the above definition in a way that will allow us a pairwise comparison of strategies, which also considers all possible explored lengths. More formally, we define the following:

▶ **Definition 1.** *Let S_1, S_2 denote two search strategies, we define the* discovery ratio of S_1 against S_2*, denoted by $dr(S_1, S_2)$, as*

$$dr(S_1, S_2) = \sup_{l \in \mathbb{R}^+} \frac{D(S_1, l)}{D(S_2, l)}.$$

Moreover, given a class $\mathcal{S}$ of search strategies, the discovery ratio of S against the class $\mathcal{S}$ *is defined as*

$$dr(S, \mathcal{S}) = \sup_{S' \in \mathcal{S}} dr(S, S').$$

In the case $\mathcal{S}$ is the set Σ of all possible strategies, we simply call $dr(S, \mathcal{S})$ the discovery ratio of S*, and we denote it by $dr(S)$.*

Intuitively, the discovery ratio preserves the worst-case nature of competitive analysis, and at the same time bypasses the need for an "offline optimum" solution. Note that if a strategy S has competitive ratio c then it also has discovery ratio c; this follows easily from the fact that for every hider position H, $c(S, H) \geq D(S, |H|)$. However, the opposite is not necessarily true.

It is worth pointing out that one could have defined the discovery ratio over a discrete, countable space (i.e., the target hides at some integer distance from the origin), which turns out to be identical to the *bijective ratio*. This performance measure was introduced in [5] as an extension of (exact) bijective analysis of online algorithms [4], and which in turn is based on the pairwise comparison of the costs induced by two online algorithms over all request sequences of a certain size. Bijective analysis has been applied in fundamental online problems (with a discrete, finite set of requests) such as paging and list update [6], k-server [14, 5], and online search[1] [15].

In what concerns linear search, in this work we choose to present the analysis over a "continuous" space of requests for two reasons. First, we demonstrate that this is indeed possible, which can be useful for other online problems which are defined over a continuous setting of requests (e.g., k-server problems defined over a metric space rather than over a finite graph). Second, the discretization introduces certain unnecessary and undesirable technical issues, e.g., in the choice of the "right" t for strategy R_t (see Lemma 11). While the analysis is still tractable for our problem, for more complex search domains such as star search, the discrete analysis may be too complicated to yield results. We further discuss the connections between the discovery and the bijective ratios in Section 4.

The above observation implies that the discovery ratio inherits the appealing properties of bijective analysis, which further motivate its choice. In particular, note that bijective analysis has helped to identify theoretically efficient algorithms which also tend to perform well in practice (such as Least-Recently-Used for paging [6], and greedy-like k-server policies for certain types of metrics [5]). Furthermore, if an algorithm has bijective ratio c, then its average cost, assuming a uniform distribution over all request sequences of the same length, is within a factor c of the average cost of any other algorithm. Thus, bijective analysis can be used to establish "best of both worlds" types of performance comparisons. In fact, assuming again uniform distributions, much stronger conclusions can be obtained, in that bijective analysis implies a *stochastic dominance* relation between the costs of the two algorithms [5]. However, since the search domain is infinite, one must be careful in defining a uniform

[1] In [15], online search refers to the problem of selling a specific item at the highest possible price, and is not related to the problem of searching for a target.

distribution of requests. More specifically, one could fix $L \geq 1$ and consider the uniform density function on the space $[-L, -1] \cup [1, L]$ (where the origin is assumed to be at 0). Thus, the probability that a request is at distance at most x from the origin is $(x-1)/(L-1)$. Our results then correspond to the setting in which L is unknown to the algorithm, and thus can be arbitrarily large. For known, and thus bounded L, the situation is much more complicated, since the optimal competitive ratio now depends on L and does not have a closed formula [13]. Our overall techniques still apply but the results unavoidably will be much more technical, and probably not tight.

It should be noted that the central question we study in this work is related to a phenomenon that is not unusual in the realm of online computation. Namely, for certain online problems, competitive analysis results in very coarse performance classification of algorithms. This is due to the pessimistic, worst-case nature of the competitive ratio. The definitive example of an online problem in which this undesired situation occurs is the (standard) paging problem in a virtual memory system, which motivated the introduction of several analysis techniques alternative to the competitive ratio (see [19] for a survey). In our paper we demonstrate that a similar situation arises in the context of online search, and we propose a remedy by means of the discovery ratio. We emphasize, however, that in our main results, we apply the discovery ratio as supplementary to the competitive ratio, instead of using it antagonistically as a measure that replaces the competitive ratio altogether.

Contribution

We begin, in Section 2, by identifying the optimal tradeoff between the competitive ratio of a strategy and its discovery ratio (against all possible strategies). The result implies that there are strategies of discovery ratio $2 + \epsilon$, for arbitrarily small $\epsilon > 0$, which is tight. As corollary, we obtain that strategy `doubling` has discovery ratio equal to 3. These results allow us to set up the framework and provide some intuition for our main results, but also demonstrate that the discovery ratio, on itself, does not lead to a useful classification of strategies, when one considers the entire space of strategies.

Our main technical results are obtained in Section 3. Here, we apply synthetically both the competitive and the discovery ratios. More precisely, we restrict our interest to the set of competitively optimal strategies, which we further analyze using the discovery ratio as a supplementary measure. We prove that the strategy `aggressive`, which explores the branches to the furthest possible extent while satisfying the competitiveness constraint, has discovery ratio $\frac{8}{5}$; moreover, we show that this is the optimal discovery ratio in this setting. In contrast, we show that the strategy `doubling` has discovery ratio $\frac{7}{3}$. In addition, we provide evidence that such "aggressiveness" is requisite. More precisely, we show that any competitively optimal strategy that is also optimal with respect to the discovery ratio must have the exact same behavior as the aggressive strategy in the first five iterations.

In terms of techniques, the main technical difficulty in establishing the discovery ratios stems from answering the following question: given a length $l \in \mathbb{R}^+$, what is the strategy S that minimizes $D(S, l)$, and how can one express this minimum discovery cost? This is a type of *inverse* or *dual* problem that can be of independent interest in the context of search problems, in the spirit of a concept such as the *reach* of a strategy [23], also called *extent* in [24] (and which is very useful in the competitive analysis of search strategies). We model this problem as a linear program for whose objective value we first give a lower bound; then we show this bound is tight by providing an explicit 9-competitive strategy which minimizes $D(S, l)$.

Related work

The linear search problem was first introduced and studied in works by Bellman [10] and Beck [8]. The generalization of linear search to m concurrent, semi-infinite branches is known as *star search* or *ray search*; thus linear search is equivalent to star search for $m = 2$. Optimal strategies for linear search under the (deterministic) competitive ratio were first given by [9]. Moreover [21] gave optimal strategies for the generalized problem of star search, a result that was rediscovered later [7]. Some of the related work includes the study of randomization [26]; multi-searcher strategies [29]; multi-target searching [27, 30]; searching with turn cost [18, 3]; searching with an upper bound on the target distance [23, 13]; fault-tolerant search [17]; and the variant in which some probabilistic information on target placement is known [24, 25]. This list is not exclusive; see also Chapter 8 in the book [1].

Linear search and its generalization can model settings in which we seek an intelligent allocation of resources to tasks under uncertainty. For this reason, the problem and its solution often arises in the context of diverse fields such as AI (e.g., in the design of *interruptible algorithms* [12, 2]) and databases (e.g., *pipeline filter ordering* [16]).

Strategy `aggressive` has been studied in [23, 24] in the special case of maximizing the *reach* of a strategy (which informally is the maximum possible extent to which the branches can be searched without violating competitiveness) when we do not know the distance of the target from the origin. Although this gives some intuition that `aggressive` is indeed a good strategy, to the best of our knowledge, our work is the first that quantifies this intuition, in terms of comparing to other competitively optimal strategies using a well-defined performance measure.

Due to space limitations, some proofs are omitted or only sketched.

Preliminaries

In the context of linear search, the searcher's strategy can be described as an (infinite) sequence of lengths at which the two branches (numbered 0,1, respectively) are searched. Formally, a search strategy is determined by an infinite sequence of *search segments* $\{x_0, x_1, \ldots\}$ such that $x_i > 0$ and $x_{i+2} > x_i$ for all $i \in \mathbb{N}$, in the sense that in iteration i, the searcher starts from the origin, searches branch i mod 2 to distance x_i from the origin, and then returns back to O. We require that the search segments induce a complete exploration of both branches of the line, in that for every $d \in \mathbb{R}^+$, there exist $i, j \in \mathbb{N}$ such that $x_{2i} \geq d$, and $x_{2j+1} \geq d$.

The constraint $x_{i+2} > x_i$ implies that the searcher explores a new portion of the line in each iteration. It is easy to see that any other strategy X that does not conform to the above (namely, a strategy such that iterations i, $i+1$ search the same branch, or a strategy in which $x_{i+2} \leq x_i$ can be transformed to a conforming strategy X' such that for any hider H, $c(X', S) \leq c(X, H)$). For convenience of notation, we will define x_i to be equal to 0, for all $i < 0$. Given a strategy X, we define $T_n(X)$ (or simply T_n, when X is clear from context) to be equal to $\sum_{i=0}^{n} x_i$. For $n < 0$, we define $T_n := 0$.

We say that the searcher *turns* in iteration i at the moment it switches directions during iteration i, namely when it completes the exploration of length x_i and returns back to the origin. Moreover, at any given point in time t (assuming a searcher of unit speed), the number of turns incurred by time t is defined accordingly.

We will denote by Σ the set of all search strategies, and by Σ_c the subset of Σ that consists of strategies with competitive ratio c. Thus Σ_9 is the set of competitively optimal strategies, and $\Sigma_\infty \equiv \Sigma$. When evaluating the competitive ratio, we will make the standard assumption that the target must be at distance at least 1 from O, since no strategy can have bounded competitive ratio if this distance can be arbitrarily small.

2 Strategies of optimal discovery ratio in Σ

We begin, by establishing the optimal tradeoff between the competitive ratio and the discovery ratio against all possible strategies. This will allow us to obtain strategies of optimal discovery ratio, and also setup some properties of the measure that will be useful in Section 3.

Let X, Y, denote two strategies in Σ, with $X = (x_0, x_1, \ldots)$. From the definition of the discovery ratio we have that

$$\mathrm{dr}(X,Y) = \sup_{i\in\mathbb{N}} \sup_{\delta\in(0,x_i-x_{i-2}]} \frac{D(X, x_{i-1}+x_{i-2}+\delta)}{D(Y, x_{i-1}+x_{i-2}+\delta)}.$$

Note that for $i = 0$, we have

$$\frac{D(X, x_{i-1}+x_{i-2}+\delta)}{D(Y, x_{i-1}+x_{i-2}+\delta)} = \frac{D(X,\delta)}{D(Y,\delta)} \le \frac{\delta}{\delta} = 1.$$

This is because for all $\delta \le x_0$, $D(X,\delta) = \delta$, and for all $\delta > 0$, $D(Y,\delta) \ge \delta$. Therefore,

$$\mathrm{dr}(X,Y) = \sup_{i\in\mathbb{N}^*} \sup_{\delta\in(0,x_i-x_{i-2}]} \frac{D(X, x_{i-1}+x_{i-2}+\delta)}{D(Y, x_{i-1}+x_{i-2}+\delta)}. \tag{2}$$

The following theorem provides an expression of the discovery ratio in terms of the search segments of the strategy.

▶ **Theorem 2.** *Let $X = (x_0, x_1, \ldots)$. Then*

$$dr(X,\Sigma) = \sup_{i\in\mathbb{N}^*} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1}+x_{i-2}}.$$

Proof. Fix $Y \in \Sigma$. From the definition of search segments in X, we have that

$$D(X, x_{i-1}+x_{i-2}+\delta) = 2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta, \quad \text{for } \delta \in (0, x_i - x_{i-2}]. \tag{3}$$

Moreover, for every Y, we have

$$D(Y, x_{i-1}+x_{i-2}+\delta) \ge x_{i-1}+x_{i-2}+\delta. \tag{4}$$

Substituting (3) and (4) in (2) we obtain

$$\mathrm{dr}(X,Y) \le \sup_{i\in\mathbb{N}^*} \sup_{\delta\in(0,x_i-x_{i-2}]} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{x_{i-1}+x_{i-2}+\delta} \le \sup_{i\in\mathbb{N}^*} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1}+x_{i-2}}. \tag{5}$$

For the lower bound, consider a strategy $Y_i = (y_0^i, y_1^i, \ldots)$, for which $y_0^i = x_{i-1} + x_{i-2} + \delta$ (the values of y_j^i for $j \neq 0$ are not significant, as long as Y_i is a valid strategy). Clearly, $D(Y_i, x_{i-1}+x_{i-2}+\delta) = x_{i-1}+x_{i-2}+\delta$. Therefore, (2) implies

$$\mathrm{dr}(X,Y_i) \ge \sup_{\delta\in(0,x_i-x_{i-2}]} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{x_{i-1}+x_{i-2}+\delta} = \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1}+x_{i-2}}. \tag{6}$$

The lower bound on $\mathrm{dr}(X,\Sigma)$ follows from $\mathrm{dr}(X,\Sigma) \ge \sup_{i\in\mathbb{N}^*} \mathrm{dr}(X,Y_i)$. ◀

In particular, note that for $i = 2$, Theorem 2 shows that for any strategy X,

$$\mathrm{dr}(X, \Sigma) \geq \frac{3x_0 + 2x_1}{x_0 + x_1} \geq 2.$$

We will show that there exist strategies with discovery ratio arbitrarily close to 2, thus optimal for Σ. To this end, we will consider the geometric search strategy defined as $G_\alpha = (1, \alpha, \alpha^2, \ldots)$, with $\alpha > 1$.

▶ **Lemma 3.** *For G_α defined as above, we have $dr(G_\alpha, \Sigma) = \frac{2\alpha^2+\alpha-1}{\alpha^2-1}$.*

In particular, Lemma 3 shows that the discovery ratio of G_α tends to 2, as $\alpha \to \infty$, hence G_α has asymptotically optimal discovery ratio. However, we can show a stronger result, namely that G_α achieves the optimal trade-off between the discovery ratio and the competitive ratio. This is established in the following theorem. Note that the competitive ratio of G_α is easily verified to be $1 + 2\frac{\alpha^2}{\alpha-1}$ (and is minimized for $\alpha = 2$).

▶ **Theorem 4.** *For every strategy $X \in \Sigma$, there exists $\alpha > 1$ such that $dr(X, \Sigma) \geq \frac{2\alpha^2+\alpha-1}{\alpha^2-1}$ and $cr(X) \geq 1 + 2\frac{\alpha^2}{\alpha-1}$.*

In order to prove Theorem 4, we will use of a result by Gal [22] and Schuierer [31] which, informally, lower-bounds the supremum of an infinite sequence of functionals by the supremum of simple functionals of a certain geometric sequence, and which we state here in a simplified form. This result will allow us to lower bound the supremum of a sequence of functionals by the supremum of simple functionals of a geometric sequence. Given an infinite sequence $X = (x_0, x_1, \ldots)$, define $X^{+i} = (x_i, x_{i+1}, \ldots)$ as the suffix of the sequence X starting at x_i.

▶ **Theorem 5** ([22, 31])**.** *Let $X = (x_0, x_1, \ldots)$ be a sequence of positive numbers, r an integer, and $\alpha = \limsup_{n\to\infty}(x_n)^{1/n}$, for $\alpha \in \mathbb{R} \cup \{+\infty\}$. Let F_i, $i \geq 0$ be a sequence of functionals which satisfy the following properties:*

1. *$F_i(X)$ only depends on $x_0, x_1, \ldots, x_{i+r}$,*
2. *$F_i(X)$ is continuous for all $x_k > 0$, with $0 \leq k \leq i + r$,*
3. *$F_i(\lambda X) = F_i(X)$, for all $\lambda > 0$,*
4. *$F_i(X + Y) \leq \max(F_i(X), F_i(Y))$, and*
5. *$F_{i+1}(X) \geq F_i(X^{k+1})$, for all $k \geq 1$,*

then

$$\sup_{0\leq i<\infty} F_i(X) \geq \sup_{0\leq i<\infty} F_i(G_\alpha).$$

Proof of Theorem 4. Let $X = (x_0, x_1, \ldots)$ denote a strategy in Σ. From (6) we know that

$$\mathrm{dr}(X, \Sigma) \geq \sup_i F_i(X),$$

where $F_i(X)$ is defined as the functional $\frac{2\sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1}+x_{i-2}}$. Moreover, the competitive ratio of X can be lower-bounded by

$$\mathrm{cr}(X) \geq \sup_i F_i'(X), \quad \text{where } F_i'(X) = 1 + 2\frac{\sum_{j=0}^{i+1} x_j}{x_i}.$$

This follows easily by considering a hider placed at distance $x_i + \epsilon$, with $\epsilon \to 0$, at the branch that is searched by X in iteration i.

It is easy to see that both $F_i(X)$ and $F'_i(X)$ satisfy the conditions of Theorem 5 (this also follows from Example 7.3 in [1]). Thus, there exists α defined as in the statement of Theorem 5 such that

$$\mathrm{dr}(X,\Sigma) \geq \sup_i F_i(G_\alpha) = \frac{2\sum_{j=0}^{i-1}\alpha^j + \alpha^{i-2}}{\alpha^{i-1}+\alpha^{i-2}}, \text{ and} \tag{7}$$

$$\mathrm{cr}(X,\Sigma) \geq \sup_i F'_i(G_\alpha) = 1 + 2\frac{\sum_{j=0}^{i+1}\alpha^j}{\alpha^i}. \tag{8}$$

It is easy to verify that if $\alpha = 1$, then $\mathrm{dr}(X,\Sigma), \mathrm{cr}(X,\Sigma) = \infty$. We can thus assume that $\alpha > 1$, and thus obtain from (7), (8), after some manipulations, that

$$\mathrm{dr}(X,\Sigma) \geq \sup_i \frac{2(\alpha^2 - \frac{1}{\alpha^{i-2}}) + \alpha - 1}{\alpha^2 - 1} = \frac{2\alpha^2+\alpha-1}{\alpha^2-1}, \text{ and}$$

$$\mathrm{cr}(X,\Sigma) \geq 1 + \sup_i \frac{2\sum_{j=0}^{i+1}\alpha^j}{\alpha^i} = \sup_i 1 + 2\frac{\alpha^2 - \frac{1}{\alpha^i}}{\alpha-1} = 1 + 2\frac{\alpha^2}{\alpha-1},$$

which concludes the proof. ◀

Note, however, that although G_α, with $\alpha \to \infty$ has optimal discovery ratio, its competitive ratio is unbounded. Furthermore, strategy `doubling` $\equiv G_2$ has optimal competitive ratio equal to 9, whereas its discovery ratio is equal to 3. This motivates the topic of the next section.

3 The discovery ratio of competitively optimal strategies

In this section we focus on strategies in Σ_9, namely the set of competitively optimal strategies. For any strategy $X \in \Sigma_9$, it is known that there is an infinite set of linear inequalities that relate its search segments, as shown in the following lemma (see, e.g, [24]).

▶ **Lemma 6.** *The strategy $X = (x_0, x_1, x_2, \ldots)$ is in Σ_9 if and only if its segments satisfy the following inequalities*

$$1 \leq x_0 \leq 4, \quad x_1 \geq 1 \quad \textit{and} \quad x_n \leq 3x_{n-1} - \sum_{i=0}^{n-2} x_i, \quad \textit{for all } n \geq 1.$$

We now define a class of strategies in Σ_9 as follows. For given $t \in [1,4]$, let R_t denote the strategy whose search segments are determined by the linear recurrence

$$x_0 = t, \quad \text{and} \quad x_n = 3x_{n-1} - \sum_{i=0}^{n-2} x_i, \quad \text{for all } n \geq 1.$$

In words, R_t is such that for every $n > 1$, the inequality relating $x_0, \ldots, x_n$ is tight. The following lemma determines the search lengths of R_t as function of t. The lemma also implies that R_t is indeed a valid search strategy, for all $t \in [1,4]$, in that $x_n > x_{n-2}$, for all n, and $x_n \to \infty$, as $n \to \infty$.

▶ **Lemma 7.** *The strategy R_t is defined by the sequence $x_n = t(1+\frac{n}{2})2^n$, for $n \geq 0$. Moreover, $T_n(R_t) = t(n+1)2^n$.*

Proof. The lemma is clearly true for $n \in \{0,1\}$. For $n \geq 2$, the equality $x_n = 3x_{n-1} - \sum_{i=0}^{n-2} x_i$ implies that $T_n = \sum_{i=0}^{n} x_i = 4x_{n-1}$. Therefore,

$$T_n - T_{n-1} = 4x_{n-1} - 4x_{n-2} \Rightarrow x_n = 4(x_{n-1} - x_{n-2}).$$

The characteristic polynomial of the above linear recurrence is $\xi^2 - 4\xi + 4$, with the unique root $\xi = 2$. Thus, x_n is of the form $x_n = (a + bn)2^n$, for $n \geq 0$, where a and b are determined by the initial conditions $x_0 = t$ and $x_1 = 3t$. Summarizing, we obtain that for $n \geq 0$ we have that $x_n = t(1 + \frac{n}{2})2^n$, and $T_n = 4x_{n-1} = t(n+1)2^n$. ◀

Among all strategies in R_t we are interested, in particular, in the strategy R_4. This strategy has some intuitively appealing properties: It maximizes the search segments in each iteration (see Lemma 9) and minimizes the number of turns required to discover a certain length (as will be shown in Corollary 10). Using the notation of the introduction, we can say that $R_4 \equiv$ `aggressive`. In this section we will show that `aggressive` has optimal discovery ratio among all competitively optimal strategies. Let us denote by $\bar{x}_i$ the search segment in the i-th iteration in `aggressive`.

▶ **Corollary 8.** *The strategy* `aggressive` *can be described by the sequence* $\bar{x}_n = (n+2)2^{n+1}$, *for* $n \geq 0$. *Moreover,* $T_n($`aggressive`$) = (n+1)2^{n+2}$, *for* $n \geq 0$.

The following lemma shows that, for any given n, the total length discovered by any competitively optimal strategy X at the turning point of the n-th iteration cannot exceed the corresponding length of `aggressive`. Its proof can also be found in [24], but we give a different proof using ideas that we will apply later (Lemma 11).

▶ **Lemma 9.** *For every strategy* $X = (x_0, x_1, \ldots)$ *with* $X \in \Sigma_9$, *it holds that* $x_n \leq \bar{x}_n$, *for all* $n \in \mathbb{N}$, *where* $\bar{x}_n$ *is the search segment in the* n*-th iteration of* `aggressive`. *Hence, in particular, we have* $x_n + x_{n-1} \leq \bar{x}_n + \bar{x}_{n-1}$, *for all* $n \in \mathbb{N}$.

Proof. For a given $n \geq 0$, let P_n denote the following linear program.

$$\begin{aligned} \max \quad & x_n \\ \text{subject to} \quad & 1 \leq x_0 \leq 4, \\ & x_1 \geq 1, \\ & x_i \leq 3x_{i-1} - \sum_{j=0}^{i-2} x_j, \quad 1 \leq i \leq n. \end{aligned}$$

We will show, by induction on i, that for all $i \leq n$,

$$x_n \leq (i+2)2^{i-1}x_{n-i} - i2^{i-1}T_{n-i-1}(X).$$

The lemma will then follow, since for $i = n$ we have

$$x_n \leq (n+2)2^{n-1}x_0 \leq (n+2)2^{n-1} \cdot 4 = (n+2)2^{n+1} = \bar{x}_n,$$

where the last equality is due to Corollary 8. We will now prove the claim. Note that, the base case, namely $i = 1$, follows directly from the LP constraint. For the induction hypothesis, suppose that for $i \geq 1$, it holds that

$$x_n \leq (i+2)2^{i-1}x_{n-i} - i2^{i-1}T_{n-i-1}(X). \tag{9}$$

We will show that the claim holds for $i+1$. Since

$$x_{n-i} \leq 3x_{n-i-1} - T_{n-i-2}(X), \tag{10}$$

then

$$\begin{aligned} x_n &\leq (i+2)2^{i-1}(3x_{n-i-1} - T_{n-i-2}(X)) - i2^{i-1}T_{n-i-1}(X) && \text{(subst. (10) in (9))} \\ &= (i+2)2^{i-1}(3x_{n-i-1} - T_{n-i-2}(X)) - i2^{i-1}(T_{n-i-2}(X) + x_{n-i-1}) && \text{(def. } T_{n-i-1}\text{)} \\ &= (i+3)2^i x_{n-i-1} + (i+1)2^i T_{n-i-2}(X), && \text{(arranging terms)} \end{aligned}$$

which completes the proof of the claim. ◀

Given strategy X and $l \in \mathbb{R}^+$, define $m(X,l)$ as the number of turns that X has performed by the time it discovers a total length equal to l. Also define

$$m^*(l) = \inf_{X \in \Sigma_9} m(X,l),$$

that is, $m^*(l)$ is the minimum number of turns that a competitively optimal strategy is required to perform in order to discover length equal to l. From the constraint $x_0 \leq 4$, it follows that clearly $m^*(l) = 0$, for $l \leq 4$. The following corollary to Lemma 9 gives an expression for $m^*(l)$, for general values of l.

▶ **Corollary 10.** *For given $l > 4$, $m^*(l) = m(\texttt{aggressive}, l) = \min\{n \in \mathbb{N}_{\geq 1} : (3n+5)2^n \geq l\}$.*

Proof. From Lemma 9, the total length discovered by any $X \in \Sigma_9$ at the turning point of the n-th iteration cannot exceed $\bar{x}_n + \bar{x}_{n-1}$ for $n \geq 1$, which implies that $m^*(l) = n$, if $l \in (\bar{x}_{n-1} + \bar{x}_{n-2}, \bar{x}_n + \bar{x}_{n-1}]$ for $n \geq 1$. In other words,

$$m^*(l) = \min\{n \in \mathbb{N}_{\geq 1} : \bar{x}_n + \bar{x}_{n-1} \geq l\}.$$

From Corollary 8, we have $\bar{x}_n = (n+2)2^{n+1}$, for $n \geq 0$. Hence,

$$m^*(l) = \min\{n \in \mathbb{N}_{\geq 1} : (3n+5)2^n \geq l\}.$$ ◀

The following lemma is a central technical result that is instrumental in establishing the bounds on the discovery ratio. For a given $l \in \mathbb{R}^+$, define

$$d^*(l) = \inf_{X \in \Sigma_9} D(X,l).$$

In words, $d^*(l)$ is the minimum cost at which a competitively optimal strategy can discover a length equal to l. Trivially, $d^*(l) = l$ if $l \leq 4$. Lemma 11 gives an expression of $d^*(l)$ for $l > 4$ in terms of $m^*(l)$; it also shows that there exists a $t \in (1,4]$ such that the strategy R_t attains this minimum cost.

We first give some motivation behind the purpose of the lemma. When considering general strategies in Σ, we used a lower bound on the cost for discovering a length l as given by (4), and which corresponds to a strategy that never turns. However, this lower bound is very weak when one considers strategies in Σ_9. This is because a competitive strategy needs to turn sufficiently often, which affects considerably the discovery costs.

We also give some intuition about the proof. We show how to model the question by means of a linear program. Using the constraints of the LP, we first obtain a lower bound on its objective in terms of the parameters l and $m^*(l)$. In this process, we also obtain a lower bound on the first segment of the strategy (x_0); this is denoted by t in the proof. In the next step, we show that the strategy R_t has discovery cost that matches the lower bound on the objective, which suffices to prove the result.

▶ **Lemma 11.** *For $l > 4$, it holds*

$$d^*(l) = D(R_t, l) = l \cdot \frac{6m^*(l)+4}{3m^*(l)+5}, \qquad \text{where } t = l \cdot \frac{2^{2-m^*(l)}}{3m^*(l)+5} \in (1,4].$$

Proof. Let $X = (x_0, x_1, \ldots) \in \Sigma_9$ denote the strategy which minimizes the quantity $D(X, l)$. Then there must exist a smallest $n \geq m^*(l)$ such that the searcher discovers a total length l during the n-th iteration. More precisely, suppose that this happens when the searcher is at branch $n \bmod 2$, and at some position p (i.e., distance from O), with $p \in (x_{n-2}, x_n]$. Then we have $x_{n-1} + p = l$, and

$$d^*(l) = D(X, l) = 2\sum_{i=0}^{n-1} x_i + p = 2\sum_{i=0}^{n-1} x_i + (l - x_{n-1}) = 2\sum_{i=0}^{n-2} x_i + x_{n-1} + l.$$

Therefore, $d^*(l)$ is the objective of the following linear program.

$$\begin{aligned}
\min \quad & 2\sum_{i=0}^{n-2} x_i + x_{n-1} + l \\
\text{subject to} \quad & x_n + x_{n-1} \geq l, \\
& 1 \leq x_0 \leq 4, \\
& x_{i-2} \leq x_i, && i \in [2, n] \\
& 1 \leq x_i \leq 3x_{i-1} - \sum_{j=0}^{i-2} x_j, && i \in [1, n].
\end{aligned}$$

Recall that $n \geq m^*(l)$ is a fixed integer. Let Obj denote the objective value of the linear program. We claim that, for $1 \leq i \leq n$,

$$x_{n-i} \geq \frac{2^{2-i}}{3i+5} l + \frac{3i-1}{3i+5} T_{n-i-1} \quad \text{and} \quad \text{Obj} \geq \frac{6i+4}{3i+5} l + \frac{9 \cdot 2^i}{3i+5} T_{n-i-1}.$$

The claim provides a lower bound of the objective, since for $i = n$ it implies that

$$x_0 \geq \frac{2^{2-n}}{3n+5} l \quad \text{and} \quad \text{Obj} \geq \frac{6n+4}{3n+5} l \geq \frac{6m^*(l)+4}{3m^*(l)+5} l,$$

where the last inequality follows from the fact $n \geq m^*(l)$. We will argue later that this lower bound is tight.

First, we prove the claim, by induction on i, for all $i \leq n$. We first show the base case, namely $i = 1$. Since $x_n \leq 3x_{n-1} - T_{n-2}$ and $x_n + x_{n-1} \geq l$, it follows that

$$x_{n-1} \geq l - x_n \geq l - (3x_{n-1} - T_{n-2}) \Rightarrow x_{n-1} \geq \frac{l}{4} + \frac{T_{n-2}}{4}, \quad \text{hence}$$

$$\text{Obj} = l + 2T_{n-2} + x_{n-1} \geq l + 2T_{n-2} + \frac{l}{4} + \frac{T_{n-2}}{4} = \frac{5}{4} l + \frac{9}{4} T_{n-2},$$

thus the base case holds. For the induction step, suppose that

$$x_{n-i} \geq \frac{2^{2-i}}{3i+5} l + \frac{3i-1}{3i+5} T_{n-i-1} \quad \text{and} \quad \text{Obj} \geq \frac{6i+4}{3i+5} l + \frac{9 \cdot 2^i}{3i+5} T_{n-i-1}.$$

Then,

$$\begin{aligned}
3x_{n-i-1} - T_{n-i-2} &\geq x_{n-i} && \text{(by LP constraint)}\\
&\geq \frac{2^{2-i}}{3i+5}l + \frac{3i-1}{3i+5}T_{n-i-1} && \text{(ind. hyp.)}\\
&= \frac{2^{2-i}}{3i+5}l + \frac{3i-1}{3i+5}(T_{n-i-2} + x_{n-i-1}) && \text{(def. } T_{n-i-1})
\end{aligned}$$

By rearranging terms in the above inequality we obtain

$$\begin{aligned}
&(3 - \frac{3i-1}{3i+5})x_{n-i-1} \geq \frac{2^{2-i}}{3i+5}l + (1 + \frac{3i-1}{3i+5})T_{n-i-2} \Rightarrow\\
&\frac{6i+16}{3i+5}x_{n-i-1} \geq \frac{2^{2-i}}{3i+5}l + \frac{6i+4}{3i+5}T_{n-i-2} \Rightarrow x_{n-i-1} \geq \frac{2^{1-i}}{3i+8}l + \frac{3i+2}{3i+8}T_{n-i-2},
\end{aligned}$$

and

$$\begin{aligned}
\text{Obj} &\geq \frac{6i+4}{3i+5}l + \frac{9\cdot 2^i}{3i+5}T_{n-i-1} && \text{(ind. hyp.)}\\
&= \frac{6i+4}{3i+5}l + \frac{9\cdot 2^i}{3i+5}(T_{n-i-2} + x_{n-i-1}) && \text{(def. } T_{n-i-1})\\
&\geq \frac{6i+4}{3i+5}l + \frac{9\cdot 2^i}{3i+5}T_{n-i-2} + \frac{9\cdot 2^i}{3i+5}(\frac{2^{1-i}}{3i+8}l + \frac{3i+2}{3i+8}T_{n-i-2}) && \text{(ind. hyp.)}\\
&= \frac{6i+10}{3i+8}l + \frac{9\cdot 2^{i+1}}{3i+8}T_{n-i-2}.
\end{aligned}$$

This concludes the proof of the claim, which settles the lower bound on $d^*(l)$. It remains to show that this bound is tight. Consider the strategy R_t, with $t = \frac{2^{2-m^*(l)}}{3m^*(l)+5}l$. In what follows we will show that R_t is a feasible solution of the LP, and that $D(R_t, l) = \frac{6m^*(l)+4}{3m^*(l)+5}l$.

First, we show that $t \in (1, 4]$. For the upper bound, from Corollary 10, we have $(3m^*(l)+5)2^{m^*(l)} \geq l$, which implies that

$$1 \geq l \cdot \frac{2^{-m^*(l)}}{3m^*(l)+5} \Rightarrow 4 \geq l \cdot \frac{2^{2-m^*(l)}}{3m^*(l)+5} \Rightarrow 4 \geq t.$$

In order to show that $t > 1$, consider first the case $l \in (4, 5]$. Then $m^*(l) = 1$, which implies that

$$t = \frac{2^{2-m^*(l)}}{3m^*(l)+5}l = \frac{l}{4} \geq 1.$$

Moreover, if $l > 5$, by Corollary 10, $m^*(l)$ is the smallest integer solution of the inequality $(3n+5)2^n \geq l$, then $(3m^*(l)+2)2^{m^*(l)-1} < l$, hence

$$\begin{aligned}
t &= \frac{2^{2-m^*(l)}}{3m^*(l)+5}l = \frac{4l}{(3m^*(l)+5)2^{m^*(l)}} = \frac{2l}{(3m^*(l)+2)2^{m^*(l)-1}\cdot\frac{3m^*(l)+5}{3m^*(l)+2}}\\
&> \frac{2l}{l\cdot\frac{3m^*(l)+5}{3m^*(l)+2}} = \frac{6m^*(l)+4}{3m^*(l)+5} > 1.
\end{aligned}$$

The last inequality holds since we have $m^*(l) \geq 1$, for $l > 5$. This concludes that $t \in (1, 4]$, and R_t is a feasible solution of the LP since R_t satisfies all other constraints by its definition.

It remains thus to show that $D(R_t, l) = \frac{6m^*(l)+4}{3m^*(l)+5} l$. By Lemma 7, we have

$$\begin{aligned} x_{m^*(l)} + x_{m^*(l)-1} &= t\left(1 + \frac{m^*(l)}{2}\right) 2^{m^*(l)} + t\left(1 + \frac{m^*(l)-1}{2}\right) 2^{m^*(l)-1} \\ &= t \cdot 2^{m^*(l)} \cdot \frac{3m^*(l)+5}{4} = \frac{2^{2-m^*(l)}}{3m^*(l)+5} l \cdot 2^{m^*(l)} \cdot \frac{3m^*(l)+5}{4} = l. \end{aligned}$$

Then R_t has exactly discovered a total length l right before the $m^*(l)$-th turn. Hence,

$$\begin{aligned} D(R_t, l) &= 2T_{m^*(l)-2} + x_{m^*(l)-1} + l \\ &= t \cdot (m^*(l)-1)\, 2^{m^*(l)-1} + t \cdot \left(1 + \frac{m^*(l)-1}{2}\right) 2^{m^*(l)-1} + l && \text{(by Lemma 7)} \\ &= t \cdot \frac{(3m^*(l)-1)2^{m^*(l)}}{4} + l && \text{(arranging terms)} \\ &= \frac{2^{2-m^*(l)}}{3m^*(l)+5} l \cdot \frac{(3m^*(l)-1)2^{m^*(l)}}{4} + l && \text{(substituting } t\text{)} \\ &= \left(\frac{3m^*(l)-1}{3m^*(l)+5} + 1\right) \cdot l = \frac{6m^*(l)+4}{3m^*(l)+5} \cdot l. && \text{(arranging terms)} \end{aligned}$$

This concludes the proof of the lemma. ◀

We are now ready to prove the main results of this section. Recall that for any two strategies X, Y, $\mathrm{dr}(X, Y)$ is given by (2). Combining with (3), as well as with the fact that for $Y \in \Sigma_9$, we have that $D(Y, l) \geq d^*(l)$, (from the definition of d^*), we obtain that

$$\mathrm{dr}(X, \Sigma_9) = \sup_{i \in \mathbb{N}^*} \sup_{\delta \in (0, x_i - x_{i-2}]} F_i(X, \delta), \quad \text{where } F_i(X, \delta) = \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{d^*(x_{i-1} + x_{i-2} + \delta)}. \tag{11}$$

Recall that for the strategy $\texttt{aggressive} \equiv R_4 = (\bar{x}_0, \bar{x}_1, \ldots)$, its segments $\bar{x}_i$ are given in Corollary 8.

▶ **Theorem 12.** *For the strategy* `aggressive` *it holds that* $dr(\texttt{aggressive}, \Sigma_9) = 8/5$.

Proof. We will express the discovery ratio using (11). For $i = 1$, and $\delta \in (0, \bar{x}_1]$, we have that

$$F_1(\texttt{aggressive}, \delta) = \frac{2\bar{x}_0 + \delta}{d^*(\bar{x}_0 + \delta)} = \frac{8 + \delta}{d^*(4 + \delta)}.$$

From Lemma 11, $d^*(4+\delta) = (4+\delta) \cdot \frac{6 \cdot 1 + 4}{3 \cdot 1 + 5} = \frac{5(4+\delta)}{4}$; this is because $1 \leq m^*(4+\delta) \leq m^*(16) = 1$. Then,

$$F_1(\texttt{aggressive}, \delta) = \frac{8+\delta}{\frac{5(4+\delta)}{4}} = \frac{32 + 4\delta}{20 + 5\delta}, \text{ hence } \sup_{\delta \in (0, \bar{x}_1]} F_1(\texttt{aggressive}, \delta) = \frac{8}{5}. \tag{12}$$

For given $i \geq 2$, and $\delta \in (0, \bar{x}_i - \bar{x}_{i-2}]$, we have

$$F_i(\texttt{aggressive}, \delta) = \frac{2T_{i-1} + \bar{x}_{i-2} + \delta}{d^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \delta)},$$

where T_{i-1} is given by Corollary 8. Moreover, from Lemma 11 we have that

$$d^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) = (\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) \cdot \frac{6m^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) + 4}{3m^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) + 5} = (\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) \cdot \frac{6i+4}{3i+5},$$

where the last equality follows from the fact that $m^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) = i$. This is because

$$i \le m^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \delta) \le m^*(\bar{x}_{i-1} + \bar{x}_{i-2} + \bar{x}_i - \bar{x}_{i-2}) = m^*(\bar{x}_i + \bar{x}_{i-1}) = i.$$

Substituting with the values of the search segments as well as T_{i-1}, we obtain that

$$F_i(\texttt{aggressive}, \delta) = \frac{i \cdot 2^{i+2} + i \cdot 2^{i-1} + \delta}{((i+1)2^i + i \cdot 2^{i-1} + \delta) \cdot \frac{6i+4}{3i+5}} = \frac{9i \cdot 2^{i-1} + \delta}{((3i+2)2^{i-1} + \delta) \cdot \frac{6i+4}{3i+5}}.$$

Since

$$\frac{\partial F_i(\texttt{aggressive}, \delta)}{\partial \delta} = -\frac{2^{i+1}(3i-1)(3i+5)}{(3i+2)(2^n(3i+2) + 2\delta)^2} \le 0,$$

then $F_i(\texttt{aggressive}, \delta)$ is monotone decreasing in δ. Thus

$$\sup_{\delta \in (0, \bar{x}_i - \bar{x}_{i-2}]} F_i(\texttt{aggressive}, \delta) = \frac{9i \cdot 2^{i-1}}{((3i+2)2^{i-1}) \cdot \frac{6i+4}{3i+5}} = \frac{9i(3i+5)}{(3i+2)(6i+4)},$$

and then

$$\sup_{i \in \mathbb{N}_{i \ge 2}} \sup_{\delta \in (0, \bar{x}_i - \bar{x}_{i-2}]} F_i(\texttt{aggressive}, \delta) = \frac{(9 \cdot 2)(3 \cdot 2 + 5)}{(3 \cdot 2 + 2)(6 \cdot 2 + 4)} = \frac{99}{64} < \frac{8}{5}. \tag{13}$$

Combining (11), (12) and (13) yields the proof of the theorem. ◀

The following theorem shows that `aggressive` has optimal discovery ratio among all competitively optimal strategies.

▶ **Theorem 13.** *For every strategy $X \in \Sigma_9$, we have $dr(X, \Sigma_9) \ge \frac{8}{5}$.*

Proof. Let $X = (x_0, \ldots)$. We will consider two cases, depending on whether $x_0 < 4$ or $x_0 = 4$. Suppose, first, that $x_0 < 4$. In this case, for sufficiently small ϵ, we have $m^*(x_0 + \epsilon) = 0$, which implies that $d^*(x_0 + \epsilon) = x_0 + \epsilon$, and therefore.

$$F_1(X, \epsilon) = \frac{2x_0 + \epsilon}{d^*(x_0 + \epsilon)} = \frac{2x_0 + \epsilon}{x_0 + \epsilon},$$

from which we obtain that

$$\sup_{\delta \in (0, x_1]} F_1(X, \delta) \ge F_1(X, \epsilon) \ge \frac{2x_0 + \epsilon}{x_0 + \epsilon} \to 2, \text{ as } \epsilon \to 0^+.$$

Next, suppose that $x_0 = 4$. In this case, for $\delta \in (0, x_1]$, it readily follows that $F_1(X, \delta) = F_1(\texttt{aggressive}, \delta)$. Therefore, from (12), we have that

$$\sup_{\delta \in (0, x_1]} F_1(X, \delta) = \sup_{\delta \in (0, x_1]} \frac{32 + 4\delta}{20 + 5\delta} = \frac{8}{5}.$$

The lower bound follows directly from (11). ◀

Recall that `doubling` $\equiv G_2 = (2^0, 2^1, 2^2, \ldots)$. The following theorem shows that within Σ_9, `doubling` has worse discovery ratio than `aggressive`. The proof follows along the lines of the proof of Theorem 12, where instead of using the search segments $\bar{x}_i$ of `aggressive`, we use the search segment $x_i = 2^i$ of `doubling`.

▶ **Theorem 14.** *We have $dr(\texttt{doubling}, \Sigma_9) = \frac{7}{3}$.*

A natural question arises: Is `aggressive` the unique strategy of optimal discovery ratio in Σ_9? The following theorem provides evidence that optimal strategies cannot be radically different than `aggressive`, in that they must mimic it in the first few iterations.

▶ **Theorem 15.** *Strategy $X = (x_0, x_1, \ldots) \in \Sigma_9$, has optimal discovery ratio in Σ_9 only if $x_i = \bar{x}_i$, for $0 \leq i \leq 4$.*

Proof. Consider a strategy $X(x_0, x_1, \ldots) \in \Sigma_9$. Recall that the discovery ratio of X is given by Equation (11). We will prove the theorem by induction on i.

We first show the base case, namely $i = 0$. The base case holds by the argument used in the proof of Theorem 13 which shows that if $x_0 < 4$, then $\mathrm{dr}(X, \Sigma_9) \geq 2$. For the induction step, suppose that, if X has optimal discovery ratio then for $j \in [0, i]$, $x_j = \bar{x}_j$, with $i < 4$. We will show $x_{i+1} = \bar{x}_{i+1}$ by contradiction, hence assume $x_{i+1} < \bar{x}_{i+1}$. For sufficiently small $\epsilon > 0$, we have

$$\begin{aligned} m^*(x_{i+1} + x_i + \epsilon) &= m^*(x_{i+1} + \bar{x}_i + \epsilon) && \text{(by induction hypothesis)} \\ &\leq m^*(\bar{x}_{i+1} + \bar{x}_i) && \text{(by monotonicity of } m^* \text{ and Lemma 9)} \\ &= i + 1, && \text{(by definition of } m^*) \end{aligned}$$

which implies that, by Lemma 11,

$$d^*(x_i + x_{i-1} + \epsilon) = (x_i + x_{i-1} + \epsilon) \cdot \frac{6 \cdot m^*(x_{i+1} + x_i + \epsilon) + 4}{3 \cdot m^*(x_{i+1} + x_i + \epsilon) + 5} \leq (x_i + x_{i-1} + \epsilon) \cdot \frac{6 \cdot (i+1) + 4}{3 \cdot (i+1) + 5}. \tag{14}$$

Therefore

$$\begin{aligned} F_{i+2}(X, \epsilon) &= \frac{2 \cdot \sum_{j=0}^{i+1} x_j + x_i + \epsilon}{d^*(x_{i+1} + x_i + \epsilon)} \\ &= \frac{2T_i(\texttt{aggressive}) + 2x_{i+1} + \bar{x}_i + \epsilon}{d^*(x_{i+1} + \bar{x}_i + \epsilon)} && \text{(by ind. hyp.)} \\ &\geq \frac{2T_i(\texttt{aggressive}) + 2x_{i+1} + \bar{x}_i + \epsilon}{(x_{i+1} + \bar{x}_i + \epsilon) \cdot \frac{6 \cdot (i+1) + 4}{3 \cdot (i+1) + 5}} && \text{(Equation (14))} \\ &= \frac{(i+1)2^{i+3} + (i+2)2^{i+1} + 2x_{i+1} + \epsilon}{(x_{i+1} + (i+2)2^{i+1} + \epsilon) \cdot \frac{6 \cdot (i+1) + 4}{3 \cdot (i+1) + 5}} && \text{(Corollary 8)} \\ &\geq \frac{(i+1)2^{i+3} + (i+2)2^{i+1} + (i+3)2^{i+3} + \epsilon}{(i+3)2^{i+2} + (i+2)2^{i+1} + \epsilon} \cdot \frac{3i+8}{6i+10}. && \text{(monoton. on } x_{i+1}) \end{aligned}$$

Hence

$$\sup_{\delta \in (0, x_{i+2} - x_i]} F_{i+2}(X, \delta) \geq \frac{(i+1)2^{i+3} + (i+2)2^{i+1} + (i+3)2^{i+3}}{(i+3)2^{i+2} + (i+2)2^{i+1}} \cdot \frac{3i+8}{6i+10} = \frac{9i+18}{6i+10},$$

which is greater than $\frac{8}{5}$ if $i \leq 3$. We conclude, from (11) that $\mathrm{dr}(X, \Sigma_9) > 8/5$, which is a contradiction. ◀

4 Connections between the discovery and the bijective ratios

In this last section we establish a connection between the discovery and the bijective ratios. Bijective analysis was introduced in [4] in the context of online computation, assuming that each request is drawn from a discrete, finite set. For instance, in the context of the paging

problem, each request belongs to the set of all pages. Let $\mathcal{I}_n$ denote the set of all requests of size n. For a cost-minimization problem Π with discrete, finite requests, let $\pi : \mathcal{I}_n \to \mathcal{I}_n$ denote a bijection over $\mathcal{I}_n$. Given two online algorithms A and B for Π, the bijective ratio of A against B, is defined as

$$\mathrm{br}(A, B) = \min_{\pi:\mathcal{I}_n \to \mathcal{I}_n} \sup_{\sigma \in \mathcal{I}_n} \frac{A(\sigma)}{B(\pi(\sigma))}, \text{ for all } n \geq n_0,$$

where $A(\sigma)$ denotes the cost of A on request sequence σ.

Assuming $\mathcal{I}_n$ is finite, an equivalent definition of $\mathrm{br}(A, B)$ is as follows. Let $A(i, n)$ denote the i-th least costly request sequence for A among request sequences in $\mathcal{I}_n$. Then

$$\mathrm{br}(A, B) = \sup_n \max_i \frac{A(i, n)}{B(i, n)}.$$

Consider in contrast, the linear search problem. Here, there is only one request: the unknown position of the hider (i.e., $n = 1$). However, the set of all requests is not only infinite, but uncountable. Thus, the above definitions do not carry over to our setting, and we need to seek alternative definitions. One possibility is to discretize the set of all requests (as in [5]). Namely, we may assume that the hider can hide only at integral distances from the origin. Then given strategies S_1, S_2, one could define the bijective ratio of S_1 against S_2 as $\sup_i \frac{S_1(i)}{S_2(i)}$, where $S(i)$ denotes the i-th least costly request (hider position) in strategy S.

While the latter definition may indeed be valid, it is still not a faithful representation of the continuous setting. For instance, for hiding positions "close" to the origin, the discretization adds overheads that should not be present, and skews the expressions of the ratios. For this reason, we need to adapt the definition so as to reflect the continuous nature of the problem. Specifically, note that while the concept "the cost of the i-th least costly request in S" is not well-defined in the continuous setting, the related concept of "the cost for discovering a total length equal to l in S" is, in fact, well defined, and is precisely the value $D(S, l)$. We can thus define the bijective ratio of S_1 against S_2 as

$$\mathrm{br}(S_1, S_2) = \sup_l \frac{D(S_1, l)}{D(S_2, l)},$$

which is the same as the definition of the discovery ratio (Definition 1).

References

1 S. Alpern and S. Gal. *The theory of search games and rendezvous*. Kluwer Academic Publishers, 2003.

2 S. Angelopoulos. Further connections between contract-scheduling and ray-searching problems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1516–1522, 2015.

3 S. Angelopoulos, D. Arsénio, and C. Dürr. Infinite linear programming and online searching with turn cost. *Theoretical Computer Science*, 670:11–22, 2017.

4 S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. On the Separation and Equivalence of Paging Strategies. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 229–237, 2007.

5 S. Angelopoulos, M. Renault, and P. Schweitzer. Stochastic dominance and the bijective ratio of online algorithms, 2016. `arXiv:1607.06132`.

6 S. Angelopoulos and P. Schweitzer. Paging and List Update under Bijective Analysis. *Journal of the ACM*, 60(2):7:1–7:18, 2013.

7 R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106:234–244, 1993.
8 A. Beck. On the linear search problem. *Naval Research Logistics*, 2:221–228, 1964.
9 A. Beck and D.J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8(4):419–429, 1970.
10 R. Bellman. An optimal search problem. *SIAM Review*, 5:274, 1963.
11 P. Berman. *Online Algorithms: The State of the Art*, chapter Online searching and navigation, pages 232–241. Springer, 1998.
12 D.S. Bernstein, L. Finkelstein, and S. Zilberstein. Contract algorithms and robots on rays: unifying two scheduling problems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1211–1217, 2003.
13 P. Bose, J. De Carufel, and S. Durocher. Searching on a line: A complete characterization of the optimal solution. *Theoretical Computer Science*, 569:24–42, 2015.
14 J. Boyar, S. Irani, and K. S. Larsen. A Comparison of Performance Measures for Online Algorithms. *Algorithmica*, 72(4):969–994, 2015.
15 J. Boyar, K.S. Larsen, and A. Maiti. A comparison of performance measures via online search. *Theoretical Computer Science*, 532:2–13, 2014.
16 A. Condon, A. Deshpande, L. Hellerstein, and N. Wu. Algorithms for Distributional and Adversarial Pipelined Filter Ordering Problems. *ACM Transactions on Algorithms*, 5(2):24:1–24:34, 2009.
17 J. Czyzowicz, K. Georgiou, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, and S. Shende. Search on a Line by Byzantine Robots. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC 2016)*, pages 27:1–27:12, 2016.
18 E.D. Demaine, S.P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361:342–355, 2006.
19 R. Dorrigiv and A. López-Ortiz. A Survey of Performance Measures for On-Line Algorithms. *SIGACT News*, 36(3):67–81, 2005.
20 S. Gal. A general search game. *Israel Journal of Mathematics*, 12:32–45, 1972.
21 S. Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27:17–30, 1974.
22 S. Gal. *Search Games*. Academic Press, 1980.
23 C. Hipke, C. Icking, R. Klein, and E. Langetepe. How to find a point in the line within a fixed distance. *Discrete Applied Mathematics*, 93:67–73, 1999.
24 P. Jaillet and M. Stafford. Online searching. *Operations Research*, 49:234–244, 1993.
25 M-Y. Kao and M.L. Littman. Algorithms for informed cows. In *Proceedings of the AAAI 1997 Workshop on Online Search*, 1997.
26 M-Y. Kao, J.H. Reif, and S.R. Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–80, 1996.
27 D. G. Kirkpatrick. Hyperbolic dovetailing. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2009.
28 E. Koutsoupias, C.H. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 280–289, 1996.
29 A. López-Ortiz and S. Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theoretical Computer Science*, 310(1–3):527–537, 2004.
30 A. McGregor, K. Onak, and R. Panigrahy. The oil searching problem. In *Proceedings of the 17th European Symposium on Algorithms (ESA)*, pages 504–515, 2009.
31 S. Schuierer. Lower bounds in online geometric searching. *Computational Geometry: Theory and Applications*, 18(1):37–53, 2001.

Bipartite Diameter and Other Measures Under Translation

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering,
New York University, Brooklyn, NY 11201, USA
boris.aronov@nyu.edu

Omrit Filtser
Department of Computer Science, Ben-Gurion University of the Negev,
Beer-Sheva 84105, Israel
omritna@post.bgu.ac.il

Matthew J. Katz
Department of Computer Science, Ben-Gurion University of the Negev,
Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

Khadijeh Sheikhan
Department of Computer Science and Engineering, Tandon School of Engineering,
New York University, Brooklyn, NY 11201, USA
khadijeh.sheikhan@gmail.com

Abstract

Let A and B be two sets of points in $\mathbb{R}^d$, where $|A| = |B| = n$ and the distance between them is defined by some bipartite measure $dist(A, B)$. We study several problems in which the goal is to translate the set B, so that $dist(A, B)$ is minimized. The main measures that we consider are (i) the *diameter* in two and three dimensions, that is $\text{diam}(A, B) = \max\{d(a, b) \mid a \in A, b \in B\}$, where $d(a, b)$ is the Euclidean distance between a and b, (ii) the *uniformity* in the plane, that is $\text{uni}(A, B) = \text{diam}(A, B) - d(A, B)$, where $d(A, B) = \min\{d(a, b) \mid a \in A, b \in B\}$, and (iii) the *union width* in two and three dimensions, that is $\text{union\_width}(A, B) = \text{width}(A \cup B)$. For each of these measures we present efficient algorithms for finding a translation of B that minimizes the distance: For diameter we present near-linear-time algorithms in $\mathbb{R}^2$ and $\mathbb{R}^3$, for uniformity we describe a roughly $O(n^{9/4})$-time algorithm, and for union width we offer a near-linear-time algorithm in $\mathbb{R}^2$ and a quadratic-time one in $\mathbb{R}^3$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Translation-invariant similarity measures, Geometric optimization, Minimum-width annulus

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.8

Funding *Boris Aronov*: Work on this paper by Boris Aronov was supported by NSF Grants CCF-11-17336, CCF-12-18791, and CCF-15-40656, and by grant 2014/170 from the US-Israel Binational Science Foundation.
Omrit Filtser: Work on this paper by Omrit Filtser was supported by the Israel Ministry of Science, Technology & Space, and by the Lynn and William Frankel Center.
Matthew J. Katz: Work on this paper by Matthew Katz was supported by grant 1884/16 from the Israel Science Foundation, and by grant 2014/170 from the US-Israel Binational Science Foundation.
Khadijeh Sheikhan: Work on this paper by Khadijeh Sheikhan was supported by NSF Grant CCF-12-18791.

Acknowledgements We would like to thank Pankaj K. Agarwal, Mark de Berg, and Timothy Chan for their assistance in the preparation of this manuscript.

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 8; pp. 8:1–8:14

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

1 Introduction

Determining the similarity between two sets of points in a metric space, and, in general, determining the value of some measure defined for two sets of points, is a well investigated problem in computational geometry. Sometimes, however, the answer that is obtained is meaningless, unless one of the sets undergoes some transformation before performing the computation. In this paper, we consider a family of problems in which the goal is to compute a translation which minimizes some bipartite measure. For example, one of the measures that we consider is the bipartite *diameter*, which is the distance between the farthest bichromatic pair, that is the maximum distance between a point from one set and a point from the other set.

The motivation for studying these problems is twofold. The first, as mentioned, is to find a translation for which the computed value is most meaningful. The second is when we are allowed to translate one of the sets in order to minimize some bipartite measure. In general, problems in which the goal is to find a transformation of a given type that minimizes or maximizes some measure are fundamental in computational geometry and have been studied extensively. It is therefore somewhat surprising that the natural versions that we study here have not been considered before. For example, another measure that we consider is the bipartite *uniformity*, which is the difference between the bipartite diameter and the distance between the closest bichromatic pair. When this difference is small, all bichromatic distances are similar, which is often a desirable property due to its close connection to the notions of fairness and balancing. Thus, the optimization problem in this case is to translate one of the sets to achieve the best possible uniformity.

Formally, let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be two sets of points in $\mathbb{R}^d$. For the sake of simplicity, we assume that $m = n$, and obtain bounds that depend only on n; however, it is not difficult to adapt our algorithms and bounds to the case where the sets A and B have different sizes. We are interested in problems of the following kind: Find a translation t^* that minimizes some bipartite measure of A and $B + t$ over all translations t, where $B + t$ denotes B translated by t.[1] The main bipartite measures that we consider are (i) *diameter*, denoted $\mathrm{diam}(A, B)$, and defined as $\max\{d(a, b) \mid a \in A, b \in B\}$, where $d(a, b)$ is the Euclidean distance between a and b, (ii) *uniformity*, denoted $\mathrm{uni}(A, B)$, and defined as $\mathrm{diam}(A, B) - d(A, B)$, where $d(A, B) = \min\{d(a, b) \mid a \in A, b \in B\}$, and (iii) *union width*, denoted $\mathrm{union\_width}(A, B)$, and defined as $\mathrm{width}(A \cup B)$, i.e., the width of the union of the two sets.

Notice that while for the (one-sided) Hausdorff distance (see below) one considers the distance from each point of A to its *closest* point in B, for the bipartite diameter measure one considers the distance from each point of A to its *farthest* point in B: The former variant is more relevant when B represents a set of homogeneous facilities, equally acceptable, while the latter variant is more relevant when B represents a set of unique facilities such that it is desirable to be close to all of them.

Related work. When comparing two sets of points of the same size, a natural approach is to find a matching or a mapping of one set to the other, such that the distances between the matched points are small. For instance, in the problem of congruence testing [6, 17],

[1] This class of problems naturally extends to other types of transformations, such as rotations, rigid motions, homothethies, similarity transformations, etc. In this paper, we will confine ourselves to translations, unless otherwise stated.

one needs to decide if there exists a geometric transformation (a combination of translation, rotation, and reflection) that maps a point set A exactly or approximately into a point set B of the same size. Another example is the well-known RMS distance, where the goal is to minimize the sum of squares of distances in a perfect matching between A and B [2].

In general, when one of the sets is larger than the other, we can look for a minimum partial matching, which in some sense corresponds to a copy of the smaller set in the larger one. This version of the problem (under various geometric transformations) was also widely investigated for bottleneck matching [8, 15], RMS distance [7], and more [20].

Another way to compare two sets of points of different sizes, is to use some bipartite distance measure for point sets, such as the well-known Hausdorff distance. The Hausdorff distance between two sets of points is the maximum of the distances from a point in each of the sets to the nearest point in the other set (the one-sided version of Hausdorff distance is a special case of our framework, but we do not consider it here beyond this summary). Huttenlocher et al. [19] showed that the minimum Hausdorff distance under translation in $\mathbb{R}^2$ can be computed in $O(mn(m+n)\alpha(mn)\log(mn))$ time, where m and n are the sizes of the two sets. The minimum Hausdorff distance under geometric transformations was widely investigated in the literature, and we refer to [2] for a survey of the results. A different example of bipartite measure is the maximum overlap between the convex hulls of the sets A and B. This measure was considered in [5], where, assuming A and B are point sets of size n in $\mathbb{R}^3$, an algorithm is presented that computes the optimal translation in expected time $O(n^3 \log^4 n)$.

In this paper, we focus on three bipartite measures under translation: diameter, uniformity, and width. To our knowledge, all three measures are being considered here for the first time.

The diameter of a set of n points in the plane can be computed in $O(n \log n)$ time. However, in higher dimensions the problem becomes much harder. Clarkson and Shor [13] gave a randomized algorithm with expected running time $O(n \log n)$ for points in $\mathbb{R}^3$, which is not very efficient in practice. Then there was a sequence of attempts to find a (simple) deterministic algorithm, which led to an optimal $O(n \log n)$ deterministic algorithm by Ramos [22].

The *width* of a set A of n points in the plane is the smallest distance between a pair of parallel lines, such that the closed strip between the lines contains A, and it can be easily computed in time $O(n \log n)$ using the rotating calipers method. However, again, in three dimensions the problem becomes harder, and the best-known algorithm is an $O(n^{3/2+\varepsilon})$ expected time algorithm, due to Agarwal and Sharir [3].

To compute the uniformity of two point sets under translation, we construct the minimum enclosing annulus of a set of n points in the plane (with only $O(\sqrt{n})$ extreme points). In [3], it is shown that the minimum enclosing annulus of n points in the plane (without a constraint on the number of extreme points) can be computed in $O(n^{3/2+\varepsilon})$ expected time, which is the current state of the art for this problem.

Our results. Consider the set $\mathcal{P} = \{a - b \mid a \in A, b \in B\}$ of all translations that take a point $b \in B$ to a point $a \in A$. We show that the optimal translations in the diameter and uniformity problems are the centers of the minimum enclosing circle of $\mathcal{P}$ and the minimum-width annulus containing $\mathcal{P}$, respectively. Thus, we could apply the best known algorithms for computing these objects to obtain solutions to these problems. More precisely, applying the algorithm of Megiddo [21] for computing the minimum enclosing ball would yield an $O(n^2)$-time solution for the diameter problem, in any fixed dimension, and applying the algorithm of Agarwal and Sharir [3] for computing the minimum-width annulus would

yield an $O(n^{3+\varepsilon})$-time solution for the uniformity problem in the plane. However, by making some additional observations and employing sophisticated known techniques, we are able to do much better. Specifically, we solve the diameter problem in $O(n \log n)$ time in the plane and in $O(n \log^2 n)$ expected time in three dimensions, and we solve the uniformity problem in the plane in $O(n^{9/4+\varepsilon})$ expected time, for any $\varepsilon > 0$. As a by-product of the latter result, we show that the minimum enclosing annulus of n points in the plane with only $O(\sqrt{n})$ extreme points can be computed in $O(n^{9/8+\varepsilon})$ expected time (in contrast to $O(n^{3/2+\varepsilon})$ expected time for the unconstrained case, see above).

For the union width problem under translation, we present an $O(n \log n)$-time solution in the plane and an $O(n^2)$-time one in three dimensions. Finally, we consider another new width-based measure, the *red-blue width*. The directional red-blue width (w.r.t. direction v) is the maximum red-blue distance after projecting the points onto a line parallel to direction v. The red-blue width is then defined as the minimum directional red-blue width over all directions. In other words, it measures the width of $A + (-B)$, the Minkowski sum of A and $-B$. We present solutions for the red-blue width problem under translation that run in time $O(n \log n)$ and $O(n^2)$, respectively, in the plane and in three dimensions.

2 Diameter

In the first problem that we consider, the measure is the bipartite diameter. Given two sets of points $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ in $\mathbb{R}^d$, the *bipartite diameter* of A and B is $\operatorname{diam}(A, B) = \max\{d(a, b) \mid a \in A, b \in B\}$, where $d(a, b)$ is the Euclidean distance between a and b.

▶ **Problem 1** (Bipartite Diameter under Translation)**.** *Find a translation t^* that minimizes the bipartite diameter of A and $B + t$ over all translations t. That is, for any translation t,* $\operatorname{diam}(A, B + t^*) \leq \operatorname{diam}(A, B + t)$.

Consider the set $\mathcal{P} = \{a - b \mid a \in A,\, b \in B\}$ of all possible translations taking a point of B to a point of A. Clearly, $|\mathcal{P}| = O(n^2)$.

▷ Claim 1. Let t be a translation and let S_t be the minimum enclosing ball of $\mathcal{P}$ centered at t. Then, the radius r_t of S_t is equal to $\operatorname{diam}(A, B + t)$.

Proof. Since r_t is the radius of the minimum enclosing ball of $\mathcal{P}$ centered at t,

$$\begin{aligned} r_t = \max_{a \in A,\, b \in B} d(a - b, t) &= \max_{a \in A,\, b \in B} \|(a - b) - t\| \\ &= \max_{a \in A,\, b \in B} \|a - (b + t)\| \\ &= \max_{a \in A,\, b \in B} d(a, b + t) = \operatorname{diam}(A, B + t). \end{aligned}$$

◁

▶ **Corollary 2.** *The optimal translation t^* minimizing bipartite diameter coincides with the center $c = c(\mathcal{P})$ of the minimum enclosing ball $S = S(\mathcal{P})$ of $\mathcal{P}$.*

Notice that Corollary 2 implies that the optimal translation t^* is unique. The minimum enclosing ball of a set of n points can be computed in linear time or expected linear time using, e.g., Megiddo's algorithm [21] or Welzl's randomized algorithm [24], respectively. Therefore, by Corollary 2, one can compute the optimal translation by simply finding c in $O(n^2)$ time. In this section we present near-linear-time algorithms for the problem in two and three dimensions.

Diameter in the plane. Let $\mathcal{Q}$ be the set of extreme points of $\mathcal{P}$. Denote by $\mathrm{CH}(X)$ the convex hull of a point set X.

Since $\mathcal{P}$ is the Minkowski sum of two sets of size n, it is well known [14] that $\mathcal{Q}$ has size $O(n)$ and can be constructed in linear time from $\mathrm{CH}(A)$ and $\mathrm{CH}(B)$ using the rotating calipers method of [18, 23].

Once $\mathcal{Q}$ is constructed, we compute its minimum enclosing disk $S' = S(\mathcal{Q}) = S(\mathcal{P})$.

▶ **Theorem 3.** *Let A and B be two sets of points in $\mathbb{R}^2$, both of size n. A translation t^* that minimizes the bipartite diameter of A and $B + t$ can be found in $O(n \log n)$ time.*

Diameter in three dimensions. We describe an algorithm for computing the minimum enclosing ball of $\mathcal{P}$, without computing $\mathcal{P}$ (whose size may be $\Theta(n^2)$) explicitly. We adapt Clarkson's scheme for solving LP-type problems [12] to the problem of computing the minimum enclosing ball of a set of points; see [1] for a similar-in-spirit adaptation of Clarkson's scheme to an entirely different situation.

The high-level algorithm uses an initially empty set X of points. It repeats the following process until the minimum enclosing ball is found.

1: Pick a random sample $\mathcal{R}$ of $\mathcal{P}$ of size $4n$.
2: Compute the minimum enclosing ball $S = S(\mathcal{R} \cup X)$.
3: Find the set of *violators* V, i.e., the set of all points of $\mathcal{P}$ that are not in S. If $|V| > 2n$ (there are too many violators), go to 1.
4: If $V = \emptyset$, then return S and stop, else $X \leftarrow X \cup V$ and go to 1.

We call an iteration of the algorithm that reaches line 4 "successful." Clarkson's analysis establishes that in each iteration the expected size of V is n. Therefore, for a random choice of $\mathcal{R}$, the probability of the number of violators being at most $2n$ is at least $\frac{1}{2}$, so an iteration is unsuccessful with probability at most $\frac{1}{2}$. In particular, a constant expected number of unsuccessful iterations is followed by a successful one.

On the other hand, it is not difficult to check (see Clarkson's analysis once again) that, when violators are found, one of the violators must be a point defining the minimum enclosing ball. Therefore, the number of successful iterations cannot exceed five: each iteration adds at least one of the points defining the desired ball to X and once all of them are in X, the optimal ball is discovered in line 2, there are no further violators, and the algorithm stops. Therefore the total number of iterations is expected to be $O(1)$ and the size of the set X never grows beyond $O(n)$. Thus in each iteration we invoke a standard minimum-ball algorithm on $O(n)$ points, requiring $O(n)$ expected time.

Next, we describe how to efficiently implement steps 1 and 3. A random sample of $\mathcal{P}$ can be obtained by repeatedly picking random points $a \in A$ and $b \in B$ and returning $a - b$.

The set of violators V can be found by modifying an algorithm by Chazelle et al. [10] for kth nearest neighbor search. First, consider the following problem:

▶ **Problem 2.** *Given two sets A and B, each of n points in $\mathbb{R}^3$, and a distance r, decide whether there are two points $a \in A$ and $b \in B$ with $d(a, b) > r$.*

This problem can be solved in $O(n \log n)$ expected time by the following algorithm:

1: Set $\mathcal{I}_A = \bigcap_{a \in A} D(a, r)$, where $D(a, r)$ is the ball of radius r centered at a, and construct a corresponding inside/outside point-location data structure. (This structure preprocesses the set $\{D(a, r) | a \in A\}$ to facilitate point location queries of the form "Given a point q, is it contained in $\mathcal{I}_A$ or not?"). $\mathcal{I}_A$, together with its corresponding inside/outside point-location data structure, can be computed using the randomized $O(n \log n)$-time algorithm of Clarkson and Shor [13], after which queries are answered in $O(\log n)$ time.

2: If $\mathcal{I}_A = \emptyset$, then clearly there exist two such points. Otherwise, check for each $b \in B$ whether $b \in \mathcal{I}_A$. This can be done by n point-location queries in total $O(n \log n)$ time. If for some $b \in B$, $b \notin \mathcal{I}_A$, there exists some $a \in A$ for which $d(a, b) > r$.

Now we consider the thresholded reporting version of Problem 2:

▶ **Problem 3.** *Given two sets A and B, each of n points in $\mathbb{R}^3$, a distance r and a parameter k, report all the pairs of points $a \in A$, $b \in B$ with $d(a, b) > r$, if there are at most k such pairs. Otherwise, return* `TOO_MANY` *without necessarily listing them.*

The reporting problem can be solved by building a binary tree of point-location data structures. The root of the tree corresponds to $\mathcal{I}_A$. Next, we arbitrarily divide A into two subsets A_1, A_2 of size $n/2$, and build two new point-location data structures, corresponding to $\mathcal{I}_{A_1}$ and $\mathcal{I}_{A_2}$, respectively. Then we continue recursively for A_1 and A_2. The total expected preprocessing time is $O(n \log^2 n)$.

To report the pairs with distance larger than r, we simply query the nodes of the tree as in step 2 of the decision algorithm above. Given some $b \in B$, if $b \in \mathcal{I}_A$, then $d(a, b) \leq r$ for all $a \in A$ and we can stop the search with b. Else, if $b \notin \mathcal{I}_A$, then there exists some $a \in A$ for which $d(a, b) > r$. In this case we check $\mathcal{I}_{A_1}$ and $\mathcal{I}_{A_2}$ recursively. At a leaf, $\mathcal{I}_{\{a\}} = D(a, r)$, so $b \notin \mathcal{I}_{\{a\}}$ means $d(a, b) > r$; in that case, report the pair (a, b). Keep count of the pairs reported so far. If more than k pairs have been reported, stop and return `TOO_MANY`. For any reported pair, we visit $O(\log n)$ nodes of the tree, including the ones where no pairs were reported, and perform a logarithmic-time point-location query at each node. An additional query is performed for every $b \in B$ that is not part of any pair to be reported. Thus the running time of the reporting phase is no more than $O(n \log n + k \log^2 n)$.

The expected total running time of the algorithm is $O((n + k) \log^2 n)$.

▶ **Observation 4.** *Let o be the center of the ball S and r its radius. A point $a - b \in \mathcal{P}$ is in S if and only if $d(a, b + o) \leq r$.*

Proof. The point $a - b$ is in S if and only if $d(a - b, o) \leq r$, and $d(a - b, o) = \|(a - b) - o\| = \|a - (b + o)\| = d(a, b + o)$. ◀

The set of violators V can be found by solving problem 3 with the input A, $B + o = \{b + o \mid b \in B\}$, the radius of S, and $k = 2n$. We summarize our result.

▶ **Theorem 5.** *Let A and B be two sets of points in $\mathbb{R}^3$, both of size n. A translation t^* that minimizes the bipartite diameter of A and $B + t$ can be found in $O(n \log^2 n)$ expected time.*

3 Uniformity

Define $\mathrm{uni}(A, B)$ as the difference between the largest and the smallest distances between a point of A and a point of B. Formally, we set $\mathrm{uni}(A, B) = \mathrm{diam}(A, B) - d(A, B)$, where $d(A, B) = \min\{d(a, b) \mid a \in A, b \in B\}$. The quantity $\mathrm{uni}(A, B)$ measures the uniformity of the red-blue distances. The smaller it is, the more uniform are the distances. One may consider minimizing the ratio rather than the difference of these quantities, which we leave for future research. In this section we consider the following problem:

▶ **Problem 4** (Bipartite Uniformity under Translation)**.** *Find a translation t^* that minimizes the uniformity of A and $B + t$. That is, for any translation t,* $\mathrm{uni}(A, B + t^*) \leq \mathrm{uni}(A, B + t)$.

We study this problem in the plane. Notice that in general t^* may not be unique.

▷ Claim 6. Let c be the center of a minimum-width enclosing annulus of $\mathcal{P}$. Then, $t^* = c$.

Proof. Similarly to the proof of Claim 1, for any translation t, the annulus S_t centered at the point t with radii $\mathrm{diam}(A, B + t)$ and $d(A, B + t)$ (S_t's width is thus $\mathrm{uni}(A, B + t)$), contains all the points of $\mathcal{P}$. Indeed, given some $a - b \in \mathcal{P}$, we have $d(a - b, t) = d(a, b + t)$ and $d(A, B + t) \leq d(a, b + t) \leq \mathrm{diam}(A, B + t)$. Since c is the center of the minimum-width enclosing annulus of $\mathcal{P}$, we get $\mathrm{uni}(A, B + c) \leq \mathrm{uni}(A, B + t)$ for any translation t. ◁

We are thus left with the following algorithmic problem.

▶ **Problem 5** (Restricted Minimum-Width Annulus)**.** *Given a set $\mathcal{P}$ of n^2 points in the plane with only $O(n)$ extreme points, compute the minimum-width annulus covering $\mathcal{P}$.*

Note that if we apply a standard quadratic-time algorithm from the textbook [14] to $\mathcal{P}$ as a black box, we would obtain running time $O(n^4)$. Instead, we could apply the cutting-edge algorithm of Agarwal and Sharir [3] to $\mathcal{P}$, again as a black box, to achieve $O(n^{3+\varepsilon})$ expected running time. But, as we shall see below, we improve these bounds by a more refined use of these and other tools, for the specific situation presented above.

Let $\mathcal{Q} \subset \mathcal{P}$ be the set of extreme points of $\mathcal{P}$. Let $F = \mathrm{FVor}(\mathcal{Q})$ be the farthest-point Voronoi diagram of $\mathcal{Q}$, and let $V = \mathrm{Vor}(\mathcal{P})$ be the closest-point Voronoi diagram of $\mathcal{P}$. We compute V of size $O(n^2)$ in time $O(n^2 \log n)$ and F of size $O(n)$ in time $O(n \log n)$. It is known (see, for example, [14, Section 7.4]) that the center of the minimum-width annulus covering $\mathcal{P}$ must lie at (i) a vertex of F, (ii) a vertex of V, or (iii) an intersection point between an edge of F and an edge of V. Cases (i) and (ii) can be handled in $O(n^2 \log n)$ time. Indeed, one can preprocess both F and V for point location and then locate vertices of each diagram in the other, obtaining the identities of the closest and farthest points of $\mathcal{P}$ for each Voronoi vertex and allowing one to compute the width of the annulus centered at it.

Hereafter we focus on case (iii). Its naïve implementation requires $\Omega(n^3)$ time, as the number of intersections between edges of F and V might be cubic in the worst case. (Indeed, an $O(n^3)$-time algorithm exists that simply overlays F and V. The vertices of the overlay are precisely the points described in cases (i) through (iii) above. We can now process each point in amortized constant time. See Section 4 of [14] for the routine details.)

Complete bipartite clique decomposition To do better, we start by recalling a variant of a classical fact, first observed in [11].

▶ **Fact 7.** *Let C and D be two sets, each consisting of non-crossing line segments in the plane, with $|C| = n$, $|D| = m$, and $n < m$. Then there exists a collection of pairs $\{(C_i, D_i)\}$ such that:*

(a) *$C_i \subset C$ and $D_i \subset D$.*
(b) *For every intersecting pair of segments $(c, d) \in C \times D$, there exists a unique i such that $(c, d) \in C_i \times D_i$.*
(c) *For every i, every segment in C_i intersects every segment in D_i and the slopes of all segments in C_i are larger than the slopes of all segments of D_i, or vice versa.*
(d) *The collection $\{(C_i, D_i)\}$ can be constructed in time $O((n + m) \log^2 n)$.*
(e) *The number of pairs in the collection is $O(n \log n)$.*
(f) *$\sum_i |C_i| = O(n \log^2 n)$ and $\sum_i |D_i| = O(m \log^2 n)$.*

We outline the proof here, as the version we need is slightly more general than the most commonly used one, such as in [1,3] (see [1] for a very similar construction; the distinction is in item (f), where we need separate bounds on $\sum_i |C_i|$ and $\sum_i |D_i|$); the usual assumption is that $n = m$ while in the application below we will set $m = n^2$.

Proof. Construct a 2-level hereditary segment tree on C: Build a segment tree on the segments of C so that each segment appears in $O(\log n)$ nodes and each node ν corresponds to a canonical vertical strip S_ν and a vertically ordered list C_ν of (parts) of segments of C that completely cross S_ν left-to-right. For the second level, store each of the sets C_ν in a separate balanced binary tree T_ν in vertical order; each node μ of T_ν stores a canonical subset C_μ of contiguous segments of C_ν; at the node μ we also store a second set $D_\mu \subset D$ of segments, initially empty. Now query the structure with each segment $d \in D$. It crosses $O(\log n)$ canonical vertical strips completely and its endpoints land in two leaves of the primary tree, which correspond to elementary canonical strips.

For each strip S_ν completely spanned by d, d crosses a contiguous portion of the segments of C_ν, represented by $O(\log n)$ canonical subsets, each corresponding to a node μ in T_ν. We add d to D_μ, for all such choices of ν and μ.

A very similar process handles the endpoints of D.

Having repeated this process for each $d \in D$,[2] we output (C_μ, D_μ) for all secondary tree nodes μ. It is easily verified that a pair of segments $(c, d) \in C \times D$ cross if and only if there is a (unique) μ with $d \in C_\mu$ and $d \in D_\mu$.

The number of nodes μ in the secondary tree of S_ν is $O(|C_\nu|)$ and hence the number of pairs (C_μ, D_μ) is $O(n \log n)$. By construction, each segment $c \in C$ appears in $O(\log^2 n)$ nodes of the structure and we touch $O(\log^2 n)$ nodes when searching for $d \in D$. This implies the bounds on $\sum_\mu |C_\mu|$ and on $\sum_\mu |D_\mu|$. ◀

Reduction to the minimum-"distance" problem between lines in three dimensions. We now use a reinterpretation of the problem, first noticed in [4] and most recently used in [3] to efficiently compute the minimum-width annulus covering a finite point set in the plane.

Lift the points of $\mathcal{P}$ to the standard paraboloid $z = x^2 + y^2$, obtaining the set $\mathcal{P}^*$ and the corresponding set $\mathcal{Q}^* \subset \mathcal{P}^*$; we will use an asterisk to denote a lifted object. As is well known, a minimal disk enclosing $\mathcal{P}$ in the plane corresponds to an upper tangent plane to the convex hull $\mathrm{CH}(\mathcal{P}^*)$ of $\mathcal{P}^*$ (which coincides with the upper convex hull of $\mathcal{Q}^*$), while a maximal disk empty of points of $\mathcal{P}$ corresponds to a lower tangent plane to $\mathrm{CH}(\mathcal{P}^*)$. In case (iii) described above, the upper plane passes through an edge $q_1^*q_2^*$ of the upper hull of $\mathcal{Q}^*$ and the lower plane through an edge $p_1^*p_2^*$ of the lower hull of $\mathrm{CH}(\mathcal{P}^*)$. The two planes are parallel and this event corresponds precisely to the intersection of an edge c of V separating the regions of p_1 and of p_2 and an edge d of F separating the regions of q_1 and of q_2.

It was observed in [4] that the width of the (minimal) annulus containing $\mathcal{P}$ and centered at $c \cap d$ corresponds to the "distance" between two parallel planes passing through the lines supporting edges $q_1^*q_2^*$ and $p_1^*p_2^*$ in $\mathbb{R}^3$; the distance is not measured using the conventional Euclidean metric, but using a different function that satisfies the properties enumerated in [3] (another application of their machinery is for computing the three-dimensional width of a finite point set in $\mathbb{R}^3$; in that application the distance is Euclidean, for pairs of edges that support parallel planes sandwiching the set; see [3] for the details).

In other words, we need to solve the following problem: For all pairs of lines $q_1^*q_2^*$, $p_1^*p_2^*$ supporting upper and lower edges of $\mathrm{CH}(\mathcal{P}^*)$ as above that correspond to a pair of crossing edges of F and V, find the shortest "distance" between the lines $q_1^*q_2^*$ and $p_1^*p_2^*$.

---

[2] One needs to repeat the process twice: once for d's that are "steeper" than segments of C and once for those "less steep." More precisely, for each strip S_ν, we classify segments d that span S_ν into two classes: Those that cross the left edge of S_ν lower than the right edge, *relative to the segments* of C_ν, and those that cross the left edge higher than the right edge. This way in the final pair (C_μ, D_μ) either all segments of C_μ cross those from D_μ "from below," or all "from above."

Complete bipartite case. Apply Fact 7 to the two sets of edges (line segments) C and D, producing a decomposition into pairs $\{(C_i, D_i)\}$ with the described properties. We now focus on one such pair, (C_i, D_i). By construction, each pair of edges $(c, d) \in C_i \times D_i$ intersect. We now perform the calculation on the corresponding pair of sets of lifted lines (C_i^*, D_i^*), using the "distance" defined above:

▶ **Fact 8** (Agarwal and Sharir [3]). *Given a set X of n lines and a set Y of m lines, so that every line of X lies above every line of Y, the shortest "distance" between a line of X and a line of Y can be computed in expected time $O(n^{3/4+\varepsilon}m^{3/4+\varepsilon} + n^{1+\varepsilon} + m^{1+\varepsilon})$, for any $\varepsilon > 0$.*

In particular, the best annulus width corresponding to points $c \cap d$, with $(c, d) \in (C_i \times D_i)$, corresponds precisely to the shortest "distance" between C_i^*, D_i^* as above and can be computed using Fact 8 in time $O(n_i^{3/4+\varepsilon}m_i^{3/4+\varepsilon} + n_i^{1+\varepsilon} + m_i^{1+\varepsilon})$, where $n_i = |C_i|$ and $m_i = |D_i|$.

Putting it all together. Recall that in our case $m = n^2$. Therefore, the total work required includes $O(n^2 \log n)$ for cases (i) and (ii), $O((n+m)\log^2 n) = O(n^2 \log^2 n)$ for constructing the pairs $\{(C_i, D_i)\}$, and finally the following for processing every pair (C_i, D_i), using Fact 8:

$$\sum_i O(n_i^{3/4+\varepsilon}m_i^{3/4+\varepsilon} + n_i^{1+\varepsilon} + m_i^{1+\varepsilon}),$$

subject to the constraints described in Fact 7. We bound the above expression by

$$O(n^{3\varepsilon}) \cdot \sum_i O(n_i^{3/4}m_i^{3/4} + n_i + m_i),$$

where we have used the facts that $n_i \le n$ and $m_i \le n^2$, for all i. Since $\sum_i n_i = O(n \log^2 n) = o(n^{1+\varepsilon})$ and $\sum_i m_i = O(m \log^2 n) = o(n^{2+\varepsilon})$, the last two terms are bounded by $o(n^{2+\varepsilon})$. We proceed to focus on the larger first term.

Using Hölder's inequality, we have

$$\begin{aligned}\sum_i m_i^{3/4}n_i^{3/4} &= \sum_i m_i^{3/4}(n_i^3)^{1/4} \le (\sum_i m_i)^{3/4} \cdot (\sum_i n_i^3)^{1/4} \\ &\le O(n^2 \log^2 n)^{3/4}(n^3 \cdot O(\log^2 n))^{1/4} = O(n^{9/4} \log^2 n) = O(n^{9/4+\varepsilon}),\end{aligned}$$

where we have used the fact that $\sum_i m_i = O(n^2 \log^2 n)$, $\sum_i n_i = O(n \log^2 n)$, and $n_i \le n$. Plugging everything together, the expected running time of the entire algorithm is $O(n^{9/4+4\varepsilon})$. Replacing ε by $\varepsilon/4$ in the above reasoning, we obtain:

▶ **Theorem 9.** *Given a set $\mathcal{P}$ of n^2 points in the plane that has $O(n)$ extreme points, the total expected time required to compute the minimum-width annulus enclosing $\mathcal{P}$ is $O(n^{9/4+\varepsilon})$, for any positive ε.*

Returning to our original motivation, we conclude:

▶ **Theorem 10.** *Let A and B be two sets of points in $\mathbb{R}^2$, both of size n. A translation t^* that minimizes the uniformity of A and $B + t$ can be found in $O(n^{9/4+\varepsilon})$ time.*

4 Width

In this section, first we minimize the union width measure. The width of a point set is the smallest distance between two parallel supporting hyperplanes of the set. Given two sets of points $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ in $\mathbb{R}^d$, the *union width* of A and B is defined as the width of their union, namely, union_width$(A, B) =$ width$(A \cup B)$.

▶ **Problem 6** (Union Width under Translation). *Find a translation t^* that minimizes the union width of A and $B+t$ over all translations t. That is, for any translation t, we have* $\text{union\_width}(A, B+t^*) \leq \text{union\_width}(A, B+t)$.

Directional width. The *directional width* function $\text{width}_v(X)$ of a compact set X in $\mathbb{R}^d$ gives, for every direction v, the distance between the two supporting hyperplanes of X that are orthogonal to v:

$$\text{width}_v(X) = \max_{x_1,x_2 \in X} (x_1 - x_2) \cdot v.$$

In particular, the width of a set corresponds to the minimum of its directional widths. We define the *directional union width* of A and B as the directional width of their union:

$$\text{union\_width}_v(A, B) = \text{width}_v(A \cup B).$$

▶ **Problem 7** (Directional Union Width under Translation). *For a given direction v, find a translation t^* that minimizes the directional union width of A and $B+t$ over all translations t. That is, for any translation t,* $\text{union\_width}_v(A, B+t^*) \leq \text{union\_width}_v(A, B+t)$.

▷ Claim 11. For a given direction v, the minimum directional union width under translation, $\text{union\_width}_v(A, B+t)$, is equal to the maximum of $\text{width}_v(A)$ and $\text{width}_v(B)$.

Proof. To obtain the smallest directional width we translate B so that the slab between the supporting hyperplanes of the wider set contains the other set, then $\text{union\_width}_v(A, B+t)$ will be equal to the directional width of the wider set which means:

$$\text{union\_width}_v(A, B+t) = \max(\text{width}_v(A), \text{width}_v(B)).$$ ◁

This claim reduces Problem 7 to finding the maximum of two directional widths. Now we return to Problem 6, which now reduces to finding the minimum value of the function $\max(\text{width}_v(A), \text{width}_v(B))$ over all directions v. In Sections 4.1 and 4.2, we present an $O(n \log n)$-time algorithm for the two-dimensional and a quadratic-time algorithm for the three-dimensional version of the problem. Finally, in Section 4.3, we define a bipartite measure closely related to width and show that it can be computed using similar methods with slight modifications.

4.1 Width in the plane

To compare the width of the two sets in different directions and measure the union width, we first compute the convex hulls of the two sets in $O(n \log n)$ time. We use the rotating calipers method of [18,23] to construct, for both A and B, their directional width functions $\text{width}_v(A)$ and $\text{width}_v(B)$. Each of these two functions is a piecewise-algebraic function (with a suitable choice of parametrization) of low degree with $\Theta(n)$ breakpoints. Now consider their pointwise maximum defined by $\max(\text{width}_v(A), \text{width}_v(B))$. The global minimum of this function, according to Claim 11, determines the answer to Problem 7. It can be computed by merging the two lists of breakpoints and computing the intersections between the function graphs in each interval; the resulting function still has $O(n)$ breakpoints in total and its minimum can be computed in linear time which results in the following theorem.

▶ **Theorem 12.** *The union width of two n-point sets in the plane can be computed in $O(n \log n)$ time.*

4.2 Width in three dimensions

To better understand the problem, we first review the tools used to compute the standard width of a set in $\mathbb{R}^3$. Recall that computing the width is equivalent to finding the smallest-width slab enclosing the set. In their paper [18], Houle and Toussaint showed that two supporting planes with minimum distance apart pass through either an antipodal vertex-face (VF) pair or an antipodal edge-edge (EE) pair of the convex hull. To compute and compare the antipodal pairs, they used the *Gauss map* (also called the *normal diagram*). In this transformation, which was originally introduced to computational geometry by Brown [9], the convex hull of the point set is mapped to the surface of a unit sphere $\mathbb{S}^2$. Every face is mapped to a point (the direction of its outer normal), every edge is mapped to the great circle arc connecting its two neighboring faces (the locus of the directions of all planes supporting the set at the edge), and every vertex is mapped to a region (the locus of the directions of all supporting planes at the vertex). Then they overlay the upper hemisphere of the Gauss map on the lower hemisphere and compute the intersections between them. We call the resulting diagram the *antipodal diagram*. Each vertex of the overlay corresponds to an antipodal VF or EE pair, and the width can be determined by computing the distance of the antipodal pair at these vertices and choosing the one with the smallest such distance.

The antipodal diagram encodes the antipodal pair of features for all directions and can be viewed as a representation of the directional width function; in particular, it can be used to compute the directional width for any given direction. As mentioned above, Houle and Toussaint showed that the minimum can only occur at the vertices of the antipodal diagram, not in the middle of an edge nor in the interior of a face [18].

To solve Problem 6, we need to represent the antipodal pairs and directional pairs for both sets together. We create the new combined antipodal diagram by overlaying the antipodal diagrams for A and for B.

▶ **Observation 13.** *If the minimum directional union width under translation occurs at direction v^*, then one of the following must occur (as it holds for the maximum of any two functions):*

1. $\text{width}_{v^*}(A) \geq \text{width}_{v^*}(B)$ *and v^* is a local minimum for* $\text{width}_{v^*}(A)$,
2. $\text{width}_{v^*}(B) \geq \text{width}_{v^*}(A)$ *and v^* is a local minimum for* $\text{width}_{v^*}(B)$, *or*
3. $\text{width}_{v^*}(A) = \text{width}_{v^*}(B)$ *and neither function has a local minimum at v^*.*

In cases 1 and 2, the optimal direction is a local optimum of one of the two sets as well and occurs at a vertex of the antipodal diagram. But what happens in case 3? Is it possible that the minimum occurs in the middle of an edge or in the interior of a face? In order to answer these questions we use the following lemma [18]:

▶ **Lemma 14** (Houle and Toussaint [18])**.** *Let ℓ_1 and ℓ_2 be parallel lines in $\mathbb{R}^3$. Let π_1 and π_2 be distinct parallel planes containing ℓ_1 and ℓ_2, respectively. Then there exists a preferred direction of rotation such that if π_1 and π_2 are rotated about ℓ_1 and ℓ_2, respectively, in that direction to form new parallel planes π_1' and π_2', then $d(\pi_1', \pi_2') < d(\pi_1, \pi_2)$.*

▷ Claim 15. The minimum value of $\max(\text{width}_v(A), \text{width}_v(B))$ cannot occur in the interior of a face of the antipodal diagram.

Proof. Suppose for the sake of contradiction that the optimal direction v^* lies in the interior of a face of the diagram. Being in the interior means each set has an antipodal VV pair in direction v^*. Since a VV pair cannot be an optimal direction for either of the sets separately, according to Observation 13 the directional widths of A and B are equal, and we

may translate them so that the two corresponding parallel slabs coincide. Therefore, each supporting plane passes through exactly one vertex from each set. Let π_1 and π_2 be the two supporting planes with $a_1, b_1 \in \pi_1$ and $a_2, b_2 \in \pi_2$. We can translate B so that b_1 is translated to a_1. After translation, let $\ell_2 \subset \pi_2$ be the line through a_2 and b_2 and let $\ell_1 \subset \pi_1$ be the line through a_1 parallel to ℓ_1. According to Lemma 14, there is a direction to rotate the two planes so that they remain supporting for both sets, but the distance between them is reduced, contradicting v^* being the optimal direction. ◁

We proved that minimum union width cannot occur in the interior of a face; however, unlike the width of a single set, the minimum union width may occur in the interior of an antipodal diagram edge.

(An example when this happens will be described in the full version of this paper.) Even though comparing directional width at vertices of the antipodal diagram is not sufficient anymore, the following theorem proves that the union width still can be computed in quadratic time.

▶ **Theorem 16.** *The union width of two n-point sets in three dimensions can be computed in $O(n^2)$ time.*

Proof. Each of the four subdivisions used to create the antipodal diagram for the union width has linear complexity, so their overlay has complexity $O(n^2)$ and can be computed in $O(n^2)$ time using convex subdivision overlay algorithm of Guibas and Seidel [16]. Although the minimum union width can occur at an interior point of a diagram edge, we can still compute it in $O(n^2)$. Directional union width function along each edge has constant complexity and we can find its minimum value in constant time. Since there are at most $O(n^2)$ edges and vertices in the diagram, we can compute the minimum union width in $O(n^2)$ time. ◀

4.3 Red-blue width

We now present a different interpretation of the width of a set, to motivate the definition of a new bipartite measure. Directional width of a point set X in a given direction v is the maximum of all the pairwise distances projected on that direction, $\max_{x_1,x_2 \in X}(x_1 - x_2) \cdot v$. For two sets A and B, we define the *directional red-blue width* as

$$\text{rb\_width}_v(A, B) = \max_{a \in A, b \in B} (a - b) \cdot v,$$

and the *red-blue width* of A and B as the minimum of all the directional red-blue widths:

$$\text{rb\_width}(A, B) = \min_v \text{rb\_width}_v(A, B).$$

▶ **Problem 8** (Red-blue Width under Translation)**.** *Find a translation t^* that minimizes the red-blue width of A and $B + t$ over all translations t. That is, for any translation t, we have* $\text{rb\_width}(A, B + t^*) \leq \text{rb\_width}(A, B + t)$.

▷ Claim 17. For a given direction v, the minimum directional red-blue width under translation is equal to the average of $\text{width}_v(A)$ and $\text{width}_v(B)$.

Proof. Since all the distances are projected on a line parallel to v, one can use the projection of the points to compute the width. The two sets A and B get projected to intervals with lengths equal to $\text{width}_v(A)$ and $\text{width}_v(B)$, respectively. A translation of B will translate its corresponding interval along the line without changing its length. The extreme distances that define the red-blue width are between the leftmost point of A and the rightmost point of B, and vice versa. The maximum of these two distances is always at least $\text{width}_v(A)/2 + \text{width}_v(B)/2$ and is realized when the two interval centers are aligned. ◁

So to solve Problem 8, we need to minimize the sum of the two directional widths, rather than their maximum, as in Problem 6. Using the same techniques with slight modifications we obtain the following result; details are omitted in this version.

▶ **Theorem 18.** *The red-blue width of two n-point sets A and B under translation can be computed in $O(n \log n)$ time in the plane and $O(n^2)$ time in the three-dimensional space.*

References

1 Pankaj K. Agarwal, Boris Aronov, Marc van Kreveld, Maarten Löffler, and Rodrigo Silveira. Computing Correlation between Piecewise-Linear Functions. *SIAM J. Comput.*, 42:1867–1887, 2013.

2 Pankaj K. Agarwal, Sariel Har-Peled, Micha Sharir, and Yusu Wang. Hausdorff distance under translation for points and balls. In *Proc. 19th Ann. Symp. Comput. Geometry*, pages 282–291. ACM, 2003.

3 Pankaj K. Agarwal and Micha Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discr. Comput. Geometry*, 16(4):317–337, 1996.

4 Pankaj K. Agarwal, Micha Sharir, and Sivan Toledo. Applications of Parametric Searching in Geometric Optimization. *J. Algorithms*, 17(3):292–318, 1994. doi:10.1006/jagm.1994.1038.

5 Hee-Kap Ahn, Peter Brass, and Chan-Su Shin. Maximum overlap and minimum convex hull of two convex polyhedra under translations. *Computational Geometry*, 40(2):171–177, 2008.

6 Helmut Alt, Kurt Mehlhorn, Hubert Wagener, and Emo Welzl. Congruence, similarity, and symmetries of geometric objects. *Discr. Comput. Geometry*, 3(3):237–256, 1988.

7 Rinat Ben-Avraham, Matthias Henze, Rafel Jaume, Balázs Keszegh, Orit E. Raz, Micha Sharir, and Igor Tubis. Minimum partial-matching and Hausdorff RMS-distance under translation: Combinatorics and algorithms. In *European Symp. Algorithms*, pages 100–111. Springer, 2014.

8 Arijit Bishnu, Sandip Das, Subhas C. Nandy, and Bhargab B. Bhattacharya. Simple algorithms for partial point set pattern matching under rigid motion. *Pattern Recognition*, 39(9):1662–1671, 2006.

9 K. Q. Brown. *Geometric Transforms for Fast Geometric Algorithms*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA, 1980.

10 Bernard Chazelle, Richard Cole, Franco P. Preparata, and Chee Yap. New upper bounds for neighbor searching. *Information and Control*, 68(1-3):105–124, 1986.

11 Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

12 Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42:488–499, 1995.

13 Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discr. Comput. Geometry*, 4(1):387–421, 1989.

14 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

15 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

16 Leonidas J. Guibas and Raimund Seidel. Computing convolutions by reciprocal search. *Discr. Comput. Geometry*, 2(2):175–193, 1987.

17 Paul J. Heffernan and Stefan Schirra. Approximate decision algorithms for point set congruence. *Computational Geometry*, 4(3):137–156, 1994.

18 M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-10(5):761–765, 1988.

19 Daniel P. Huttenlocher, Klara Kedem, and Micha Sharir. The Upper Envelope of Voronoi Surfaces and Its Applications. *Discr. Comput. Geometry*, 9:267–291, 1993. doi:10.1007/BF02189323.

20 Piotr Indyk and Suresh Venkatasubramanian. Approximate congruence in nearly linear time. *Computational Geometry*, 24(2):115–128, 2003.

21 Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.

22 Edgar A. Ramos. Deterministic algorithms for 3-D diameter and some 2-D lower envelopes. In *Proc. 16th Ann. Symp. Comput. Geometry*, pages 290–299. ACM, 2000.

23 M. I. Shamos. *Computational Geometry.* PhD thesis, Yale University, 1978.

24 Emo Welzl. Smallest Enclosing Disks (balls and Ellipsoids). In *Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.

Solving Simple Stochastic Games with Few Random Nodes Faster Using Bland's Rule

David Auger
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, France
david.auger@uvsq.fr

Pierre Coucheney
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, France
pierre.coucheney@uvsq.fr

Yann Strozecki
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, France
yann.strozecki@uvsq.fr

Abstract

The best algorithm so far for solving Simple Stochastic Games is Ludwig's randomized algorithm [21] which works in expected $2^{O(\sqrt{n})}$ time. We first give a simpler iterative variant of this algorithm, using Bland's rule from the simplex algorithm, which uses exponentially less random bits than Ludwig's version. Then, we show how to adapt this method to the algorithm of Gimbert and Horn [15] whose worst case complexity is $O(k!)$, where k is the number of random nodes. Our algorithm has an expected running time of $2^{O(k)}$, and works for general random nodes with arbitrary outdegree and probability distribution on outgoing arcs.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases simple stochastic games, randomized algorithm, parametrized complexity, strategy improvement, Bland's rule

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.9

Related Version A full version of the paper is available at `https://arxiv.org/abs/1901.05316`.

1 Introduction

A *simple stochastic game*, SSG for short, is a two-player zero-sum game, a turn-based version of *stochastic games* introduced by Shapley [22]. SSGs were introduced by Condon [11] and provide a simple framework that allows to study algorithmic complexity issues underlying reachability objectives. An SSG is played by moving a pebble on a graph. Some nodes are divided between players MIN and MAX: if the pebble reaches a node controlled by a player then she has to move the pebble along an arc leading to another node. Some other nodes are ruled by chance, the pebble following one outgoing arc according to some given probability distribution. Finally, there are sink nodes with a rational value, which is the gain that MAX-player achieves when the pebble reaches this sink.

Player MAX's objective is, given a starting node for the pebble, to maximize the expectation of her gain against any strategy of MIN. One can show that it is enough to consider stationary deterministic strategies for both players [11]. Though seemingly simple since the number of stationary deterministic strategies is finite, the task of finding a pair of optimal strategies, or equivalently, of computing the so-called *optimal values* of nodes, is in complexity class PPAD [13] but not known to be in P.

Simple stochastic games are a powerful model since they can simulate many other games such as parity games, mean or discounted payoff games [2, 7]. However these games are believed to be simpler than SSGs and better algorithms are known for them; in particular, parity game is in quasi-polynomial time [5]. Stochastic versions of the previous games

also exist and are computationally equivalent to SSGs [2]. Interestingly, SSGs have many application domains, for instance autonomous urban driving [9], smart energy management [8], model checking of the modal μ-calculus [23], etc.

There are some restrictions for SSGs for which the problem of finding optimal strategies is tractable. If the game is acyclic, it can be solved in linear time, and in polynomial time for almost acyclic games (few cycles or small feedback arc sets) [3]. If there is no randomness, the game can be solved in almost linear time [1]. Furthermore, Gimbert and Horn were the first to extend this result by giving Fixed Parameter Tractable (FPT) algorithms in the number of random nodes [15]. They indeed show that optimal strategies depend only on the ordering of the values of random nodes, and not on their actual values. Using this idea, they devise two algorithms. The first one exhaustively enumerates these orders until it finds one that actually corresponds to optimal values. The second one is a strategy improvement algorithm based on an iterative refinement of the orders. Both have a complexity of $k!n^{O(1)}$, where k is the number of random nodes. It has been improved to $\sqrt{k!}n^{O(1)}$ expected time in [12], by randomly selecting a good strategy as a starting point for a strategy improvement algorithm. In fact, as remarked in [6], the distance between the values of two consecutive strategies in any strategy improvement algorithm depends on the number of random nodes. Hence any SSG can be solved in time $4^k n^{O(1)}$ (in fact $\sqrt{6}^k n^{O(1)}$ using Lemma 1.1 in [3]). The complexity has been further improved to $2^k n^{O(1)}$ in [19], by using a value iteration algorithm. Here a bit of caution is in order; in some papers, random nodes can have an arbitrary outdegree and probability distribution on outgoing arcs, and in some other they must be binary with uniform distribution. In the former case, if we denote by p the bit-size of the largest probability distribution on a random node, the first two cited algorithms have a complexity of $p \cdot k!$ and $p \cdot \sqrt{k!}$. On the other hand, the two algorithms with an exponential complexity in k have an exponential dependency on p when adapted to this context.

Without the previous restrictions, only algorithms running in exponential time are known. Most of them are strategy improvement algorithms, which produce a sequence of strategies of increasing values. These algorithms, such as the classical Hoffman-Karp [18] algorithm, rely on the switch operation, which by a local best-response, produces a strategy with better value. Several ways of choosing the nodes that are switched have been proposed [24], which can be compared to the rules for pivot selection for the simplex algorithm in linear programming. Though efficient in practice, these algorithms fail to run in polynomial time on a well designed input [14]. The best algorithm so far, proposed by Ludwig [21, 16], is also a strategy iteration algorithm using a randomized version of Bland's rule [4] to choose a switch. It solves any SSG in expected time $2^{O(\sqrt{n})}$. The first analysis of this kind of algorithm is due to Kalai [20] and it has been slightly improved recently [17].

Our contributions

In Section 3, we present an iterative variant of Ludwig's recursive algorithm which uses less random bits. In the rest of the paper we adapt the idea of this algorithm to carefully enumerate orders of random nodes in an SSG. First, in Section 4, we present a pivot operation yielding a strategy improvement algorithm, which improves the one of [15]. This pivot operation comes from a randomized dichotomy on all orders that we explain in details in Section 5, using an auxilliary game similar to the one of [12]. We prove that our algorithm finds the optimal strategies in expected time polynomial in 2^k and p, where k is the number of random nodes and p is the maximum bit-length of a distribution on a random node, answering positively a question of Ibsen-Jensen and Miltersen [19].

2 Definitions and classic results on simple stochastic games

We here review definitions and results related to SSGs. We only sketch what we need and refer to longer expositions such as [11, 24] for more details.

▶ **Definition 1** (SSG). *A simple stochastic game (SSG) is defined by a directed graph $G = (V, A)$, where V is the set of nodes and A the set of arcs, together with a partition of V in four parts V_{MAX}, V_{MIN}, V_{RAN} and V_{SINK}, whose elements are respectively called* MAX*-nodes,* MIN*-nodes,* RAN*-nodes (for random) and sinks. We require that every node $x \in V$ has outdegree at least one, while sink nodes have outdegree exactly* 1 *consisting of a single loop on themselves. We also specify for every sink $x \in V_{\text{SINK}}$ a value $Val(x)$ which is a rational number, and for every random node $x \in V_{\text{RAN}}$ a rational probability distribution $p(x)$ on the outneighbours of x.*

In the original version of Condon [11], all nodes except sinks have outdegree exactly two, the probability distribution on every RAN-node is $(\frac{1}{2}, \frac{1}{2})$, and there are only two sinks, one with value 0 and another with value 1. Here, we allow more than two sinks, with general rational values, and also allow more than outdegree two for all non-sink nodes, with an arbitrary probability distribution for RAN-nodes. However, for Ludwig's Algorithm (see Algorithms 1, 2 and 3 in Section 3) we shall suppose that all MAX-nodes have outdegree 2 and call such games MAX*-binary*.

Strategies and values

We now define strategies, by which we mean stationary and pure strategies. This is enough for our purpose and it turns out to be sufficient for optimality, see [11]. Such strategies specify the choice of a neighbour for every node of a given player.

▶ **Definition 2** (Strategy). *A* strategy *for player* MAX *is a map σ from V_{MAX} to V such that $\forall x \in V_{\text{MAX}}$, $(x, \sigma(x)) \in A$.*

Strategies for player MIN are defined analogously on MIN-nodes and are usually denoted by τ.

▶ **Definition 3** (Play). *A* play *is a sequence of nodes $x_0, x_1, x_2, \dots$ such that for all $t \geq 0$, $(x_t, x_{t+1}) \in A$. Such a play is* consistent *with strategies σ and τ, respectively for player* MAX *and player* MIN*, if for all $t \geq 0$, $x_t \in V_{\text{MAX}} \Rightarrow x_{t+1} = \sigma(x_t)$ and $x_t \in V_{\text{MIN}} \Rightarrow x_{t+1} = \tau(x_t)$.*

A couple of strategies σ, τ and an initial node $x_0 \in V$ define recursively a random play consistent with σ, τ by setting (i) $x_{t+1} = \sigma(x_t)$ if $x_t \in V_{\text{MAX}}$, (ii) $x_{t+1} = \tau(x_t)$ if $x_t \in V_{\text{MIN}}$, (iii) $x_{t+1} = x_t$ if $x_t \in V_{\text{SINK}}$, and finally (iv) x_{t+1} is one of the outneighbours of x_t, randomly chosen independently of everything else according to probability distribution $p(x)$, if $x_t \in V_{\text{RAN}}$.

Hence, this defines a probability measure $\mathbb{P}^{x_0}_{\sigma,\tau}$ on plays consistent with σ, τ. Note that if a play contains a sink node x_s, then at every subsequent time the play stays in x_s. Such a play is said to *reach* sink x_s. To every play $x_0, x_1, \dots$ we associate a value which is the value of the sink reached by the play if any, and 0 otherwise. If we denote by X this value, then X is a random variable once two strategies and an initial node x are fixed. We are interested in the expected value of this quantity, which we call the value of a node $x \in V$ under strategies σ, τ: $\text{Val}_{\sigma,\tau}(x) = \mathbb{E}^x_{\sigma,\tau}(X)$ where $\mathbb{E}^x_{\sigma,\tau}$ is the expected value under probability $\mathbb{P}^x_{\sigma,\tau}$.

The goal of player MAX is to maximize this (expected) value, and the best he can ensure against a strategy τ is $\text{Val}_{*,\tau}(x) := \max_\sigma \text{Val}_{\sigma,\tau}(x)$ where the maximum is considered over all MAX-strategies (which are in finite number). Similarly, against σ player MIN can ensure that the expected value is at most $\text{Val}_{\sigma,*}(x) := \min_\tau \text{Val}_{\sigma,\tau}(x)$.

Finally, the value of a node x is $\mathrm{Val}_{*,*}(x) := \max_\sigma \mathrm{Val}_{\sigma,*}(x) = \min_\tau \mathrm{Val}_{*,\tau}(x)$. The fact that these two quantities are equal is nontrivial, and it can be found for instance in [11]. A pair of strategies σ^*, τ^* such that, for all nodes x, $\mathrm{Val}_{\sigma^*,\tau^*}(x) = \mathrm{Val}_{*,*}(x)$ always exists and these strategies are said to be *optimal strategies*. It is polynomial-time equivalent to compute optimal strategies or to compute the values of all nodes in the game.

▶ **Definition 4** (Stopping SSG). *An SSG is said to be* stopping *if for every couple of strategies almost all plays eventually reach a sink node.*

Usually, this condition is required in order to ensure simple optimality conditions (Thm. 5 below). Condon [11] proved that every SSG G can be reduced in polynomial time to a stopping SSG G' whose size is quadratic in the size of G, and whose values almost remain the same. The values of the new game are close enough to recover the values of the original game. A problem for us is that squaring the size of the game does not behave well relatively to precise complexity bounds.

However, in our case we need a milder condition. We call a MAX-strategy σ stopping if, for any MIN-strategy τ, the random play consistent with (σ, τ) reaches a sink with probability one.

▶ **Theorem 5** (Optimality conditions, [11]). *Let G be an SSG, σ a stopping* MAX*-strategy and τ a* MIN*-strategy. Then (σ, τ) are optimal strategies if and only if*

- *for every $x \in V_{\text{MAX}}$, $Val_{\sigma,\tau}(x) = \max_{(x,y)\in A} Val_{\sigma,\tau}(y)$;*
- *for every $x \in V_{\text{MIN}}$, $Val_{\sigma,\tau}(x) = \min_{(x,y)\in A} Val_{\sigma,\tau}(y)$.*

Switches and strategy improvement

Consider the usual partial order on real vectors indexed by V, i.e. for $w_1, w_2 \in \mathbb{R}^V$, denote $w_1 \leq w_2$ if $w_1(x) \leq w_2(x)$ for all $x \in V$, and denote $w_1 < w_2$ if $w_1 \leq w_2$ and at least one inequality is strict. For two MAX-strategies σ, σ', simply denote $\sigma \leq \sigma'$ (resp. $\sigma < \sigma'$) if $\mathrm{Val}_{\sigma,*} \leq \mathrm{Val}_{\sigma',*}$ (resp. $\mathrm{Val}_{\sigma,*} < \mathrm{Val}_{\sigma',*}$). Define a similar order on MIN-strategies.

A switch, given a strategy, is the fact of changing this strategy at a node (or a set of nodes) in order to obtain a new one.

▶ **Definition 6.** *Let σ, σ' be* MAX*-strategies. We say that σ' is a profitable switch of σ if for all $x \in V_{\text{MAX}}$, one has $Val_{\sigma,*}(\sigma'(x)) \geq Val_{\sigma,*}(\sigma(x))$ with this condition strict for at least one* MAX*-node (such a node is said to be switchable).*

Indeed, the following result states that such a switch actually improves values

▶ **Theorem 7** ([10], [24]). *If σ' is a profitable switch of σ, then $\sigma' > \sigma$.*

Before ending this section, please note that Th. 5 can be restated in terms of nonexistence of switchable node. Hence, we have the following result:

▶ **Theorem 8.** *A stopping* MAX*-strategy is optimal if and only if it has no switchable nodes.*

For the last section, we require another form of switch.

▶ **Theorem 9** ([10], [24]). *Let σ, σ' be stopping* MAX*-strategies and τ, τ' be* MIN*-strategies such that for all $x \in V_{\text{MAX}}$, $Val_{\sigma,\tau}(\sigma'(x)) \geq Val_{\sigma,\tau}(\sigma(x))$ and for all $x \in V_{\text{MIN}}$, $Val_{\sigma,\tau}(\tau'(x)) \geq Val_{\sigma,\tau}(\tau(x))$ with one of these conditions strict for at least one node. Then $Val_{\sigma',\tau'} > Val_{\sigma,\tau}$.*

Orders

For $k \geq 1$ consider the set of integers $[1,k] = \{1, 2, \cdots, k\}$ and let $\mathcal{T}(k)$ denote the set of total orders on $[1,k]$. For sake of clarity we view these orders as sets of couples $(i,j) \in [1,k]^2$ satisfying reflexivity, transitivity and antisymmetry.

If $t \in \mathcal{T}(k)$, it can also be described in *ascending ordering* such as $[x_1, x_2, \ldots, x_k]$ where $(x_i, x_j) \in t$ if and only if $i \leq j$. An *interval* in t is a sequence of consecutive elements in ascending ordering. The rank of an element $x \in [1,k]$ is the number of elements that are lower of equal to x in t, i.e. it is i if $x = x_i$ with notation above.

For lack of a better word, we define a *pretotal order* as an antisymmetric and reflexive relation and denote by $\mathcal{P}(k)$ the set of pretotal orders on $[1,k]$. If $p \in \mathcal{P}(k)$ and $(i,j) \notin p$ is such that $p \cup \{(i,j)\}$ is still antisymmetric, we denote simply by $p + (i,j)$ this new pretotal order.

If $t \in \mathcal{T}(k)$ and $v_1, v_2, \cdots, v_k$ are real numbers, we say that the v_i's are *nondecreasing* along t if $(i,j) \in t \Rightarrow v_i \leq v_j$. Likewise, we say that t is a *nondecreasing order* for $v_1, v_2, \ldots, v_k$.

3 Iterative formulation of Ludwig's algorithm

In this part, we suppose that G is MAX-binary. Hence, if a node x is switchable there is a single possibility for changing the strategy's choice at this node. Let $switch(\sigma, x)$ denote the profitable switch obtained from σ by switching σ at node x.

3.1 Bland's rule version

In [21], Ludwig mentions that his algorithm is a version of Bland's rule, however he does not make it explicit and gives a recursive definition. We formulate his algorithm iteratively (see Algorithm 1), and show that instead of randomly choosing a node at every step, we can choose a total order on nodes prior to the execution of the algorithm. This version uses much less random bits : $O(n \log n)$ bits instead of $2^{O(\sqrt{n})}$ in average in Ludwig's version.

Algorithm 1: Bland's rule formulation for Ludwig's Algorithm.

input : G max-binary SSG, initial stopping MAX-strategy σ.
output : an optimal MAX-strategy

· Pick randomly and uniformly a total order Θ on MAX-nodes
while σ *is not optimal* **do**
  · compute the set of *switchable* nodes for σ
  · let x be the first switchable node in order Θ
  · $\sigma \longleftarrow switch(\sigma, x)$
return σ

By Theorems 7 and 8 if we proceed by switching Strategy σ until there are no more switchable nodes, we reach an optimal strategy in a finite number of steps. The number of steps is at most the number of MAX-strategies, i.e. $2^{|V_{\text{MAX}}|}$. However, we have the following:

▶ **Theorem 10.** *The expected number of strategies considered by Alg. 1 is at most* $e^{2\sqrt{|V_{\text{MAX}}|}}$.

3.2 Analysis of Algorithm 1

Our strategy to prove Theorem 10 is to reformulate Alg. 1 as a recursive algorithm (see Alg. 3), which is close to Ludwig's algorithm in [21]. The proofs of this section will be provided in a long version of this article; they are quite similar to Ludwig's, with a bit of caution on the moments where random choices are made. In particular, we detail our strategy in this part since it will be helpful to understand our results in section 4 where the context is more involved.

Stated as above, it is perhaps unclear how Alg. 1 has a recursive structure. Too see this, consider an execution of Alg. 1, and let x_1 be the last MAX-node in the order Θ. In the beginning, the current strategy σ makes an initial choice $\sigma(x_1)$ on x_1, which does not change until the first time when x_1 becomes switchable (if this happens). If x_1 is switched, then $\sigma(x_1)$ will then remain unchanged until the end of this algorithm. Hence, once Θ is fixed, we can think of this execution as two parts, where $\sigma(x_1)$ is fixed in each part. These can then be decomposed as subparts where $\sigma(x_1)$ and $\sigma(x_2)$ are fixed (where x_2 is the second-to-last MAX-node in order Θ), and so on.

Generalization to partially fixed strategies

To formalize the discussion above, we give a generalization which can be applied to the case where $\sigma(x)$ is fixed for some vertices in a given set F (see Alg. 2).

In the following, if F is a set of MAX-nodes and σ is a MAX-strategy, a (σ, F)-*compatible strategy* is any MAX-strategy σ' such that $\forall x \in F$, $\sigma'(x) = \sigma(x)$. For F and σ fixed, there is always a (σ, F)-strategy that is better than all others. It can be obtained by solving the game where any $x \in F$ is replaced by a random node with a probability 1 to go to $\sigma(x)$. We call such a (σ, F)-compatible strategy *optimal* and we denote it by $\mathrm{opt}(\sigma, F)$. In particular, an optimal $(\sigma, \emptyset)$-strategy is an optimal strategy for G, whereas σ is the only (σ, V_{MAX})-compatible strategy.

Algorithm 2: Iterative formulation for Ludwig's Algorithm with partial strategies.

input : G MAX-binary SSG, total order Θ on V_{MAX}, subset $F \subset V_{\text{MAX}}$, initial MAX-strategy $\sigma = \sigma_0$.

output : a (σ, F)-compatible optimal MAX-strategy $\mathrm{opt}(\sigma, F)$.

while σ *is not an optimal* (σ_0, F)*-compatible strategy* **do**

- compute the set of *switchable* nodes for σ
- let v be the first switchable node in order Θ which is not in F
- $\sigma \longleftarrow \mathit{switch}(\sigma, v)$

return σ

Recursive reformulation

Finally, we give a recursive version of Alg. 2 (see Alg. 3) which we use to derive the bound. The equivalence between these two algorithms should be clear by the previous explanations.

Evaluating the number of switches

Let $f^{\Theta}(\sigma, F)$ be the total number of switches performed by Algorithm 3 on input σ, Θ, F. We consider for the following lemma an execution of this algorithm.

Algorithm 3: Recursive formulation for Ludwig's Algorithm with partial strategies.

input : G MAX-binary SSG, total order Θ on V_{MAX}, subset $F \subset V_{\text{MAX}}$, initial MAX-strategy σ_0.

output : a (σ, F)-compatible optimal MAX-strategy $\text{opt}(\sigma, F)$.

if $F == V_{\text{MAX}}$ **then** return σ
else
  · Let v_0 be the last node not in F according to order Θ
  · Recursively compute σ_1, an optimal $(\sigma_0, F \cup \{v_0\})$-compatible strategy
  if σ_1 *is an optimal* (σ_0, F)*-compatible strategy* **then return** σ_1
  else
    · Let $\sigma_2 \longleftarrow \mathit{switch}(\sigma_1, v_0)$
    · Recursively compute σ^*, an optimal $(\sigma_2, F \cup \{v_0\})$-compatible strategy
    return σ^*

▶ **Lemma 11.** *Let σ_0 be the initial strategy and v_0 be the last node which is not in F, according to order Θ. Define $B \subset V_{\text{MAX}} \setminus F$ to be the set of nodes v such that $\text{opt}(\sigma_0, F \cup \{v\}) \not> \text{opt}(\sigma_0, F \cup \{v_0\})$. Then $f^\Theta(\sigma_0, F) \leq f^\Theta(\sigma_0, F \cup \{v_0\}) + 1 + f^\Theta(\sigma_2, F \cup B\})$, where σ_2 is $\text{opt}(\sigma_0, F \cup \{v_0\}) = \sigma_1$, switched at v_0.*

Now, let us denote $\Phi(n) = \sup_{G,\sigma} \mathbb{E}^\Theta \left[f^\Theta_G(\sigma, \emptyset) \right]$ where the supremum is considered over all SSG G with n MAX-nodes and all MAX-strategies σ in G. The average is considered over all possible prior choices of order Θ, the rest of the algorithm being deterministic.

▶ **Lemma 12.** *For all $n \geq 1$, $\Phi(n) \leq \Phi(n-1) + 1 + \frac{1}{n} \sum_{i=0}^{n-1} \Phi(i)$.*

In order to conclude and prove Theorem 10, we now just have to infer the bound for sequences satisfying the conclusion of Lemma 12.

▶ **Lemma 13** (Lemma 9 of [21])**.** *Let $\Phi(n)$ be such that $\Phi(0) = 0$ and for all $n \geq 1$, $\Phi(n) \leq \Phi(n-1) + 1 + \frac{1}{n} \sum_{i=0}^{n-1} \Phi(i)$. Then for all $n \geq 0$, $\Phi(n) \leq e^{2\sqrt{n}}$.*

4 Simple stochastic games with few random nodes

The idea that in an SSG, the optimal strategies depend only on the ordering of the values of RAN-nodes, and not on their actual values, has been introduced by Gimbert and Horn in [15]. Their main idea is that, if one gives an ordering $r_1 r_2 \cdots r_k$ of RAN-nodes such that $\text{Val}_{*,*}(r_i)$ is nondecreasing with i, then MAX will try to reach a node r_i with i as high as possible, whereas MIN will try to minimize this index; this idea is hereafter formalized by the notion of forcing sets and forcing strategies (Section 4.1). Gimbert and Horn use this fact to derive an algorithm that will enumerate all possible orders on RAN-nodes an will identify one with the property mentionned above, yiedling the optimal strategies and values for G.

The algorithm that we describe and analyse in the rest of this paper (Alg. 4) uses the same principle, but iterates through orders in a special way, similarly to the iteration through strategies made by Ludwig's algorithms (see Section 3). We will derive a similar bound for the average number of iterations of this randomized algorithm. Hence, our main algorithm is still a variation on Bland's rule for pivot selection. The difficulty here does not lie in the proof of the bound, but in the description of the technique used to iterate on orders.

In [15], the game remains the same during the execution of the algorithm, but we proceed differently:

- in Section 4.1, we describe how to associate to every total order $t \in \mathcal{T}(k)$ a new SSG $G[t]$, and we show that this game can be solved in polynomial time.
- in Section 4.2, we prove that there is an optimal order $t^* \in \mathcal{T}(k)$ such that the optimal values of $G[t^*]$ give directly the optimal values of G; it is also the order that maximises values of $G[t]$ among all total orders t. If an order t is not optimal, we describe a *pivot* operation yielding from t a new order t' such that the optimal values of $G[t']$ improve those of $G[t]$.
- the proof of the bound will be derived in Section 5.

4.1 Modified game and forcing strategies

We need to assume that the games we consider enjoy some basic properties in order to describe our algorithm without considering too many special cases.

▶ **Definition 14.** *An SSG is in* canonical form *(CF) if* MAX *has a stopping strategy and only* RAN*-nodes can have an outgoing arc to a sink.*

To ensure these conditions, one can first in linear time find and remove all nodes from which MIN player can force the game never to reach neither a sink node nor a RAN-node (see e.g. [1, 11]). These nodes have value 0 and can as well be removed from the game. Then, all probabilities on RAN-nodes are modified by giving them a very small probability to go to a sink. One can prove as in [11] that values remain almost the same. The second condition ensures that all MAX and MIN nodes have to reach a RAN-node in order to reach a sink. It can be done by adding a dummy random node before every sink.

In all that follows we suppose that G is an SSG in CF with random nodes $r_1, r_2, \ldots, r_k$. Let $t \in \mathcal{T}(k)$ be a total order on $[1, k]$. We define a game $G[t]$ as follows (the same construction is presented in [12]). Start with a copy of G. For every $1 \leq i \leq k$, add a MIN-node denoted i to $G[t]$, which we call *control node*; add an arc (i, r_i); for every arc $(x, r_i) \in A$, remove this arc and add an arc (x, i); finally, for every $(i, j) \in t$, $i \neq j$, add the arc (i, j) to $G[t]$.

So basically, every control node $i \in [1, k]$ intercepts all arcs entering in r_i (see Fig. 1), and has an arc to every other control node $j \in [1, k]$ which is greater than i in t. In the game $G[t]$, the set of sinks, MAX-nodes and RAN-nodes remain the same as in G, whereas the set of MIN-nodes will be denoted $V_{\text{MIN}} \cup [1, k]$, where V_{MIN} is the set of MIN-nodes in G. This allows us to directly identify MAX-strategies in $G[t]$ and in G, and to identify projections onto V_{MIN} of MIN-strategies in $G[t]$, to MIN-strategies in G.

Now, suppose we remove first all sinks and random nodes of $G[t]$, and then turn every control node i into a sink with a value equal to its rank in t. This transformation clearly turns $G[t]$ into a game G' without random nodes.

▶ **Definition 15** (Forcing strategy)**.** *By identifying strategies in $G[t]$ and G', we say that any optimal strategy for* MAX *or* MIN *in G' is a* t-forcing strategy *of $G[t]$.*

In t-forcing strategies, the players try to ensure the reaching of a control node as high as possible for MAX, and as low as possible for MIN, in the order t. We refer to [1] and [15] for more details about how one can compute these optimal strategies in linear time, using the so-called *deterministic attractors*.

▶ **Definition 16** (Forcing set)**.** *For any control node $i \in [1, k]$, define the* forcing set for i*, denoted* $\text{FOR}[t](i)$*, as the set of* MAX *and* MIN*-nodes that reach i if the game is played with a couple (σ_t, τ_t) of t-forcing strategies (forcing sets are independant of the choice of the strategies as long as they are t-forcing).*

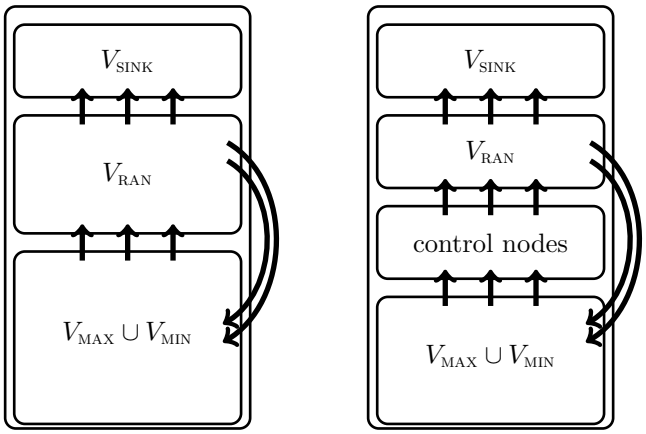
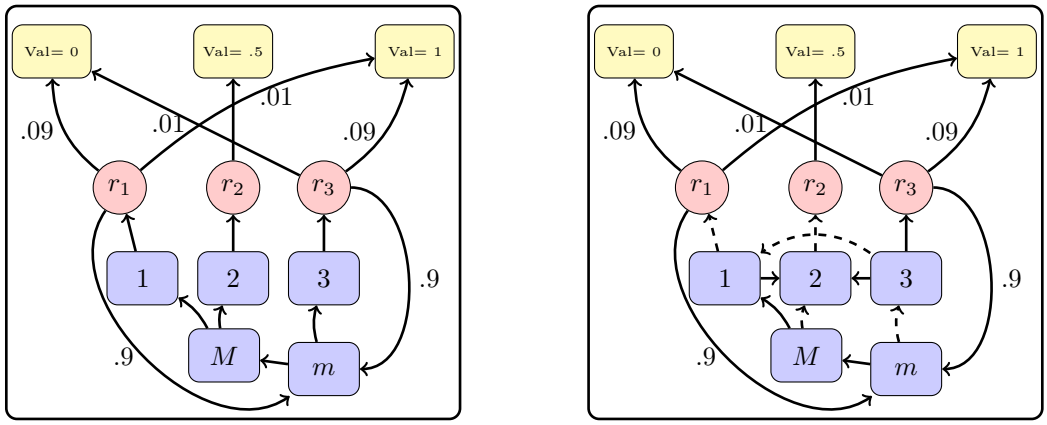


Figure 1 On the left, the structure of a game G in canonical form, only random nodes can directly access sink nodes. On the right, the structure of $G[t]$.

An example of an SSG turned into a modified SSG and of computation of forcing strategies is presented in Fig. 2.



(a) **(b)**

step	total order	forcing strategy for (M, m)	values of RAN-nodes	values of MIN-control nodes
0	$[r_3 r_1 r_2]$	(r_2, r_3)	$.1, .5, .18$	$.1, .5, .1$
1	$[r_1 r_3 r_2]$	(r_2, r_3)	$.46, .5, .54$	$.46, .5, .5$
2	$[r_1 r_2 r_3]$	(r_2, M)	$.46, .5, .54$	$.46, .5, .54$

Figure 2
Fig. (2a): example of an SSG taken from [15] with the additional MIN-control nodes $1, 2, 3$ before each RAN-node. Node m (resp. M) belongs to player MIN(resp. player MAX). Note that we add the dummy RAN-node r_2 so that the game is in CF.
Fig. (2b): solving game $G[t]$ with total order $t = [r_3 r_1 r_2]$. Arcs between control nodes are added according to t. Forcing strategies for m and M, and optimal strategy for nodes m_i are shown with dashed edge. Hence, the forcing set of 1 is $\{1\}$, for 2 it is $\{2, M\}$, and $\{3, m\}$ for 3.
Table: a run of Algorithm 4. The values of the RAN-nodes and the MIN-control nodes are given in the order from left to right.

Here are basic properties on $G[t]$ which should explain why we consider this game.

▶ **Lemma 17.**
(i) *if G has stopping* MAX*-strategy, so does $G[t]$;*
(ii) *optimal values $Val_{*,*}(i)$ of control nodes $i \in [1,k]$ in $G[t]$ are nondecreasing along t;*
(iii) *optimal strategies in $G[t]$ coincide with forcing strategies for order t on $V_{\text{MAX}} \cup V_{\text{MIN}}$;*
(iv) *the game $G[t]$ can be solved in polynomial time.*

Proof. To see why (i) is true, just note that since t is an antisymmetric relation, this does not create new cycles among MIN-nodes.

Suppose now that $(i,j) \in t$. By optimality for the MIN player, and since $G[t]$ is stopping, $\text{Val}_{*,*}(i)$ is the minimum value of $\text{Val}_{*,*}(x)$ for all outneighbours x of i (see Th. 5). Since j is an outneighbour of i in $G[t]$, we have $\text{Val}_{*,*}(i) \leq \text{Val}_{*,*}(j)$. Hence (ii) is true.

Now, consider replacing in $G[t]$ every control node $i \in [1,k]$ by a new sink s_i with value $\text{Val}_{*,*}(i)$. Clearly the values of this new game remain the same. But, by construction of $G[t]$, random nodes have no incoming arcs and they could be as well removed without changing the optimal values on $V_{\text{MAX}} \cup V_{\text{MIN}}$. By reducing the game in this way, we get a deterministic game whose optimal values on $V_{\text{MAX}} \cup V_{\text{MIN}}$ are the same as those of $G[t]$. By definition, optimal strategies of this game are t-forcing strategies, hence (iii) is true.

Finally, to solve $G[t]$ we can choose a couple (σ_t, τ_t) of t-forcing strategies and search for optimal strategies in $G[t]$ that match with (σ_t, τ_t) on $V_{\text{MAX}} \cup V_{\text{MIN}}$. Hence, the strategy of all MAX-nodes is fixed, and only MIN-strategies on control nodes are computed by solving a one player SSG. It can be done in polynomial time by linear programming (see [11]). ◀

As explained in the proof above, to solve $G[t]$, it is enough to compute t-forcing strategies on $V_{\text{MAX}} \cup V_{\text{MIN}}$, which can be done in linear time, and then to solve a one player SSG with only $O(k)$ nodes.

4.2 Value intervals and pivot

In what follows, we write $\text{Val}[t]$ for the vector of optimal values of $G[t]$.

▶ **Definition 18** (Constrained control node)**.** *We say that a control node $i \in [1,k]$ is constrained in $G[t]$ if $Val[t](i) < Val[t](r_i)$.*

Constrained control nodes are similar to switchable nodes in SSG. In fact, we can characterize optimality of an order by the absence of constrained node as follows.

▶ **Lemma 19** (Optimal order)**.** *Let $t \in \mathcal{T}(t)$. The game $G[t]$ does not have any constrained control nodes if and only if the forcing strategies (σ_t, τ_t) are optimal strategies for G. In this case we say that t is an optimal order for G.*

Proof. First note that since G is in CF, σ_t is always stopping.

If $G[t]$ does not have any constrained control nodes, then optimal strategies are the forcing strategies (σ_t, τ_t) on $V_{\text{MAX}} \cup V_{\text{MIN}}$, together with the choice (i, r_i) for each control node $i \in [1,k]$. Then, by merging the control nodes with their associated random node while removing the unused arcs between the control nodes (hence recovering the initial game G), the values on the remaining nodes are kept, and so are the optimality conditions of Th. 5.

If (σ_t, τ_t) are optimal strategies for G, then the values $v_1, v_2, \ldots, v_k$ of the RAN-nodes are nondecreasing along order t. Hence, by turning G into $G[t]$ and extending strategies (σ_t, τ_t) with the choice (i, r_i) for each control node $i \in [1,k]$, we will obtain values that satisfy optimality conditions and such that $Val[t](i) = Val[t](r_i)$, showing that i is not constrained. ◀

We define the *value interval* of a control node $i \in [1,k]$ as the set of $j \in [1,k]$ that share the same optimal value in $G[t]$, i.e. $\mathrm{Val}[t](i) = \mathrm{Val}[t](j)$. This set is indeed an interval in order t by (ii) of Lemma 17, i.e. its elements are consecutive in order t.

▶ **Definition 20.** *The pivot operation on a control node $i \in [1,k]$ for the order t is the transformation of t into a new order $t' \in \mathcal{T}(k)$, obtained by moving i just after the end of its value interval in t.*

Note that if i is the last node of its value interval, then the pivot operation does nothing. Also note that if i is constrained, it cannot be the last node of its value interval (we shall only pivot on constrained control nodes).

Example. Let $k = 7$ and let t be in ascending order $[7,2,4,1,3,6,5]$. Suppose that the values of control nodes are, in this order $[0.2, 0.2, 0.3, 0.3, 0.3, 0.4, 0.4]$. The value intervals are $[7,2]$, $[4,1,3]$ and $[6,5]$. The pivot operation on 4 places 4 after 3, so that the obtained order would be $[7,2,1,3,4,6,5]$.

The following theorem shows that the pivot operation increases the value vector, which will enable us to design a strategy improvement algorithm on the forcing strategies (where the improvement is on $\mathrm{Val}[t]$ rather than on values in the original game G). A similar theorem is proved in [12] to build a different strategy improvement algorithm.

▶ **Theorem 21.** *Let $t \in \mathcal{T}(t)$ and $i \in [1,k]$ be a constrained control node. If t' is obtained from $t \in \mathcal{T}(k)$ by pivoting on i, then $Val[t'] > Val[t]$.*

Proof. Consider a new game $G[t+t']$ which is obtained from G like $G[t]$ and $G[t']$ but with arcs (i,j) for $i \neq j$ between control nodes for all $(i,j) \in t \cup t'$. Let (σ,τ) and (σ',τ') be respective optimal strategies in $G[t]$ and $G[t']$. We can interpret these strategies as strategies in $G[t+t']$. Since the only difference between $G[t]$, $G[t']$ and $G[t+t']$ are the arcs between control nodes, all strategies (σ,τ) give exactly the same values in $G[t]$ and in $G[t+t']$, and a similar observation can be made for $G[t']$. Hence, to prove the result, is is enough to show that $\mathrm{Val}_{\sigma',\tau'} > \mathrm{Val}_{\sigma,\tau}$ in $G[t+t']$. Note that whereas σ and σ' are respective stopping MAX-strategies of $G[t]$ and $G[t']$, they could be not stopping in $G[t+t']$. However it is not difficult to see that conclusion of Th. 9 would still apply. Hence it is sufficient to show in $G[t+t']$ that changing (σ,τ) into (σ',τ') makes a nondecreasing switch on every node, and is increasing in at least one node.

In the order t, let $I = [i, i_1, i_2, \cdots, i_\ell]$, with $\ell \geq 1$ be the increasing sequence of consecutive nodes sharing the same value as i for (σ,τ) (i.e. the value interval of i, starting from i). Since i is constrained, $\ell \geq 1$.

The pivot operation transforms this part of t into $[i_1, i_2, \cdots, i_\ell, i]$, hence the only differences between $G[t]$ and $G[t']$ are the ℓ arcs (i, i_c) for $1 \leq c \leq \ell$ that are inverted into (i_c, i). Hence, if $j \notin I$, it keeps the same position relatively to all other control nodes when we change the order t into t', hence $\textsc{For}[t](j) = \textsc{For}[t'](j)$. Hence, when we change (σ,τ) to (σ',τ'), either there is no switch in $\textsc{For}[t](j)$, or it is between nodes of the same values.

For nodes $x \in \textsc{For}[t](j)$ with $j \in I$, clearly $\sigma'(x)$ (resp. $\tau'(x)$) is in some $\textsc{For}[t](j')$ with $j' \in I$. Since all these nodes also share the same value (by definition of the value interval), these switches are also between nodes of same values.

Suppose that there is a decreasing switch on a control node, i.e. for a $j \in [1,k]$ we have $\mathrm{Val}[t](\tau'(j)) < \mathrm{Val}[t](j)$. In this case $\tau'(j)$ should be stricly before j in t since optimal values are increasing along t. So we could not have $(j, \sigma'(j)) \in t$ but should have $(j, \sigma'(j)) \in t'$. The only possibility is $\sigma'(j) = i$ and $j \in I$. Since these nodes are in the same value interval, once again this switch is unchanging, a contradiction.

We showed that no switch from (σ, τ) to (σ', τ') is decreasing. Now consider the case of i during the pivot operation. Since i is constrained, $\mathrm{Val}[t](i) < \mathrm{Val}[t](r_i)$. Since $\tau'(i)$ can either be equal to r_i or to some j which is stricly after the value interval of i in order t, hence has a greater value, we see that the switch at i must be increasing. ◀

4.3 Main algorithm

Algorithm 4 consists in iterating on orders $t \in \mathcal{T}(k)$, by picking randomly a pivotable element in t and updating t by a pivot on i, until we reach an optimal order.

Here is the **pivot selection rule**. First, prior to the execution of the algorithm, we choose randomly and uniformly an order Θ on the set of all $\frac{k(k-1)}{2}$ unordered pairs of control nodes $\{i, j\}$, with $i, j \in [1, k]$. Then, at each step of the algorithm, consider the game $G[t]$, and remove one by one the arcs between control nodes, following order Θ. During this process, choose as pivot the first constrained control node, if any, which is disconnected from the following nodes of its value interval. In more detail, for a given order t, compute $\mathrm{Val}[t]$ and then partition the control nodes into value intervals. Each constrained control node i has $d(i)$ arcs leading to other control nodes from the same value interval, where $d(i)$ is its distance in t to the last element of this interval. Enumerating Θ in ascending order, the pivot is the first constrained node i whose $d(i)$ arcs are encountered.

Example. Continued from the previous example with $k = 7$ and value intervals $[7, 2]$, $[4, 1, 3]$ and $[6, 5]$. Suppose that the order Θ starts $\{2, 5\}, \{7, 6\}, \{1, 4\}, \{2, 7\} \ldots$. The first element that is disconnected from its value interval is 7 which is the one we choose as a pivot leading to order $[2, 7, 4, 1, 3, 6, 5]$.

Algorithm 4: Iterative version for Bland's rule on random nodes.

input : G SSG, initial total order $t \in \mathcal{T}(k)$.
output : optimal MAX and MIN-strategies.

· pick randomly and uniformly a total order Θ on all pairs of control nodes $\{i, j\}$
· compute values in $G[t]$ (poly. time, see Lemma 17)
while *t is not optimal (Lemma 19)* **do**
  · choose a pivot i by the pivot selection rule
  · update t by pivoting on i as in Def. 20
  · compute values in $G[t]$ (poly. time, see Lemma 17)
return *a couple of forcing strategies for order t in G.*

By Th. 21, no order $t \in \mathcal{T}(k)$ is repeated during the execution of Algorithm 4; since $\mathcal{T}(k)$ is finite, the algorithm reaches in a finite number of steps an order $t^* \in \mathcal{T}(k)$ which has no constrained node, i.e. which is optimal by Lemma 19. Hence, Algorithm 4 computes optimal strategies for G in at most $k!$ steps. However, we claim the following result, which will be proved in the next section.

▶ **Theorem 22.** *Alg. 4 computes optimal strategies for G in at most $e^{\sqrt{2 \cdot k}}$ expected steps.*

Note that for k large enough we have $e^{\sqrt{2 \cdot k}} < k!$, whose growth is roughly equivalent to $2^{k \log k}$. Moreover, the algorithm uses $O(k^2 \log k)$ random bits to choose the order Θ on pairs.

5 Analysis of Algorithm 4

In this section we prove Theorem 22. To do this we shall reformulate Algorithm 4 as a recursive algorithm, but we need additional notions for this. The recursive formulation also

reveals the nature of the algorithm: it computes an optimal order on control nodes by finding the right order between each pair of these nodes using dichotomy. This allows the same analysis as for Ludwig's Algorithm and its variants.

5.1 Modified game $G[p]$ for a pretotal order p

If $p \in \mathcal{P}(k)$ is a pretotal order, we define $G[p]$ exactly as was defined $G[t]$ for a total order $t \in \mathcal{T}(k)$ in Section 4.1. The only difference is that, since p is not total, a control node $i \in [1,k]$ only has arcs to those $j \neq i \in [1,k]$ such that $(i,j) \in p$.

To simplify notation, for any node x in $G[p]$, define $\mathrm{Val}_*[p](x) := \mathrm{Val}^{G[p]}_{*,*}(x)$ as the optimal value of x in $G[p]$. We can now directly extend some of the observations of Lemma 17 to pretotal orders.

▶ **Lemma 23.** *If $p \in \mathcal{P}(k)$, then optimal values of control nodes $i \in [1,k]$ in $G[p]$ are nondecreasing in order p, i.e. if $(i,j) \in p$ then $Val_*[p](i) \leq Val_*[p](j)$.*

In order to solve $G[p]$, the algorithm will recursively compute an optimal total ordering of control nodes $i \in [1,k]$ extending p. Thus, for all total orders $t \in \mathcal{T}(k)$ extending $p \in \mathcal{P}(k)$, we need to assign a value in $G[p]$, which we denote $\mathrm{Val}[p](t)$. Here is how we define it.

▶ **Definition 24.** *Let $t \in \mathcal{T}(k)$ extending $p \in \mathcal{P}(k)$. The values $Val[p](t)$ associated to t in $G[p]$ are the values Val_{σ_t,τ_t} where σ_t and τ_t satisfy:*

(i) *σ_t and τ_t are forcing strategies for $G[t]$;*

(ii) *τ_t statisfies the* MIN*-optimality conditions (Thm. 5) on every control node $i \in [1,k]$.*

As a summary, $\mathrm{Val}_*[p]$ is the vector of optimal values of game $G[p]$ while $\mathrm{Val}[p](t)$ is the vector of optimal values of $G[p]$ when the strategies in V_{MIN} and V_{MAX} are forcing strategies. It follows that $\mathrm{Val}[p](t) \leq \mathrm{Val}_*[p]$. Recall that $\mathrm{Val}[t]$ is the vector of optimal values of $G[t]$. Then we have $\mathrm{Val}[t] = \mathrm{Val}_*[t] = \mathrm{Val}[t](t)$.

▶ **Definition 25** (Optimal order)**.** *Let $p \in \mathcal{P}(k)$ and $t \in \mathcal{T}(k)$ extending p ($p \subset t$). We say that t is an optimal total order for p if $Val_*[p] = Val[p](t)$.*

The next lemma proves the existence and gives a characterization of optimal orders.

▶ **Lemma 26.** *Suppose G is in CF and let $p \in \mathcal{P}(k), t \in \mathcal{T}(k), p \subset t$. Then the following conditions are equivalent:*

(i) *t is an optimal order for p;*

(ii) *t is a nondecreasing ordering of the values $Val_*[p](i)$ for $i \in [1,k]$;*

(iii) *$Val_*[p] = Val[t]$.*

Proof of Lemma 26. First, note that, by definition, optimality conditions are satisfied at control nodes in the definition of $\mathrm{Val}[p](t)$, so it is always true that values $\mathrm{Val}[p](t)(i)$ are nondecreasing along p.

Suppose now that t is optimal for p, i.e. $\mathrm{Val}_*[p] = \mathrm{Val}[p](t)$, and suppose that t is not a nondecreasing ordering of the values $\mathrm{Val}_*[p](i)$ for $i \in [1,k]$. Then there must be two consecutive i, j in order t such that $\mathrm{Val}_*[p](i) > \mathrm{Val}_*[p](j)$, and we must have $(i,j) \in t \setminus p$. Consider the order t' that we obtain from t by inverting j and i. Clearly, this order also extends p since the only inversion between t and t' is (i,j). By an argument similar to the proof of Theorem 21, it is easy to obtain that $\mathrm{Val}[p](t') > \mathrm{Val}[p](t)$, which contradicts optimality. Hence we proved $(i) \Rightarrow (ii)$.

Now suppose (ii). Let (σ^*, τ^*) be an optimal strategy of $G[p]$; we show that optimality conditions are met in $G[t]$. First note that (σ^*, τ^*) has the same values in $G[p]$ and $G[t]$. For the nodes in $V_{\text{MAX}} \cup V_{\text{MIN}}$, no new arcs are added so the optimality conditions are still

satisfied. Now, consider control nodes. An arc $(i,j) \in t \backslash p$ cannot lead to a lower value for i by assumption (ii). Hence the optimality conditions are still satisfied on control nodes, strategies (σ^*, τ^*) are optimal for $G[t]$, so finally $\mathrm{Val}_*[p] = \mathrm{Val}[t]$ and we proved $(ii) \Rightarrow (iii)$.

Since t involves more arcs than p between the control MIN-nodes, we have $\mathrm{Val}[t] \leq \mathrm{Val}[p](t) \leq \mathrm{Val}_*[p]$. Assume (iii), then $\mathrm{Val}_*[p] = \mathrm{Val}[p](t)$. Hence we proved $(iii) \Rightarrow (i)$. ◀

5.2 Recursive formulation

We now give Algorithm 5, a recursive formulation of Algorithm 4. We will prove that these two algorithms compute exactly the same sequence of total orders, and use the recursive formulation to derive a bound.

Algorithm 5: Recursive version for Bland's rule on random nodes.

input : G SSG, order Θ on all pairs $\{i,j\}$ with $i \neq j \in [1,k]$, initial total order t_0 in $\mathcal{T}(k)$ extending a pretotal order $p_0 \in \mathcal{P}(k)$.

output : optimal total order of $G[p_0]$

if p_0 *is total (i.e.* $p_0 = t_0$*)* **then return** t_0

else

- · select according to Θ the last pair $\{i,j\}$ s.t. $(i,j) \in t_0 \setminus p_0$
- · let $p_1 = p_0 + (i,j)$ and $p_2 = p_0 + (j,i)$
- · recursively solve $G[p_1]$ with initial order t_0, giving optimal total order t_1 for $G[p_1]$
- **if** t_1 *is optimal for* $G[p_0]$ *(apply criterium in Lemma 26)* **then return** t_1
- **else**
  - · let t_2 be the total order obtained by pivoting in t_1 along i
  - · recursively solve $G[p_2]$ with initial order t_2, giving optimal order t^* for $G[p_1]$
  - **return** t^*.

▶ **Definition 27.** *Let $p_0 \in \mathcal{P}(k)$ and $\{i,j\} \notin p_0$ such that $p_1 = p_0 + (i,j)$ is still a pretotal order. We say that the addition of (i,j) to p_0 is constraining, or that (i,j) is constrained, if $Val_*[p_1] < Val_*[p_0]$.*

When an arc is constrained, it is essential to the MIN-optimal strategy in $G[p_1]$; in other words removing this arc would increase optimal values.

▶ **Lemma 28.** *Suppose G is in CF and let $p \in \mathcal{P}(k)$, $t \in \mathcal{T}(k)$, $p \subset t$. Then the following conditions are equivalent:*

(i) *t is an optimal ordering for p;*

(ii) *the addition of every arc $(i,j) \in t \setminus p$ to p is not constraining;*

Proof of Lemma 28. Let $p_1 = p_0 + (i,j)$. Since $p \subset p_1 \subset t$, we have $\mathrm{Val}[t] \leq \mathrm{Val}_*[p_1] \leq \mathrm{Val}_*[p]$.

Assume that t is an optimal order for p. If the addition of (i,j) to p is constraining, then $\mathrm{Val}_*[t] \leq \mathrm{Val}_*[p_1] < \mathrm{Val}_*[p]$ which contradicts $\mathrm{Val}[t] = \mathrm{Val}_*[p]$ by Lemma 26. Hence we proved $(i) \Rightarrow (ii)$.

Assume that no arc in $t \backslash p$ is constraining. Then, add sequentially arcs in $t \backslash p$ to $G[p]$ until we get $G[t]$, hence forming a sequence of games $G[p] = G[p_0], G[p_1], \ldots, G[t]$. If none of these arcs is used in $G[t]$ then $\mathrm{Val}[t] = \mathrm{Val}_*[p]$ and (i) is proved. Otherwise consider the first

arc (i,j) such that $\mathrm{Val}_*[p_\ell + (i,j)] < \mathrm{Val}_*[p_\ell]$. This implies that $\mathrm{Val}_*[p_\ell](j) < \mathrm{Val}_*[p_\ell](i)$. But $\mathrm{Val}_*[p] = \mathrm{Val}_*[p_\ell]$ since no constraining arc has been added until step ℓ. Hence $\mathrm{Val}_*[p](j) < \mathrm{Val}_*[p](i)$, and finally (i,j) is constraining for p, a contradiction. ◀

Consider a run of the recursive algorithm and let t be a total order at any step of the run. Let us inspect the first time where t is modified. Order t will be optimal for a sequence of pretotal orders that are obtained from t by removing one by one pairs in order Θ as long as they are not constrained. This in fact amounts to ascending the recursive call tree. Let (i,j) be the first constrained pair and p_0 the pretotal order obtained once (i,j) is removed. Then t is turned into t' by pivoting on node i as we did in iterative Alg. 4. We show now that control node i is same as the pivot selected by the pivot selection rule.

Note first that, during the process of removing the pairs one by one in order Θ, the value intervals of $G[t]$ are kept unchanged until the pretotal order $p_0 + (i,j)$ is reached. Since removing (i,j) implies an increase of the optimal values, it means that i and j were in the same value interval and that i had no other neighbour in that interval. Note here that, as a consequence, $p_0 + (j,i)$ is then guaranteed to be a pretotal order. Clearly, node i is the first control node in that situation. So the choice of node i exactly obeys the pivot selection rule.

Finally, the following lemma enables us to analyze the complexity of Algorithm 5.

▶ **Lemma 29.** *Let $p_0 \in \mathcal{P}(k)$ and $\{i,j\} \notin p_0$ such that $p_1 = p_0 + (i,j)$ and $p_2 = p_0 + (j,i)$ are pretotal orders and where the addition of (i,j) to p_0 is constraining. Let t_1 be an optimal total order for p_1, t_2 obtained from t_1 by pivoting in i, and let t_2^* be an optimal total order for p_2.*

Let (i_1,j_1) such that $Val_*[p_0 + (i,j)] \not\leq Val_*[p_0 + (i_1,j_1)]$. Then for any total order t obtained by Algorithm 5 between t_1 and t_2^* (including those), one has $(j_1,i_1) \in t$.

Proof. Suppose that $(i_1,j_1) \in t$. Then $t \supset p_0 + (i_1,j_1)$ hence $\mathrm{Val}[t] \leq \mathrm{Val}_*[p_0 + (i_1,j_1)]$.

On the other hand, since the pivot operation is increasing values, we have $\mathrm{Val}_*[p_0+(i,j)] = \mathrm{Val}[t_1] < \mathrm{Val}[t_2] \leq \mathrm{Val}[t]$, so $\mathrm{Val}_*[p_0 + (i,j)] \leq \mathrm{Val}_*[p_0 + (i_1,j_1)]$, a contradiction. ◀

Using this result, the proof for the complexity bound is the same as the proof of Theorem 10 using the recursive formulation. Let $f^\Theta(G, p_0, t_0)$ be the total number of pivots performed by Algorithm 5 on input G, p_0, t_0 for an order Θ on pairs.

Now define $\Phi(m) = \sup_{G,p_0,t_0} \mathbb{E}^\Theta\left[f^\Theta(G,p_0,t_0]\right)$ where the supremum is taken over all games G, pretotal orders p_0 and total orders t_0 extending p_0 such that $t_0 \setminus p_0$ is of size at most m. The expectation is taken over all possible uniform choices for Θ.

Then by Lemma 29, $\Phi(m)$ will satisfy Lemma 12, hence the claimed bound of Th. 22 by Lemma 13 since the depth of the recursive tree is at most $\frac{k(k-1)}{2}$.

References

1. Daniel Andersson, Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen. Deterministic graphical games revisited. In *Conference on Computability in Europe*, pages 1–10. Springer, 2008.
2. Daniel Andersson and Peter Bro Miltersen. The complexity of solving stochastic games on graphs. In *International Symposium on Algorithms and Computation*, pages 112–121. Springer, 2009.
3. 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.
4. Robert G Bland. New finite pivoting rules for the simplex method. *Mathematics of operations Research*, 2(2):103–107, 1977.

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*, pages 252–263. ACM, 2017.

6 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A Henzinger. Termination criteria for solving concurrent safety and reachability games. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 197–206. SIAM, 2009.

7 Krishnendu Chatterjee and Nathanaël Fijalkow. A reduction from parity games to simple stochastic games. In *GandALF*, pages 74–86, 2011.

8 Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

9 Taolue Chen, Marta Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *International Conference on Quantitative Evaluation of Systems*, pages 322–337. Springer, 2013.

10 Anne Condon. On Algorithms for Simple Stochastic Games. In *Advances in computational complexity theory*, pages 51–72, 1990.

11 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

12 Decheng Dai and Rong Ge. New results on simple stochastic games. In *International Symposium on Algorithms and Computation*, pages 1014–1023. Springer, 2009.

13 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

14 Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on*, pages 145–156. IEEE, 2009.

15 Hugo Gimbert and Florian Horn. Simple stochastic games with few random vertices are easy to solve. In *Foundations of Software Science and Computational Structures*, pages 5–19. Springer, 2008.

16 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.

17 Thomas Dueholm Hansen and Uri Zwick. An improved version of the Random-Facet pivoting rule for the simplex algorithm. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 209–218. ACM, 2015.

18 Alan J Hoffman and Richard M Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

19 Rasmus Ibsen-Jensen and Peter Bro Miltersen. Solving simple stochastic games with few coin toss positions. In *European Symposium on Algorithms*, pages 636–647. Springer, 2012.

20 Gil Kalai. A subexponential randomized simplex algorithm. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 475–482. ACM, 1992.

21 Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and computation*, 117(1):151–155, 1995.

22 Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

23 Colin Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of IGPL*, 7(1):103–124, 1999.

24 Rahul Tripathi, Elena Valkanova, and VS Anil Kumar. On strategy improvement algorithms for simple stochastic games. *Journal of Discrete Algorithms*, 9(3):263–278, 2011.

Distributed Coloring of Graphs with an Optimal Number of Colors

Étienne Bamas
School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, Switzerland
etienne.bamas@epfl.ch

Louis Esperet
Laboratoire G-SCOP (CNRS, Univ. Grenoble Alpes), Grenoble, France
louis.esperet@grenoble-inp.fr

Abstract

This paper studies sufficient conditions to obtain efficient distributed algorithms coloring graphs optimally (i.e. with the minimum number of colors) in the LOCAL model of computation. Most of the work on distributed vertex coloring so far has focused on coloring graphs of maximum degree Δ with at most $\Delta+1$ colors (or Δ colors when some simple obstructions are forbidden). When Δ is sufficiently large and $c \geq \Delta - k_\Delta + 1$, for some integer $k_\Delta \approx \sqrt{\Delta} - 2$, we give a distributed algorithm that given a c-colorable graph G of maximum degree Δ, finds a c-coloring of G in $\min\{O((\log\Delta)^{13/12}\log n), 2^{O(\log\Delta+\sqrt{\log\log n})}\}$ rounds, with high probability. The lower bound $\Delta - k_\Delta + 1$ is best possible in the sense that for infinitely many values of Δ, we prove that when $\chi(G) \leq \Delta - k_\Delta$, finding an optimal coloring of G requires $\Omega(n)$ rounds. Our proof is a light adaptation of a remarkable result of Molloy and Reed, who proved that for Δ large enough, for any $c \geq \Delta - k_\Delta$ deciding whether $\chi(G) \leq c$ is in P, while Embden-Weinert *et al.* proved that for $c \leq \Delta - k_\Delta - 1$, the same problem is NP-complete. Note that the sequential and distributed thresholds differ by one.

Our first result covers the case where the chromatic number of the graph ranges between $\Delta - \sqrt{\Delta}$ and $\Delta + 1$. Our second result covers a larger range, but gives a weaker bound on the number of colors: For any sufficiently large Δ, and $\Omega(\log\Delta) \leq k \leq \Delta/100$, we prove that every graph of maximum degree Δ and clique number at most $\Delta - k$ can be efficiently colored with at most $\Delta - \varepsilon k$ colors, for some absolute constant $\varepsilon > 0$, with a randomized algorithm running in $O(\log n/\log\log n)$ rounds with high probability.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Mathematics of computing → Graph algorithms; Theory of computation → Distributed algorithms

Keywords and phrases Graph coloring, distributed algorithm, maximum degree

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.10

Related Version The full version of the paper [1] is available at `arXiv:1809.08140`.

Funding *Louis Esperet*: Partially supported by ANR Project GATO (ANR-16-CE40-0009-01) and LabEx PERSYVAL-Lab (ANR-11-LABX-0025).

Acknowledgements We thank David Harris for pointing out the updated version of [11] and for his kind remarks on earlier versions of the paper. We also thank two anonymous reviewers for their detailed comments and suggestions.

1 Introduction

The graph coloring problem plays an important role in distributed computing, since it is used as a subroutine in distributed algorithms for a large variety of problems (see the recent survey book of Barenboim and Elkin [2] for more details and further references). The central problem in distributed coloring is the $(\Delta+1)$-coloring problem, where a graph of maximum

degree at most Δ has to be colored with at most $\Delta + 1$ colors (see [10] and [5] for the fastest deterministic and randomized algorithms to date and more on the history of the problem). The bound $\Delta + 1$ on the number of colors is best possible in general, but it follows from Brooks' Theorem that any connected graph of maximum degree Δ which is neither an odd cycle nor a complete graph can indeed be colored with Δ colors, instead of $\Delta + 1$, and there has been some work to find fast distributed algorithms coloring such graphs with Δ colors. The problem was first considered by Panconesi and Srinivasan [18], and it was recently proved in [12] that the Δ-coloring problem can be solved with a randomized algorithm running in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds when $\Delta \geq 4$, or $O((\log \log n)^2)$ rounds when Δ is a constant. On the other hand, it was proved in [4] that a randomized algorithm solving the Δ-coloring problem needs $\Omega(\log \log n)$ rounds. These results, as well as all the other algorithms mentioned in this paper, are proved in the LOCAL model of computation (see below for more details).

The main idea of Δ-coloring is that by forbidding some simple obstructions (complete graphs and odd cycles), we can save one color (compared with the easier $(\Delta + 1)$-coloring problem) while still having a fast algorithm, whether sequential or distributed. A natural question is: can we go further? Is there some small set of obstructions (that can be easily recognized locally, at least when Δ is sufficiently large), such that if we forbid these obstructions we can find fast distributed algorithms coloring graphs of maximum degree Δ with $\Delta - 1$ colors? Or $\Delta - 2$ colors? Or $\Delta - k$ colors, for some constant k?

The sequential version of this question turned out to have a very precise answer. For any Δ, let k_Δ be the maximum integer k such that $(k + 1)(k + 2) \leq \Delta$. It can be checked that $k_\Delta = \lfloor \sqrt{\Delta + 1/4} - 3/2 \rfloor$ and thus $\sqrt{\Delta} - 3 < k_\Delta < \sqrt{\Delta} - 1$. The following was proved by Embden-Weinert, Hougardy and Kreuter [8].

▶ **Theorem 1.1** ([8]). *For $3 \leq c \leq \Delta - k_\Delta - 1$, we cannot test for c-colorability of graphs with maximum degree Δ in polynomial time unless $P = NP$.*

The following strong converse was then proved by Molloy and Reed [17].

▶ **Theorem 1.2** ([17]). *For sufficiently large (but constant) Δ, and every $c \geq \Delta - k_\Delta$, there is a linear time deterministic algorithm to test whether graphs of maximum degree Δ are c-colorable. Furthermore, there is a polynomial time deterministic algorithm that will produce a c-coloring whenever one exists.*

Our main result will be to prove that a similar dichotomy occurs in the LOCAL model, with a slightly larger tractability threshold ($\Delta - k_\Delta + 1$ instead of $\Delta - k_\Delta$).

▶ **Theorem 1.3.** *For sufficiently large Δ, and any $c \geq \Delta - k_\Delta + 1$, there is a distributed randomized algorithm running w.h.p. in $\min\{O((\log \Delta)^{13/12} \log n), 2^{O(\log \Delta + \sqrt{\log \log n})}\}$ rounds, that takes a graph G with maximum degree Δ as input, and does the following: either some vertex outputs a certificate that G is not c-colorable, or the algorithm finds a c-coloring of G.*

Here, w.h.p. (with high probability) means with probability at least $1 - O(n^{-\alpha})$, for any fixed $\alpha > 0$. Note that the chromatic number of G can be smaller than the threshold $\Delta - k_\Delta + 1$, what matters is that the number c of available colors is at least this threshold. We will prove that the value of $\Delta - k_\Delta + 1$ is sharp, in the following sense.

▶ **Theorem 1.4.** *When $c \leq \Delta - k_\Delta - 1$ (for any value of Δ), and when $c = \Delta - k_\Delta$ (for infinitely many values of Δ), there exist arbitrarily large graphs G of maximum degree Δ for which $\chi(G) = c$, and such that any distributed algorithm coloring G with c colors takes $\Omega(n/\Delta)$ rounds.*

In the LOCAL model of computation, if the algorithm runs in r rounds, the color assigned to a vertex v is based only on the (subgraph induced by the) vertices at distance at most r from v. The fact that when $c \geq \Delta - k_\Delta + 1$, it can be decided whether G is c-colorable by only looking at each neighborhood was already proved by Molloy and Reed [17] (see Theorem 4.1). In this paper, we are mostly interested in *producing* such a coloring in a distributed way, and it is a priori unclear that it can be done in a small number rounds. For instance, in the LOCAL model it can be decided in a single round whether a graph has maximum degree at most two (and is therefore 3-colorable), but finding a 3-coloring of a path takes an unbounded number of rounds [15].

An interesting difference between Theorems 1.3 and 1.2 (besides the fact that the sequential and distributed thresholds are not the same), is that in the sequential result it is important that Δ is a constant. If Δ depends on n, then Molloy and Reed [17] proved that the tractability threshold is around $\Delta - \Theta(\log \Delta)$ colors. On the other hand, in the distributed setting there is no requirement on Δ.

It should be mentioned that efficient distributed coloring algorithms involving the chromatic number are not frequent. A rare example of such an algorithm involving a general class of graphs (not just paths or cycles, or line-graphs for instance) is the following result of Schneider and Wattenhofer [20]: when $\Delta = \Omega(\log^{1+1/\log^* n} n)$ and $\chi = O(\Delta/\log^{1+1/\log^* n} n)$, they find a randomized distributed algorithm coloring graphs of maximum degree Δ and chromatic number χ with at most $(1 - 1/O(\chi))\Delta$ colors w.h.p., and running w.h.p. in $O(\log \chi + \log^* n)$ rounds. Two significant differences with our result are the requirement on Δ and the fact that the number of colors in the resulting coloring is not best possible. We also note that in the setting of Theorem 1.2 and Theorem 1.5 below, the chromatic number is an additive factor away from the maximum degree, while the result of Schneider and Wattenhofer [20] mentioned above asks for a much larger (multiplicative) gap between χ and Δ.

Theorem 1.3 covers in particular the situation where $\chi(G) \geq \Delta - \sqrt{\Delta} + 1$ (and in this case, gives an efficient algorithm to obtain an optimal coloring of the graph). Recall that Brooks' theorem (and its algorithmic variants) colors graphs of maximum degree $\Delta \geq 3$ distinct from $K_{\Delta+1}$ (or equivalently, with clique number at most Δ) with at most Δ colors. Our next result generalizes the algorithmic versions of Brooks' theorem in the following direction.

▶ **Theorem 1.5.** *There exists $\Delta_0 > 0$ such that for every $\Delta \geq \Delta_0$ and $2^{59} \log \Delta \leq k \leq \frac{\Delta}{100}$, there exists a randomized distributed algorithm that given an n-vertex graph of maximum degree Δ, does the following: either some vertex outputs a clique of size more than $\Delta - k$ if such a clique exists, or the algorithm finds a coloring with at most $\Delta - 2^{-23}k$ colors. The round complexity is the minimum of $O(\log_\Delta n + \log_k \Delta) + 2^{O(\sqrt{\log \log n})}$ and $2^{O(\log \Delta + \sqrt{\log \log n})}$ w.h.p., and in particular it is $O(\log n / \log \log n)$ w.h.p.*

We start with some preliminaries on distributed computing, probability, and graph theory in Section 2. We then prove Theorem 1.5 in Section 3. It turns out that the proof of Theorem 1.5 contains several ingredients that will be reused in the proof of Theorem 1.3. In Section 4, we prove Theorem 1.4 and explain how to adapt the proof of Theorem 1.2 in [17] to prove Theorem 1.3. We conclude with some remarks in Section 5.

2 Preliminaries

2.1 Distributed computing

We consider the classical LOCAL model of computation, which is a distributed model in which the network corresponds to the graph under consideration, i.e. each vertex of the graph corresponds to a processor, with infinite computational power, and vertices can communicate with their neighbors in synchronous rounds (in this model there is no restriction on the size of the messages exchanged by two neighboring vertices during each round of communication). Each vertex knows the number n of vertices and its own id (a distinct integer between 1 and n). In this paper, the vertices also know the maximum degree Δ of the graph, and some number c of colors. Once the communication between the nodes is over, each vertex outputs a value (in our case, an integer between 1 and c corresponding to its color in a proper coloring of the graph, or some subset of its neighbors which cannot be colored with c colors). The complexity of the algorithm is the number of rounds of communication.

2.2 Vertex coloring

A *c-coloring* of a graph G is an assignment of integers from $\{1, \ldots, c\}$ to the vertices of G such that any two adjacent vertices receive distinct colors. The chromatic number $\chi(G)$ of G is the least c such that G has a c-coloring.

In this paper it will be convenient to consider a slightly more general scenario, in which the colors available for each vertex are not necessarily the same. A *list-assignment* L for G is a collection of lists $L(v)$ of colors, one for each vertex v of G. Given a list-assignment L, an *L-list-coloring* of G is a coloring of G (i.e. any two adjacent vertices receive distinct colors, as before), with the additional constraint that each vertex v is colored with a color from its own list $L(v)$. A simple greedy algorithm shows that if for each vertex v, $|L(v)| \geq d_G(v) + 1$ (where $d_G(v)$ denotes the degree of v in G), then G has an L-list-coloring. This is a very useful generalisation of the fact that any graph of maximum degree Δ is $(\Delta + 1)$-colorable.

In this paper we will repeatedly use the following two important algorithmic results on list-coloring. The first result was proved in [3]

▶ **Theorem 2.1** ([3]). *Let G be a graph of maximum degree Δ and let L be a list-assignment such that for any vertex v, $|L(v)| - d_G(v) \geq 1$. Then an L-list-coloring of G can be found by a distributed randomized algorithm running in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds, w.h.p.*

The following stronger result was then proved in [7].

▶ **Theorem 2.2** ([7]). *Let G be a graph of maximum degree Δ and let L be a list-assignment such that for any vertex v, $|L(v)| - d_G(v) \geq \epsilon\Delta$, for some $\epsilon > 0$. Then an L-list-coloring of G can be found by a distributed randomized algorithm running in $O(\log(1/\epsilon)) + 2^{O(\sqrt{\log \log n})}$ rounds, w.h.p.*

Note that Theorem 2.1 can be deduced from Theorem 2.2 by simply setting $\epsilon = 1/\Delta$.

The setting in which these two results will be applied is the following. Let G be a graph of maximum degree Δ with a subset S of vertices that are colored with at most c colors. We want to extend the c-coloring of S to a c-coloring of G (i.e. find a c-coloring of G that agrees with the original coloring on S).

Let $U = V(G) - S$ be the set of uncolored vertices, and for each vertex $u \in U$, let $L(u)$ be the subset of colors from $1, \ldots, c$ that do not appear among the neighbors of u in S. Note

that extending the c-coloring of S to a c-coloring of G is the same as finding an L-list-coloring of $G[U]$, the subgraph of G induced by U.

Let us denote the degree of a vertex $u \in U$ in $G[U]$ by $d_U(u)$. The following simple observation will be particularly useful in combination with Theorems 2.1 or 2.2.

▶ **Observation 2.3.** *If $u \in U$ has at least ℓ repeated colors in its neighborhood, then $|L(u)| - d_U(u) \geq c + \ell - d_G(u) \geq c + \ell - \Delta$.*

Note that this observation will sometimes be used without an explicit number of repeated colors (i.e. $\ell = 0$) and the statement above simply becomes $|L(u)| - d_U(u) \geq c - d_G(u)$.

2.3 Probabilistic tools

Consider a set X of independent random variables, and a set $B = B_1, \ldots, B_n$ of (typically bad) events, each depending on a subset of the variables from X. Consider the graph H with vertex-set B, with an edge between two events if the set of variables they depend on intersect. The graph H is called the *event dependency graph*. Let $d \geq 2$ be the maximum degree of H, and let p be the maximum probability of an event from B.

We will use the following algorithmic versions of the Lovász Local Lemma [6, 11].

▶ **Theorem 2.4** ([6]). *If $epd^2 < 1$, then there is a distributed randomized algorithm, running in H in $O(\log_{1/epd^2}(n))$ rounds w.h.p., that finds a value assignment to the variables of X such that no event from B holds.*

▶ **Theorem 2.5** ([11]). *If $2^{15}pd^8 < 1$, then there is a distributed randomized algorithm, running in H in $2^{O(\log d + \sqrt{\log\log n})}$ rounds w.h.p., that finds a value assignment to the variables of X such that no event from B holds.*

It should be noted that in each subsequent application of Theorem 2.4 or 2.5, the event dependency graph H will only be considered implicitly. The reason is that the variables of X will be associated to the vertices of some other graph G, and the events from B will correspond to connected subgraphs of G of constant radius. Thus, the outcomes of Theorems 2.4 and 2.5 will be computed in G directly (the round complexity is then simply multiplied by a constant, which does not change the asymptotic complexity).

We shall also use the following version of Talagrand's inequality (see the appendix in [17]).

▶ **Theorem 2.6** (Talagrand's Inequality). *Let X be a non-negative random variable whose value is determined by n independent trials $T_1, \ldots, T_n$ and satisfying the following for some $c, r \geq 0$:*

- *changing the outcome of any one trial changes the value of X by at most c.*
- *for any s, if $X \geq s$ then there is a set of at most rs trials whose outcomes certify $X \geq s$.*

Then for any $t \geq 0$,

$$\mathbb{P}\left(|X - \mathbb{E}(X)| > t + 20c\sqrt{r\mathbb{E}(X)} + 64c^2r\right) \leq 4 \cdot \exp\left(-\frac{t^2}{8c^2r(\mathbb{E}(X) + t)}\right)$$

2.4 The dense decomposition

The graph decomposition described in this section is due to Reed [19] (see also [16, 17]). A somewhat similar (although not completely equivalent) decomposition was recently used by Harris, Schneider, and Su [14] (see also [5]) in the context of distributed $(\Delta + 1)$-coloring algorithms.

Consider a graph $G = (V, E)$ of maximum degree Δ. We call a vertex *d-dense* if its neighborhood has more than $\binom{\Delta}{2} - d\Delta$ edges (note that d might depend on Δ). A vertex v that is not d-dense is said to be *d-sparse*.

We say that $S, X_1, X_2, \ldots, X_t$ is a *d-dense decomposition of G* if each of the following holds:

1. $S, X_1, X_2, \ldots, X_t$ partition V;
2. every X_i has between $\Delta - 8d$ and $\Delta + 4d$ vertices;
3. there are at most $8d\Delta$ edges between X_i and $V - X_i$;
4. a vertex is adjacent to at least $\frac{3\Delta}{4}$ vertices of X_i if and only if it is in X_i;
5. every vertex in S is d-sparse.

The sets X_i are called the *dense components* and S is called the *sparse component*. Note that a simple consequence of (4) and (2) is that each dense component has diameter at most 2, provided that $d \leq \frac{\Delta}{8}$. The proof of the following result is given in the full version of the paper [1].

▶ **Lemma 2.7.** *A d-dense decomposition of G can be constructed in $O(1)$ rounds for every $d \leq \frac{\Delta}{100}$.*

3 Graphs with small clique number

In this section we prove Theorem 1.5. We will need the following two results, whose proofs are inspired from the proofs of Lemmas 10 and 16 in [17] (see also Section 10.3 in [16]).

▶ **Lemma 3.1.** *Let G be a graph of (sufficiently large) maximum degree Δ and let $\ell \geq 2^{54} \log \Delta$. Then there is a distributed randomized algorithm that finds a partial coloring of G with $\Delta/2$ colors in $\min\{O(\log_\Delta n), 2^{O(\log \Delta + \sqrt{\log \log n})}\}$ rounds w.h.p., such that for each uncolored vertex v with at least $\ell\Delta$ pairs of non-adjacent vertices in $N(v)$, there are more than $2^{-18}\ell$ repeated colors in $N(v)$.*

▶ **Lemma 3.2.** *Let $S, X_1, \ldots, X_t$ be a $2^{-4}k$-dense decomposition of a graph G of maximum degree $\Delta \geq 30k$ and clique number at most $\Delta - k$. Then there is a distributed randomized algorithm that extends any c-coloring of S with $c \geq \Delta - k/48$ colors to a c-coloring of G in $O(\log_k \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds, w.h.p.*

We now explain how these two results can be combined to provide a proof of Theorem 1.5. It should be mentioned that we have made no significant effort to optimize the various constants appearing throughout the proof, and have chosen instead to focus on making the proof as simple as possible. The proofs of Lemmas 3.1 and 3.2 can be found in the full version of the paper [1].

Proof of Theorem 1.5. If G contains a clique on more than $\Delta - k$ vertices, it can be found in $O(1)$ rounds so we may assume in the remainder that G has clique number at most $\Delta - k$.

We start by using Lemma 2.7 to compute a $2^{-4}k$-dense decomposition $S, X_1, X_2, \ldots, X_t$ of G (note that we have $2^{-4}k \leq 2^{-4}\Delta/30 \leq \Delta/100$, as required). Let T be the vertices of S with degree at least $\Delta - 2^{-5}k$ in S. Since each vertex of $v \in T$ is $2^{-4}k$-sparse, $N(v)$ contains at least

$$\binom{\Delta - 2^{-5}k}{2} - \binom{\Delta}{2} + 2^{-4}k\Delta \geq 2^{-5}k\Delta$$

pairs of non-adjacent vertices in S.

Using Lemma 3.1 with $\ell = 2^{-5}k$, we then obtain a partial coloring of S with at most $\Delta/2 \leq \Delta - 2^{-24}k$ colors in $\min\{O(\log_\Delta n), 2^{O(\log\Delta+\sqrt{\log\log n})}\}$ rounds w.h.p., such that each uncolored vertex of T has more than $2^{-23}k$ repeated colors in its neighborhood. Let U be the set of uncolored vertices of S, and for each vertex of $v \in U$, let $L(v)$ be the set of colors from $1, \ldots, \Delta - 2^{-24}k$ that do not appear in the neighborhood of v. We claim that

$$\text{for each } v \in U,\ |L(v)| - d_U(v) \geq 2^{-24}k, \tag{1}$$

where $d_U(v)$ denotes the number of neighbors of v in U, or equivalently the degree of v in $G[U]$.

To see why (1) holds, consider first the case $v \in U - T$. Observe that in this case v has degree at most $\Delta - 2^{-5}k$ in S, and thus (1) follows directly from Observation 2.3 with $c = \Delta - 2^{-24}k$, $\ell = 0$, and $d_S(v) \leq \Delta - 2^{-5}k$ (which implies $c - d_S(v) \geq \Delta - 2^{-24}k - \Delta + 2^{-5}k \geq 2^{-24}k$).

Assume now that $v \in U \cap T$. Since each uncolored vertex of T has more than $2^{-23}d$ repeated colors in its neighborhood, (1) follows directly from Observation 2.3 with $c = \Delta - 2^{-24}k$ and $\ell = 2^{-23}k$ (which implies $c - \Delta + \ell = \Delta - 2^{-24}k - \Delta + 2^{-23}k = 2^{-24}k$). This concludes the proof of (1).

It follows from (1) that we can use Theorem 2.2 with $\epsilon = 2^{-24}k/\Delta$ to extend the partial coloring of S to all the vertices of S in $O(\log(\Delta/k)) + 2^{O(\sqrt{\log\log n})}$ rounds, w.h.p.

It remains to extend the coloring of S to the dense components $X_1, \ldots, X_t$. Using Lemma 3.2, the coloring of S can then be extended to $X_1, \ldots, X_t$ in $O(\log(\Delta/k)) + 2^{O(\sqrt{\log\log n})}$ rounds, w.h.p. It follows that the overall round complexity is the minimum of $O(\log_\Delta n + \log_k \Delta) + 2^{O(\sqrt{\log\log n})}$ and $2^{O(\log\Delta+\sqrt{\log\log n})}$. In particular, it is w.h.p. $O(\log n/\log\log n)$, for any value of Δ, which concludes the proof of Theorem 1.5. ◀

4 Graphs with chromatic number close to the maximum degree

In this section, we prove the main result of this paper.

We start with a sketch of the proof of Theorem 1.4 (the full proof is given in [1]), and then explain how Theorem 1.3 can be deduced from appropriate parts of the proof of Theorem 1.2 in [17]. It should be noted that our assumption that $c \geq \Delta - k_\Delta + 1$ makes the proof of Theorem 1.3 significantly easier than the proof of Theorem 1.2 in [17], where the main difficulty comes from the case $c = \Delta - k_\Delta$.

4.1 Reducers

A *stable set*, or *independent set*, is a set of pairwise non-adjacent vertices. A *c-reducer* in a graph G is a subset D of vertices consisting of a clique C with $c-1$ vertices and a disjoint stable set S such that every vertex of C is adjacent to all of S but none of $V(G) - D$. Given a graph G with a c-reducer $D = (C, S)$, the graph H obtained from G by removing C and identifying all the vertices of S into a single vertex is called the *reduction* of G with respect to D. Note that G is c-colorable if and only if H is c-colorable, and thus c-reductions preserve c-colorability and non-c-colorability.

Proof of Theorem 1.4 (Sketch). Let Δ be an integer, and assume that either (1) $c \leq \Delta - k_\Delta - 1$, or (2) $c = \Delta - k_\Delta$ and $\Delta = (k_\Delta + 1)(k_\Delta + 2)$.

For $i \geq 1$, we define a graph G_i of maximum degree Δ and a subset C_i of G_i inductively as follows. G_1 is the complete graph on $c+1$ vertices, and C_1 is the set of vertices of G_1. For any $i \geq 2$, G_i is obtained from G_{i-1} by removing an arbitrary vertex v_{i-1} of C_{i-1}, adding a stable set S_i of size $\Delta - c + 2$ and a $(c-1)$-clique C_i such that (1) each neighbor of v_{i-1} in G_{i-1} is adjacent to exactly one vertex of S_i, and (2) each vertex of S_i is adjacent to all the vertices of C_i.

In order to make sure that the maximum degree of G_i is at most Δ, while performing (1) we split as evenly as possible the degree of v_{i-1} between the vertices of S_i (each edge between v_{i-1} and some neighbor u in G_{i-1} becomes an edge joining u and some vertex of S_i in G_i, and we want the degrees of the vertices of S to be as balanced as possible). Since $|S_i| = \Delta - c + 2$, each vertex of C_i has degree Δ in G_i. Each vertex of S_i must also have degree at most Δ so it can have up to $\Delta - c + 1$ neighbors in G_{i-1}. Since v_{i-1} has degree at most Δ, and $(\Delta - c + 2)(\Delta - c + 1) \geq \Delta$, the edges incident to v_{i-1} in G_{i-1} can be split among the vertices of S_i in such way that each vertex of S_i has degree at most Δ in G_i.

We now make a couple of remarks on G_i. It can be observed that G_{i-1} is the reduction of G_i with respect to some c-reducer, and since G_1 is a clique on $c+1$ vertices and reductions preserve c-non-colorability, G_i is not c-colorable. It is also easy to see that any proper subgraph of G_i has chromatic number at most c and G_i has diameter at least $\frac{n}{2\Delta}$, where n denotes the number of vertices of G_i.

Let G be the graph obtained from G_i by deleting a single edge between a vertex of layer $i/2$ (i.e. a vertex that was added at step $i/2$) and a vertex of layer $i/2+1$. As a proper subgraph of G_i, G has maximum degree at most Δ and chromatic number at most c, and it can be checked that any ball of radius less than $\frac{n}{8\Delta}$ in G_i is isomorphic to a ball of the same radius in G. Since G_i is not c-colorable, it follows from a classical observation of Linial [15], that G cannot be colored optimally (i.e. with c colors) in less than $\frac{n}{8\Delta}$ rounds. ◀

4.2 Overview of the proof of Theorem 1.3

We start by considering the first part of the statement of Theorem 1.3: if G is not c-colorable, then some vertex is supposed to output a *certificate* that G is not c-colorable. In order to do so, we will use the following result of Molloy and Reed (Theorem 5 in [17]).

▶ **Theorem 4.1.** *For sufficiently large Δ, and for $c \geq \Delta - k_\Delta + 1$, if G has maximum degree at most Δ, and $\chi(G) > c$, then there is some vertex v in G such that the subgraph induced by $\{v\} \cup N(v)$ is not c-colorable.*

In the LOCAL model of computation, testing the c-colorability of all closed neighborhoods (i.e. all the balls of radius 1) in G can be done in a constant number of rounds, and any vertex finding a non c-colorable subgraph in its closed neighborhood can simply output this subgraph as a certificate of non c-colorability of G. It might be worth pointing that we heavily use the unbounded computational power of the nodes (and the unbounded bandwidth of the edges) in the LOCAL model here when $\Delta \gg \log n$. However, when $\Delta = O(\log n)$, all the closed neighborhoods have logarithmic size, so testing their c-colorability takes polynomial time (in n) in any classical model of computation. Moreover, when $\Delta = O(1)$ the same task can be performed in constant time in any classical model of computation.

We can now assume that G is c-colorable, and the goal is to find a c-coloring of G in $\min\{O((\log\Delta)^{13/12}\log n), 2^{O(\log\Delta+\sqrt{\log\log n})}\}$ rounds w.h.p. The high-level description of the proof is as follows: we set $d = 10^6\sqrt{\Delta}$ and start by computing a d-dense decomposition $S, X_1, \ldots, X_t$ of G. We then delete all the sets X_i that are c-reducers or such that $\overline{G[X_i]}$

has a matching of size at least $100\sqrt{\Delta}$. These sets will be colored at the very end, once the rest of the graph will be colored, using a proof very similar to that of Lemma 3.2, in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ additional rounds (Lemmas 4.2 and 4.3). So we can assume that no set X_i is a c-reducer or has a large antimatching. Using this assumption, we then find a specific c-coloring in each set X_i, independently of the other sets X_j, with desirable properties (Lemma 4.4). Using this coloring of each set X_i, we will construct a new graph F from G by contracting the color classes from the dense sets into single vertices, and adding suitable edges at strategic places in the graph (Lemma 4.5). All these contractions and edge additions can be easily simulated in G, since they involve pairs of vertices at distance at most 4 apart. The final part will consist in coloring F with c colors, and from this coloring it will be easy to deduce a c-coloring of G. Note that because of the edge additions and contraction, the maximum degree of F is not bounded by Δ anymore, but it remains $O(\Delta)$. The coloring of F is then obtained by a very intricate semi-random process. Fortunately, for us it boils down to repeated applications of the Lovász Local Lemma (more precisely, $O((\log \Delta)^{13/12})$ successive applications), and we just need to make sure that Theorem 2.4 and 2.5 can be substituted everywhere in the proof (Lemma 4.6). With this high-level view in mind, we now proceed with the proof.

4.3 Proof of Theorem 1.3

Let $d = 10^6\sqrt{\Delta}$. We first compute a d-dense decomposition $S, X_1, \ldots, X_t$ of G in $O(1)$ rounds using Lemma 2.7.

A c-reducer $D = (C, S')$ is said to be *deletable* if there are fewer than c vertices in $G - D$ with a neighbor in S. Observe that if $D = (C, S')$ is a deletable c-reducer in G, then any c-coloring of $G - D$ can be extended to D (since there is a color which does not appear in the neighborhood of S' in $G - D$). It was observed in [17, Observation 8] that when $c \geq \Delta - k_\Delta + 1$, any c-reducer is deletable. It has the following consequence.

▶ **Lemma 4.2.** *Let X^r be the union of all the c-reducers X_i. Then there is a distributed randomized algorithm (running in G) that extends any c-coloring of $G - X^r$ to G in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds, w.h.p.*

Proof. For each c-reducer $X_i = (C_i, S_i)$, perform the reduction of G with respect to X_i (i.e. delete the clique C_i, and identify all the vertices of S_i into a single vertex v_i). Let R be the resulting graph, and let N be the set of newly created vertices in R. Note that the c-coloring of $G - X^r$ corresponds to a c-coloring of $R - N$, and our goal is simply to extend this coloring to R (once this is done, we only have to assign the color of v_i to all the vertices of the stable set S_i in G, and to color C_i with the $c - 1$ colors distinct from that of v_i, which can clearly be done in $O(1)$ rounds). Since each X_i we consider here is deletable, each vertex $v_i \in N$ has degree at most $c - 1$ in R. It follows from Observation 2.3 and Theorem 2.1 (similarly as in Section 3) that the c-coloring of $R - N$ can be extended to N by a distributed randomized algorithm running in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds w.h.p., as desired. ◀

We say that a dense set X_i is *hollow* if $\overline{G[X_i]}$ (the complement of $G[X_i]$) contains a matching of size at least $100\sqrt{\Delta}$. We now rephrase Lemma 16 from [17] for our convenience (the proof of Lemma 4.3 follows the same lines as that of Lemma 3.2).

▶ **Lemma 4.3.** *Let X^h be the union of the all the hollow sets X_i. Then any c-coloring of $G - X^h$ can be extended to G by a distributed randomized algorithm running w.h.p. in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds.*

We temporarily delete from G all the X_i that are c-reducers or hollow. These sets of vertices will be colored at the very end using Lemmas 4.2 and 4.3. Let H be the graph obtained from G by removing the dense components from Lemmas 4.2 and 4.3. Note that the restriction of the decomposition $S, X_1, \ldots, X_t$ to H is still a d-dense decomposition of H, and for convenience we keep denoting it in this way (even if some sets X_i have disappeared). It follows from our construction that no dense set X_i in H is a c-reducer or is such that $\overline{H[X_i]}$ contains a matching of size at least $100\sqrt{\Delta}$.

Given a subset Y of vertices from some dense component X_i, an *external neighbor* of Y is a vertex outside of X_i with a neighbor in Y. Recall that a coloring of a graph G partitions the vertex-set of G into stable sets, which are called the *color classes* associated to the coloring. Given a c-coloring of X_i, we define C_i as the set of vertices of X_i whose color class is a singleton. We say that a c-coloring of X_i is *nice* if:

(1) C_i is a clique of size at least $\Delta - 2 \cdot 10^6\sqrt{\Delta}$,

(2) each vertex from any color class of size at least 3 is adjacent to all the vertices of C_i, and

(3) if $\{x, y\}$ is a color class of size 2, then either there is $z \in C_i$ such that x, y are both adjacent to all the vertices of $C_i - \{z\}$, or one of x, y is adjacent to all the vertices of C_i and the other is adjacent to all but at most $\frac{\Delta}{4} + 10^7\sqrt{\Delta}$ vertices of C_i.

Note that the unique c-coloring of a c-reducer is nice. Lemma 4.3 now allows us to use the following result of [17]. The proof heavily uses the crucial property that no dense set X_i contains a large antimatching.

▶ **Lemma 4.4** (Lemmas 19, 20, 21, and 25 in [17]). *Each dense set X_i of H has a nice c-coloring such that:*

(a) *If a color class is not the unique largest colour class in X_i, then it has at most $\frac{\Delta}{2} + 10\sqrt{\Delta}$ external neighbors.*

(b) *Every color class of X_i has at most $c - \sqrt{\Delta} + 3$ external neighbors.*

(c) *If there is a colour class of X_i with more than $c - 10^8\sqrt{\Delta}$ external neighbor, then $|C_i| \geq c - 2 \cdot 10^8$ and each vertex of C_i has at most $3 \cdot 10^8$ external neighbors.*

(d) *If there is a colour class of X_i with more than $c - 2\sqrt{\Delta} + 3$ external neighbours then $|C_i| = c - 1$ and each vertex of C_i has at most 5 external neighbors.*

(e) *If there is a colour class of X_i with more than $c - 2\Delta^{3/4}$ external neighbors then $|C_i| \geq c - 5\Delta^{1/4}$ and each vertex of C_i has at most $8\Delta^{1/4}$ external neighbors.*

We stress that the union of the c-colorings of each of the dense components X_i is not necessarily a c-coloring of the union of the dense components: there might be some edges between vertices of different sets X_i having the same color. It should be noted that parts (b)–(e) of this result, as stated here, look a bit different from their counterparts from Lemma 25 in [17]. Indeed, each of properties (b)–(e) in Lemma 25 from [17] starts by the precondition *"If X_i is not a reducer or a near-reducer"*. We assumed earlier that X_i is not a c-reducer, so this part of the precondition can certainly be omitted in our case. A *c-near-reducer* is a subgraph D which is the union of a clique C of size $c - 1$ and a stable set S' of size $\Delta - c + 1$, such that each vertex of C is adjacent to every vertex of S' (in particular each vertex of C has at most one neighbor outside D). Note that each vertex of S' has at most $\Delta - c + 1$ neighbors outside D, and thus S' has at most $(\Delta - c + 1)^2$ neighbors outside D. Since $c \geq \Delta - k_\Delta + 1$ and $\sqrt{\Delta} - 3 < k_\Delta = \left\lfloor \sqrt{\Delta + 1/4} - 3/2 \right\rfloor \leq \sqrt{\Delta + 1/4} - 3/2$, S' has at most

$$(\Delta - c + 1)^2 \leq k_\Delta^2 \leq \Delta - 3k_\Delta - 2 \leq c - 2k_\Delta - 3 \leq c - 2\sqrt{\Delta} + 3$$

neighbors outside D. In particular, in our case (i.e. when $c \geq \Delta - k_\Delta + 1$), any dense set X_i which is a c-near-reducer satisfies Lemma 4.4(a)–(e), so we can indeed remove the preconditions from Lemma 25 in [17]. Note also that since each dense set X_i has diameter at most 2, a nice coloring of each X_i with the additional properties of Lemma 4.4 can be found in $O(1)$ rounds.

Based on the nice c-coloring of each of the dense components X_i resulting from Lemma 4.4, we now construct (locally) a new graph F from H, which will be easier to color with a semi-random procedure, and such that any c-coloring of F can be turned (locally and efficiently) into a c-coloring of H.

▶ **Lemma 4.5** (Lemma 12 as used in the proof of Theorem 43 in [17]). *We can construct locally in H in $O(1)$ rounds a graph F of maximum degree at most $10^9\Delta$ (such that a c-coloring of H can be deduced from any c-coloring of F in $O(1)$ rounds) and find a partition of the vertices of F into $S, B, A_1, \ldots, A_t$ such that:*

- **(a)** *Every A_i is a clique with $c - 10^8\sqrt{\Delta} \leq |A_i| \leq c$.*
- **(b)** *Every vertex of A_i has at most $10^8\sqrt{\Delta}$ neighbors in $F - A_i$.*
- **(c)** *There is a set $\mathrm{All}_i \subseteq B$ of $c - |A_i|$ vertices which are adjacent to all of A_i. Every other vertex of $F - A_i$ is adjacent to at most $\frac{3}{4}\Delta + 10^8\sqrt{\Delta}$ vertices of A_i.*
- **(d)** *Every vertex of S either has fewer than $\Delta - 3\sqrt{\Delta}$ neighbors in S or has at least $900\Delta^{3/2}$ non-adjacent pairs of neighbors within S.*
- **(e)** *Every vertex of B has fewer than $c - \sqrt{\Delta} + 9$ neighbors in $F - \bigcup_j A_j$.*
- **(f)** *If a vertex $\in B$ has at least $c - \Delta^{3/4}$ neighbors in $F - \bigcup_j A_j$, then there is some i such that: v has at most $c - \sqrt{\Delta} + 9$ neighbors in $F - A_i$ and every vertex of A_i has at most $30\Delta^{1/4}$ neighbors in $F - A_i$.*
- **(g)** *For every A_i, every two vertices outside of $A_i \cup \mathrm{All}_i$ which have at least $2\Delta^{9/10}$ neighbors in A_i are joined by an edge of F.*

There is one subtlety in the application of Lemma 12 from [17]: the statement of Lemma 12 there start with the precondition *"For any minimum counterexample"*. Here we avoid this precondition in the same way Molloy and Reed avoid it in their application of Lemma 12 in the algorithmic proof of Theorem 43 from [17] (by starting to remove deletable reducers and hollow sets).

We explain briefly how the graph F is constructed in [17] to stress that the construction can indeed be performed locally in H (and then in G).

The construction starts by doing the following for each colored dense component X_i. Recall that C_i was defined above as the set of vertices of X_i whose color class is a singleton, and it follows from the definition of a nice coloring that C_i is a clique of size at least $\Delta - 2 \cdot 10^6\sqrt{\Delta}$. Now, each color class of size at least 2 (i.e. each color class which is not a singleton) in X_i is contracted into a single vertex, and vertices and edges are added inside X_i to make it into a clique D_i of size precisely c. It can be proved using Lemma 4.4 that the maximum degree does not increase too much and that each clique D_i is not much larger than C_i (see Lemma 29 in [17]).

A significant issue when trying to find a c-coloring of H (or rather the current modification of H) is that given a clique D_i, there might be vertices outside D_i that have many neighbors (say more than $\frac{3\Delta}{4}$) in C_i. Each such vertex must be in $D_j - C_j$, for some $j \neq i$. Consider such a vertex $v \in D_j - C_j$, with many neighbors in C_i. We need to make sure that the color of v will be used by one of the few non-neighbors of v in D_i, and one way to do it is, for some vertices $w \in D_i$, to construct a set R_w of vertices with many neighbors in C_i such that

$\{w\} \cup R_w$ is a stable set and every vertex with many neighbors in C_i lies in such a set R_w. We then contract each set $\{w\} \cup R_w$ into a single vertex (this will force that all these vertices have the same color at the end), and denote by A_i the set C_i after the removal of the vertices w for which some set R_w was defined. We also set $\text{All}_i = D_i - A_i$. Again it can be proved that the maximum degree does not increase too much and each A_i is not too small compared to C_i (see Lemma 30 in [17]).

A second issue (related to the issue described above) is that we need to prevent that many different external neighbors of A_i are all colored with the same color, and their neighborhoods cover A_i (this would prevent this color from being used in A_i). The way it is solved in [17] is by adding an edge between every pair of external neighbors of A_i having at least $\Delta^{9/10}$ neighbors in A_i. It is proved (see Lemma 31 in [17]) that it does not increase the maximum degree too much and is enough to deduce properties (a)–(g) of Lemma 4.5 (the issue raised in this paragraph is in particular related to property (g)).

To sum up, F has been obtained from H by identifying (or adding edges between) pairs of vertices at distance at most 4, since each dense component has diameter at most 2 and any two vertices that have been identified or joined by an edge have a neighbor in the same dense component. Moreover, each modification has been carried out independently by each dense set X_i (even if the modifications had some impact outside of X_i), so F can be simulated by H (and then by G) with at most a small multiplicative loss on the round complexity. It is also clear that a c-coloring of H can be obtained from any c-coloring of F in $O(1)$ rounds.

It remains to show how to efficiently color F with c colors.

▶ **Lemma 4.6.** *The graph F described in Lemma 4.5 can be colored with c colors in $\min\{O((\log\Delta)^{13/12}\log n), 2^{O(\log\Delta+\sqrt{\log\log n})}\}$ rounds, w.h.p.*

We will be rather brief here (the proof of the corresponding sequential statement, Lemma 13 in [17], takes 20 pages). Consider some $1 \leq i \leq t$. Since $\text{All}_i \cup A_i$ forms a clique of size c, we need to make sure that the colors that do not appear in All_i do not appear either on too many external neighbors of A_i. A key property of the construction of F (see properties (c) and (g) in Lemma 4.5) is that for any color x, there is at most one vertex $v \notin \text{All}_i \cup A_i$ having at least $2\Delta^{9/10}$ neighbors in A_i that is colored x, and moreover v has at most $\frac{3}{4}\Delta + o(\Delta)$ neighbors in A_i. The goal will be to maintain this property throughout the whole process, namely that all of the time, at most $\frac{3}{4}\Delta + o(\Delta)$ vertices of A_i have a neighbor colored x outside of $\text{All}_i \cup A_i$ (let us call this event $E(i,x)$).

The starting point will be to color S (the d-sparse vertices, see property (d) of Lemma 4.5) randomly as in the proof of Lemma 3.1, i.e. with the property that many colors are repeated in the neighborhoods of the high degree vertices, but also with the additional property that $E(i,x)$ still holds for any i, x after the coloring.

We then proceed to extend the coloring to B. Recall that by property (e) of Lemma 4.5, each vertex of B has at most $c - \Omega(\sqrt{\Delta})$ neighbors in $F - \bigcup_j A_j$. It turns out that it is a bit too high to extend randomly the coloring of S to B while maintaining property $E(i,x)$, so instead we color the remaining vertices in this order:

1. We first color the set B_H of vertices of B with at most $c - \Delta^{3/4}$ neighbors in $F - \bigcup_j A_j$ (coloring these vertices will preserve $E(i,x)$).
2. We then color the sets A_i such that each vertex of A_i has at most $30\Delta^{1/4}$ neighbors outside of $\text{All}_i \cup A_i$.
3. We color $B_L = B - B_H$, using property (f) of Lemma 4.5 (which implies that property $E(i,x)$ can now be preserved while coloring these vertices).
4. Finally we color the sets A_i that have not been colored yet.

The proofs that desirable properties are maintained during the coloring of the vertices of S and B and the A_i are fairly similar to the proof of Lemma 3.1 (see the full version of the paper [1]), in the sense that they boil down to the estimation of the expectation of some random variables, the proof that these random variables are highly concentrated, and then some application of the Lovász Local Lemma.

We should note two important differences, though.

- The first is that instead of a single random partial coloring, followed by a greedy procedure completing the coloring, the process for coloring S, B_H, and B_L here involves multiple rounds (more specifically, at most $O((\log \Delta)^{13/12})$ rounds) of random partial coloring and a careful study of all the random variables throughout the process.
- The second is that while coloring the A_i, the partial random coloring procedure is a bit different than in the proof of Lemma 3.1. Recall that each A_i is a clique, so assigning each vertex a color uniformly at random, and then uncoloring pairs of vertices with the same color would be extremely unpractical. Instead, each A_i is colored with a permutation of the $|A_i|$ colors not appearing on All_i, taken uniformly at random among all the possible permutations. A consequence is that instead of using Talagrand's Inequality to prove the concentration of random variables around their expectation, McDiarmid's Inequality has to be used instead (see [17]), but the resulting bounds are of a similar order of magnitude.

It can be checked that in all the applications of the Lovász Local Lemma in [17], bad events correspond to subgraphs of H of bounded radius, and the probabilities of the bad events are smaller than any fixed polynomial function of the maximum degree of the event dependency graph (these probabilities are typically of order $\exp(-d^\alpha)$ or $\exp(-\beta \log^2 d)$, where $\alpha, \beta > 0$ and d is the maximum degree of the event dependency graph), so in particular Theorem 2.4 and 2.5 can be substituted everywhere in the proof, and since the semi-random process involves at most $O((\log \Delta)^{13/12})$ successive applications of the Lovász Local Lemma[1], the c-coloring of F can be obtained in $\min\{O((\log \Delta)^{13/12} \log n), 2^{O(\log \Delta + \sqrt{\log \log n})}\}$ rounds, w.h.p.

We find it necessary to insist on a technical (but important) detail here. Theorems 2.4 and 2.5 use the so-called *variable setting* of the Local Lemma, which covers most applications of the original Local Lemma but not all of them. In particular we have to be careful here since the coloring of the A_i involved random permutations of colors assigned to a given set of vertices, instead of colors chosen uniformly at random for each vertex, and it is not clear at first sight whether the former can be handled in the variable setting. It turns out that it can, since in the proof of Lemmas 39 and 40 in [17] the graph under consideration has one vertex for each uncolored A_i, and an edge between two vertices if the corresponding sets A_i are adjacent in H (since each set A_i is a clique, this graph can be simulated within H). The variable associated to each vertex is the random permutation of colors assigned to the corresponding set A_i, so this is indeed an instance of the variable setting of the Local Lemma, and we can use Theorems 2.4 and 2.5.

[1] In the proof of Molloy and Reed [17] the authors use $O(\Delta^\lambda)$ successive applications of the Local Lemma (for any fixed constant $\lambda > 0$), but the proof can easily be optimized to work with only $O((\log \Delta)^{13/12})$ applications of the Local Lemma. The bound $\log^{13/12} \Delta$ comes from the proof of the concentration of Z'_C, page 175 of [17], which dominates the other related bounds on the number of iterations in the proof of Lemma 34 of [17]. Note that the authors of [17] were aiming at a polynomial complexity, so it did not make much sense for them to replace the polynomial number of iterations by a polylogarithmic number of iterations, at the cost of tedious computations.

Now that F has been colored with c colors, we obtain a c-coloring of H in $O(1)$ rounds using Lemma 4.5, and it remains to color the dense components X_i that are c-reducers, or such that $\overline{G[X_i]}$ contains a matching of size at least $100\sqrt{\Delta}$ (recall that these dense components had been removed from the graph at the beginning of the procedure). It follows from Lemmas 4.2 and 4.3 that the c-coloring of H can be extended to the remaining dense components of G w.h.p. in $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds, which concludes the proof of Theorem 1.3. ◀

4.4 Summary of our contributions

We now make a brief summary of our contributions (to make clear what we added and subtracted from the proof of Molloy and Reed [17]).

In [17], c-reducers are dealt with slightly differently: some are simply removed as we do here, but some are reduced as in the definition of c-reduction of Section 4.1 (i.e. by removing the clique and contracting the stable set into a single vertex). This operation can create new c-reducers, and thus c-reducers have to be reduced sequentially until no c-reducer appears in the graph (the fact that it has to be done sequentially is essentially the proof of Theorem 1.4). For c-near-reducers, the situation is slightly more complicated (see Lemma 27 in [17]) but again inherently sequential. It is fortunate that in our case (i.e. when $c \geq \Delta - k_\Delta + 1$), we do not need to worry about these cases, as explained after Lemma 4.4. So our contribution is simply to have checked that the initial d-dense decomposition can be computed locally (see Lemma 2.7), that the construction of F can be performed locally, that all the applications of the Local Lemma can be also carried out locally in the phase where the c-coloring of F is obtained, and that the resulting coloring of H can be extended to G locally and efficiently (see Lemmas 4.2 and 4.3).

5 Concluding remarks

Note that using recent results of Ghaffari *et al.* [11], the randomized algorithms in Theorem 1.3 and 1.5 can be replaced by deterministic algorithms with a round complexity of $2^{O(\log \Delta + \sqrt{\log n})}$. An interesting question is whether the dependency in Δ can be significantly reduced (the same question can be asked for Theorem 1.5 and 1.3). It seems to us that techniques that have been developed so far, such as Theorem 1.8 in [11] or the ad-hoc techniques from [9], do not work well in our case.

When the maximum degree Δ is a constant, the list-coloring problem where every vertex v has a list of at least $d(v) + 1$ colors can be solved in $O(\log^* n)$ rounds [13, 15], which is much faster than the round complexity of Theorems 2.1 and 2.2. In this case it is interesting to use a slightly faster version of Theorem 2.5 from [11], with round complexity $\exp(\exp(O(\sqrt{\log \log \log n})))$, or $\exp(\exp(\exp(O(\sqrt{\log \log \log \log n}))))$, or more generally $\exp^{(i)}(O(\sqrt{\log^{(i+1)} n}))$ for any $1 \leq i \leq \log^* n - 2 \log^* \log^* n$. It is not difficult to see that in this case this round complexity dominates the other parts of the algorithms used in this paper. It follows that the round complexity in Theorem 1.3 and 1.5 in the bounded degree case can be replaced by $\exp^{(i)}(O(\sqrt{\log^{(i+1)} n}))$ for any $1 \leq i \leq \log^* n - 2 \log^* \log^* n$. Moreover, any improvement on the round complexity of the distributed Lovász Local Lemma under some criterion would immediately yield an improved complexity in Theorems 1.5 and 1.3 in the case of bounded degree graphs.

References

1. É. Bamas and L. Esperet. Distributed coloring of graphs with an optimal number of colors. *CoRR*, abs/1809.08140, 2018. arXiv:1809.08140.
2. L. Barenboim and M. Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.
3. L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The Locality of Distributed Symmetry Breaking. *J. ACM*, 63, 2016. Article 20.
4. S. Brandt, O. Fischer, J. Hirvonen, B. Keller, T. Lempiäinen, J. Rybicki, J. Suomela, and J. Uitto. A lower bound for the distributed Lovász local lemma. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*, pages 479—-488, 2016.
5. Y.-J. Chang, W. Li, and S. Pettie. An optimal distributed $(\Delta + 1)$-coloring algorithm? In *Proceedings of the 50th ACM Symposium on Theory of Computing (STOC)*, 2018.
6. K.-M. Chung, S. Pettie, and H.-H. Su. Distributed Algorithms for the Lovász Local Lemma and Graph Coloring. *Distributed Computing*, 30:261–280, 2017.
7. M. Elkin, S. Pettie, and H.-H. Su. $(2\Delta - 1)$-Edge-Coloring is Much Easier than Maximal Matching in the Distributed Setting. In *Proceedings 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 355–370, 2015.
8. T. Emden-Weinert, S. Hougardy, and B. Kreuter. Uniquely Colourable Graphs and the Hardness of Colouring Graphs of Large Girth. *Comb. Probab. Comput.*, 7(4):375–386, December 1998. doi:10.1017/S0963548398003678.
9. M. Fischer and M. Ghaffari. Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 18:1–18:16, 2017. doi:10.4230/LIPIcs.DISC.2017.18.
10. P. Fraigniaud, M. Heinrich, and A. Kosowski. Local Conflict Coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 625–634, 2016. doi:10.1109/FOCS.2016.73.
11. M. Ghaffari, D.G. Harris, and F. Kuhn. Derandomizing Local Distributed Algorithms. In *IEEE 59th Annual Symposium on Foundations of Computer Science, FOCS 2018, 7-9 October 2018, Paris, France*, 2018.
12. M. Ghaffari, J. Hirvonen, F. Kuhn, and Y. Maus. Improved Distributed Δ-Coloring. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 427–436, New York, NY, USA, 2018. ACM. doi:10.1145/3212734.3212764.
13. A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel Symmetry-Breaking in Sparse Graphs. *SIAM J. Discrete Math.*, 1(4):434–446, 1988. doi:10.1137/0401044.
14. D.G. Harris, J. Schneider, and H.-H. Su. Distributed $(\Delta + 1)$-Coloring in Sublogarithmic Rounds. *J. ACM*, 65(4):19:1–19:21, 2018. doi:10.1145/3178120.
15. N. Linial. Locality in Distributed Graph Algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.
16. M. Molloy and B.A. Reed. *Graph Colouring and the Probabilistic Method*. Algorithms and Combinatorics. Springer, 2002.
17. M. Molloy and B.A. Reed. Colouring graphs when the number of colours is almost the maximum degree. *J. Comb. Theory, Ser. B*, 109:134–195, 2014. doi:10.1016/j.jctb.2014.06.004.
18. A. Panconesi and A. Srinivasan. The Local Nature of Δ-Coloring and its Algorithmic Applications. *Combinatorica*, 15(2):255–280, 1995. doi:10.1007/BF01200759.
19. B.A. Reed. ω, Δ, and χ. *Journal of Graph Theory*, 27(4):177–212, 1998.
20. J. Schneider and R. Wattenhofer. Distributed Coloring Depending on the Chromatic Number or the Neighborhood Growth. In *Structural Information and Communication Complexity - 18th International Colloquium, SIROCCO 2011, Gdansk, Poland, June 26-29, 2011*, pages 246–257, 2011. doi:10.1007/978-3-642-22212-2_22.

On the Descriptive Complexity of Color Coding

Max Bannach
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
bannach@tcs.uni-luebeck.de

Till Tantau
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract

Color coding is an algorithmic technique used in parameterized complexity theory to detect "small" structures inside graphs. The idea is to derandomize algorithms that first randomly color a graph and then search for an easily-detectable, small color pattern. We transfer color coding to the world of descriptive complexity theory by characterizing – purely in terms of the syntactic structure of describing formulas – when the powerful second-order quantifiers representing a random coloring can be replaced by equivalent, simple first-order formulas. Building on this result, we identify syntactic properties of first-order quantifiers that can be eliminated from formulas describing parameterized problems. The result applies to many packing and embedding problems, but also to the long path problem. Together with a new result on the parameterized complexity of formula families involving only a fixed number of variables, we get that many problems lie in FPT just because of the way they are commonly described using logical formulas.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Fixed parameter tractability

Keywords and phrases color coding, descriptive complexity, fixed-parameter tractability, quantifier elimination, para-AC^0

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.11

Related Version `https://arxiv.org/abs/1901.03364`

1 Introduction

Descriptive complexity provides a powerful link between logic and complexity theory: We use a logical formula to *describe* a problem and can then infer the computational complexity of the problem just from the *syntactic structure* of the formula. As a striking example, Fagin's Theorem [9] tells us that 3-colorability lies in NP just because its describing formula ("there exist three colors such that all adjacent vertex pairs have different colors") is an existential second-order formula. In the context of fixed-parameter tractability theory, methods from descriptive complexity are also used a lot – but commonly to show that problems are *difficult*. For instance, the A- and W-hierarchies are defined in logical terms [11], but their hard problems are presumably "beyond" the class FPT of fixed-parameter tractable problems.

The methods of descriptive complexity are only rarely used to show that problems are *in* FPT. More precisely, the syntactic structure of the natural logical descriptions of standard parameterized problems found in textbooks are not known to imply that the problems lie in FPT – even though this is known to be the case for many of them. To appreciate the underlying difficulties, consider the following three parameterized problems: p-MATCHING, p-TRIANGLE-PACKING, and p-CLIQUE. In each case, we are given an undirected graph as input and a number k and we are then asked whether the graph contains k vertex-disjoint edges (a size-k matching), k vertex-disjoint triangles, or a clique of size k, respectively. The problems are known to have widely different complexities (maximal matchings can actually be found in polynomial time, triangle packing lies at least in FPT, while finding cliques is

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 11; pp. 11:1–11:16

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

W[1]-complete) but *very* similar logical descriptions:

$$\alpha_k = \exists x_1 \cdots \exists x_{2k} \big(\textstyle\bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i=1}^{k} Ex_{2i-1}x_{2i} \big), \tag{1}$$

$$\beta_k = \exists x_1 \cdots \exists x_{3k} \big(\textstyle\bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i=1}^{k} (Ex_{3i-2}x_{3i-1} \wedge Ex_{3i-2}x_{3i} \wedge Ex_{3i-1}x_{3i}) \big), \tag{2}$$

$$\gamma_k = \exists x_1 \cdots \exists x_k \big(\textstyle\bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i \neq j} Ex_i x_j \big). \tag{3}$$

The family $(\alpha_k)_{k \in \mathbb{N}}$ of formulas is clearly a natural "slicewise" description of the matching problem: A graph $\mathcal{G}$ has a size-k matching if, and only if, $\mathcal{G} \models \alpha_k$. The families $(\beta_k)_{k \in \mathbb{N}}$ and $(\gamma_k)_{k \in \mathbb{N}}$ are natural parameterized descriptions of the triangle packing and the clique problems, respectively. Well-known results on the descriptive complexity of parameterized problems allow us to infer [11] from the above descriptions that all three problems lie in W[1], but offer no hint why the first two problems actually lie in the class FPT – syntactically the clique problem arguably "looks like the easiest one" when in fact it is semantically the most difficult one. The results of this paper will remedy this: We will show that the syntactic structures of the formulas α_k and β_k imply membership of p-MATCHING and p-TRIANGLE-PACKING in FPT.

The road to deriving the computational complexity of parameterized problems just from the syntactic properties of slicewise first-order descriptions involves three major steps: First, a characterization of when the color coding technique is applicable in terms of syntactic properties of second-order quantifiers. Second, an exploration of how these results on second-order formulas apply to first-order formulas, leading to the notion of *strong* and *weak* quantifiers and to an elimination theorem for weak quantifiers. Third, we add a new characterization to the body of known characterizations of how classes like FPT can be characterized in a slicewise fashion by logical formulas.

Our Contributions I: A Syntactic Characterization of Color Coding. The hard triangle packing problem from above becomes almost trivial when we just wish to check whether a *vertex-colored* graph contains a red triangle, a green triangle, a blue triangle, a yellow triangle, and so on for k different colors. The ingenious idea behind the color coding technique of Alon, Yuster, and Zwick [1] is to reduce the original problem to the much simpler colored version by simply *randomly coloring the graph.* Of course, even if there are k disjoint triangles, we will most likely *not* color them monochromatically and differently, *but* the probability of "getting lucky" is nonzero and depends only on the parameter k. Even better, Alon et al. point out that one can *derandomize the coloring easily by using universal hash functions to color each vertex with its hash value.*

Applying this idea in the setting of descriptive complexity was recently pioneered by Chen et al. [6]. Transferred to the triangle packing problem, their argument would roughly be: "Testing for each color i whether there is a monochromatic triangle of color i can be done in first-order logic using something like $\bigwedge_{i=1}^{k} \exists x \exists y \exists z (Exy \wedge Eyz \wedge Exz \wedge C_i x \wedge C_i y \wedge C_i z)$. Next, instead of testing whether x has color i using the formula $C_i x$, we can test whether x gets hashed to i by a hash function. Finally, since computing appropriate universal hash functions only involves addition and multiplication, we can express the derandomized algorithm using an arithmetic first-order formula of low quantifier rank." Phrased differently, Chen et al. would argue that $\bigwedge_{i=1}^{k} \exists x \exists y \exists z (Exy \wedge Eyz \wedge Exz \wedge C_i x \wedge C_i y \wedge C_i z)$ together with the requirement that the C_i are pairwise disjoint is (ignoring some details) equivalent to $\delta_k = \exists p \exists q \bigwedge_{i=1}^{k} \exists x \exists y \exists z (Exy \wedge Eyz \wedge Exz \wedge \text{HASH}_k(x,p,q) = i \wedge \text{HASH}_k(y,p,q) = i \wedge \text{HASH}_k(z,p,q) = i)$, where $\text{HASH}_k(x,p,q) = i$ is a formula that is true when "x is hashed to i by a member of a universal family of hash functions indexed by q and p."

The family $(\delta_k)_{k\in\mathbb{N}}$ may seem rather technical and, indeed, its importance becomes visible only in conjunction with another result by Chen et al. [6]: They show that a parameterized problem lies in para-AC^0, one of the smallest "sensible" subclasses of FPT, if it can be described by a family $(\phi_k)_{k\in\mathbb{N}}$ of FO[$+,\times$] formulas of *bounded quantifier rank* such that the finite models of ϕ_k are exactly the elements of the kth slice of the problem. Since the triangle packing problem can be described in this way via the family $(\delta_k)_{k\in\mathbb{N}}$ of formulas, all of which have a quantifier rank 5 plus the constant number of quantifiers used to express the arithmetics in the formulas $\textsc{hash}_k(x,p,q)=i$, we get p-TRIANGLE-PACKING $\in$ FPT.

Clearly, this beautiful idea cannot work in all situations: If it also worked for the formula mentioned earlier expressing 3-colorability, 3-colorability would be first-order expressible, which is known to be impossible. Our first main contribution is a *syntactic characterization of when the color coding technique is applicable,* that is, of why color coding works for triangle packing but not for 3-colorability: For triangle packing, the colors C_i are applied to variables only inside *existential scopes* ("$\exists x\exists y\exists z$") while for 3-colorability the colors R, G, and B are also applied to variables inside universal scopes ("for all adjacent vertices"). In general, see Theorem 3.1 for the details, we show that a second-order quantification over an arbitrary number of disjoint colors C_i can be replaced by a fixed number of first-order quantifiers whenever none of the C_i is used in a universal scope.

Our Contributions II: New First-Order Quantifier Elimination Rules. The "purpose" of the colors C_i in the formulas $\bigwedge_{i=1}^{k}\exists x\exists y\exists z(Exy\wedge Eyz\wedge Exz\wedge C_ix\wedge C_iy\wedge C_iz)$ is not that the three vertices of a triangle get a particular color, but just that they get a color *different* from the color of all other triangles. Indeed, our "real" objective in these formulas is to ensure that the vertices of a triangle are *distinct* from the vertices in the other triangles – and giving vertices different colors is "just a means" of ensuring this.

In our second main contribution we explore this idea further: If the main (indeed, the only) use of colors in the context of color coding is to ensure that certain vertices are different, let us do away with colors and instead focus on the notion of *distinctness.* To better explain this idea, consider the following family, also describing triangle packing, where the only change is that we now require (a bit superfluously) that even the vertices inside a triangle get different colors: $\bigwedge_{j=1}^{k}\exists x\exists y\exists z(Exy\wedge Eyz\wedge Exz\wedge C_{3j-2}x\wedge C_{3j-1}y\wedge C_{3j}z)$. Observe that each C_i is now applied to exactly one variable (x, y, or z in one of the many literals) and the only "effect" that all these applications have is to ensure that the variables are different. In particular, the formula is equivalent to

$$\exists x_1\cdots\exists x_{3k}\bigwedge_{i\neq j}x_i\neq x_j\wedge\bigwedge_{j=1}^{k}\exists x\exists y\exists z(Exy\wedge Eyz\wedge Exz\wedge x_{3j-2}=x\wedge x_{3j-1}=y\wedge x_{3j}=z) \tag{4}$$

and these formulas are clearly equivalent to the almost identical formulas from (2).

In a sense, in (4) the many existential quantifiers $\exists x_i$ and the many $x_i\neq x_j$ literals come "for free" from the color coding technique, while $\exists x$, $\exists y$, and $\exists z$ have nothing to do with color coding. Our key observation is a syntactic property that tells us whether a quantifier comes "for free" in this way (we will call it *weak*) or not (we will call it *strong*): Definition 3.4 states (essentially) that weak quantifiers have the form $\exists x(\phi)$ such that x is not free in any universal scope of ϕ and x is used in at most one literal that is not of the form $x\neq y$. To make weak quantifiers easier to spot, we mark their bound variables with a dot (note that this is a "syntactic hint" without semantic meaning). Formulas (4) now read $\exists\dot{x}_1\cdots\exists\dot{x}_{3k}\bigwedge_{i\neq j}\dot{x}_i\neq\dot{x}_j\wedge\bigwedge_{j=1}^{k}\exists x\exists y\exists z(Exy\wedge Exz\wedge Eyz\wedge\dot{x}_{3j-2}=x\wedge\dot{x}_{3j-1}=y\wedge\dot{x}_{3j}=z)$. Observe that x, y, and z are not weak since each is used in three literals that are not inequalities.

We show in Theorem 3.5 that each ϕ is equivalent to a ϕ' whose quantifier rank depends only on the *strong quantifier rank* of ϕ (meaning that we ignore the weak quantifiers) and whose number of variables depends only on the number of strong variables in ϕ'. For instance, the formulas from (4) all have strong quantifier rank 3 and, thus, the triangle packing problem can be described by a family of constant (normal) quantifier rank. Applying Chen et al.'s characterization yields membership in para-AC^0.

As a more complex example, let us sketch a "purely syntactic" proof of the result [3, 5] that the embedding problem for graphs H of tree depth at most d lies in para-AC^0 for each d. Once more, we construct a family (ϕ_H) of formulas of constant strong quantifier rank that describes the problem. For a graph H and a rooted tree T of depth d such that H is contained in T's transitive closure (this is the definition of "H has tree depth d"), let c_1 be the root of T and let $\mathrm{children}(c)$ be the children of c in T. Then the following formula of strong quantifier rank d describes that H can be embedded into a structure:

$$\exists \dot{x}_1 \cdots \exists \dot{x}_{|H|} \big(\textstyle\bigwedge_{i \neq j} \dot{x}_i \neq \dot{x}_j \wedge \exists n_1 (n_1 = \dot{x}_{c_1} \wedge \bigwedge_{c_2 \in \mathrm{children}(c_1)} \exists n_2 (n_2 = \dot{x}_{c_2} \wedge \bigwedge_{c_3 \in \mathrm{children}(c_2)} \exists n_3 (n_3 = \dot{x}_{c_3} \wedge \bigwedge_{c_4 \in \mathrm{children}(c_3)} \exists n_4 (n_4 = \dot{x}_{c_4} \wedge \ldots \bigwedge_{c_d \in \mathrm{children}(c_{d-1})} \exists n_d (n_d = \dot{x}_{c_d} \wedge \bigwedge_{i,j \in \{1,\ldots,d\} : (c_i, c_j) \in E(H)} E n_i n_j) \ldots)))) \big).$$

Our Contributions III: Slicewise Descriptions and Variable Set Sizes. Our third contribution is a new result in the same vein as the already repeatedly mentioned result of Chen et al. [6]: Theorem 2.3 states that a parameterized problem can be described slicewise by a family $(\phi_k)_{k \in \mathbb{N}}$ of arithmetic first-order formulas that all use only a *bounded number of variables* if, and only if, the problem lies in para-$\mathrm{AC}^{0\uparrow}$ – a class that has been encountered repeatedly in the literature [2, 3, 8, 14], but for which no characterization was known. It contains all parameterized problems that can be decided by AC-circuits whose depth depends only on the parameter and whose size is of the form $f(k) \cdot n^c$.

As an example, consider the problem of deciding whether a graph contains a path of length k (no vertex may be visited twice). It can be described (for odd k) by: $\exists s \exists t \exists x (Esx \wedge \exists \dot{x}_1 (\dot{x}_1 = x \wedge \exists y (Exy \wedge \exists \dot{x}_2 (\dot{x}_2 = y \wedge \exists x (Eyx \wedge \exists \dot{x}_3 (\dot{x}_3 = x \wedge \exists y (Exy \wedge \exists \dot{x}_4 (\dot{x}_4 = y \wedge \cdots \wedge \exists x (Eyx \wedge x = t \wedge \exists \dot{x}_k (\dot{x}_k = x \wedge \bigwedge_{i \neq j} \dot{x}_i \neq \dot{x}_j) \ldots))))$. Note that, now, the strong quantifier rank depends on k and, thus, is not constant. However, there are now only four strong variables, namely s, t, x, and y. By Theorem 3.5 we see that the above formulas are equivalent to a family of formulas with a bounded number of variables and by Theorem 2.3 we see that p-LONG-PATH $\in$ para-$\mathrm{AC}^{0\uparrow} \subseteq$ FPT. These ideas also generalize easily and we give a purely syntactic proof of the seminal result from the original color coding paper [1] that the embedding problem for graphs of bounded tree *width* lies in FPT. The core observation – which unifies the results for tree width and depth – is that for each graph with a given tree decomposition, the embedding problem can be described by a formula whose strong nesting structure mirrors the tree structure and whose strong variables mirror the bag contents.

Related Work. Flum and Grohe [10] were the first to give characterizations of FPT and of many subclasses in terms of the syntactic properties of formulas describing their members. Unfortunately, these syntactic properties do not hold for the descriptions of parameterized problems found in the literature. For instance, they show that FPT contains exactly the problems that can be described by families of FO[LFP]-formulas of bounded quantifier rank – but actually describing problems like p-VERTEX-COVER in this way is more or less hopeless and yields little insights into the structure or complexity of the problem. We believe that it is no coincidence that no applications of these beautiful characterizations to concrete

problems could be found in the literature – at least prior to very recent work by Chen and Flum [7], who study slicewise descriptions of problems on structures of bounded tree depth, and the already cited article of Chen et al. [6], who *do* present a family of formulas that describe the vertex cover problem. This family internally uses the color coding technique and is thus closely related to our results. The crucial difference is, however, that we identify syntactic properties of logical formulas that imply that the color coding technique can be applied. It then suffices to find a family describing a given problem that meets the syntactic properties to establish the complexity of the problem: there is no need to actually construct the color-coding-based formulas – indeed, there is not even a need to understand how color coding works in order to decide whether a quantifier is weak or strong.

Organization of this Paper. In Section 2 we first review some of the existing work on the descriptive complexity of parameterized problems. We add to this work in the form of the mentioned characterization of the class para-$\mathrm{AC}^{0\uparrow}$ in terms of a bounded number of variables. Our main technical results are then proved in Section 3, where we establish and prove the syntactic properties that formulas must have in order for the color coding method to be applicable. In Section 4 we then apply the findings and show how membership of different natural problems in para-AC^0 and para-$\mathrm{AC}^{0\uparrow}$ (and, thus, in FPT) can be derived entirely from the syntactic structure of the formulas describing them. Full proofs can be found in the full version, but we include proof sketches in the text.

2 Describing Parameterized Problems

A happy marriage of parameterized complexity and descriptive complexity was first presented in [10]. We first review the most important definitions from [10] and then prove a new characterization, namely of the class para-$\mathrm{AC}^{0\uparrow}$ that contains all problems decidable by AC-circuits of parameter-dependent depth and "FPT-like" size. Since the results and notions will be useful later, but do not lie at the paper's heart, we keep this section brief.

Logical Terminology. We only consider first-order logic and use standard notations, with the perhaps only deviations being that we write relational atoms briefly as Exy instead of $E(x, y)$ and that the literal $x \neq y$ is an abbreviation for $\neg\, x = y$ (recall that a *literal* is an atom or a negated atom). Signatures, typically denoted τ, are always finite and may only contain relation symbols and constant symbols – with one exception: The special unary function symbol $\textsc{succ}$ may also be present in a signature. Let us write $\textsc{succ}^k$ for the k-fold application of $\textsc{succ}$, so $\textsc{succ}^3(x)$ is short for $\textsc{succ}(\textsc{succ}(\textsc{succ}(x)))$. It allows us to specify any fixed non-negative integer without having to use additional variables. An alternative is to dynamically add constant symbols for numbers to signatures as done in [6], but we believe that following [10] and adding the successor function gives a leaner formal framework. Let $\mathrm{arity}(\tau)$ be the maximum arity of relation symbols in τ.

We denote by $\textsc{struc}[\tau]$ the class of all τ-structures and by $|\mathcal{A}|$ the universe of $\mathcal{A}$. As is often the case in descriptive complexity theory, we only consider ordered structures in which the ternary predicates $\textsc{add}$ and $\textsc{mult}$ are available and have their natural meaning. Formally, we say τ is *arithmetic* if it contains all of the predicates $<$, $\textsc{add}$, $\textsc{mult}$, the function symbol $\textsc{succ}$, and the constant symbol 0 (it is included for convenience only). In this case, $\textsc{struc}[\tau]$ contains only those $\mathcal{A}$ for which $<^{\mathcal{A}}$ is a linear ordering of $|\mathcal{A}|$ and the other operations have their natural meaning relative to $<^{\mathcal{A}}$ (with the successor of the maximum element of the universe being itself and with 0 being the minimum with respect to $<^{\mathcal{A}}$). We write $\phi \in \mathrm{FO}[+, \times]$ when ϕ is a τ-formula for an arithmetic τ.

A τ-*problem* is a set $Q \subseteq \text{STRUC}[\tau]$ closed under isomorphisms. A τ-formula ϕ *describes* a τ-problem Q if $Q = \{\mathcal{A} \in \text{STRUC}[\tau] \mid \mathcal{A} \models \phi\}$ and it *describes* Q *eventually* if ϕ describes a set Q' that differs from Q only on structures of a certain maximum size.

▶ **Lemma 2.1.** *For each $\phi \in \text{FO}[+,\times]$ that describes a τ-problem Q eventually, there are quantifier-free formulas α and β such that $(\alpha \wedge \phi) \vee \beta$ describes Q.*

Proof Sketch. Setup α to test structure size. "Hardwire" into β which "small" structures lie in Q. Use SUCC to address the elements of small structures without using quantifiers. ◀

We write $\text{qr}(\phi)$ for the quantifier rank of a formula and $\text{bound}(\phi)$ for the set of its bound variables. For instance, for $\phi = \big(\exists x \exists y(Exz)\big) \vee \forall y(Px)$ we have $\text{qr}(\phi) = 2$, since the maximum nesting is caused by the two nested existential quantifiers, and $\text{bound}(\phi) = \{x, y\}$.

Let us say that ϕ is *in negation normal form* if negations are applied only to atomic formulas.

Describing Parameterized Problems. When switching from classical complexity theory to descriptive complexity theory, the basic change is that "words" get replaced by "finite structures." The same idea works for parameterized complexity theory and, following Flum and Grohe [10], let us define *parameterized problems* as subsets $Q \subseteq \text{STRUC}[\tau] \times \mathbb{N}$ where Q is closed under isomorphisms. In a pair $(\mathcal{A}, k) \in \text{STRUC}[\tau] \times \mathbb{N}$ the number k is, of course, the *parameter value* of the pair. Flum and Grohe now propose to describe such problems slicewise using formulas. Since this will be the only way in which we describe problems, we will drop the "slicewise" in the phrasings and just say that a computable family $(\phi_k)_{k\in\mathbb{N}}$ of formulas *describes* a problem $Q \subseteq \text{STRUC}[\tau] \times \mathbb{N}$ if for all $(\mathcal{A}, k) \in \text{STRUC}[\tau] \times \mathbb{N}$ we have $(\mathcal{A}, k) \in Q$ if, and only if, $\mathcal{A} \models \phi_k$. One can also define a purely logical notion of reductions between two problems Q and Q', but we will need this notion only inside the proof of Theorem 4.2 and postpone the definition till then.

For a class Φ of computable families $(\phi_k)_{k\in\mathbb{N}}$, let us write $\text{X}\Phi$ for the class of all parameterized problems that are described by the members of Φ (we chose "X" to represent a "slicewise" description, which seems to be in good keeping with the usual use of X in other classes such as XP or XL). For instance, the mentioned characterization of FPT in logical terms by Flum and Grohe can be written as $\text{FPT} = \text{X}\{(\phi_k)_{k\in\mathbb{N}} \mid \phi_k \in \text{FO}[\text{LFP}], \max_k \text{qr}(\phi_k) < \infty\}$.

We remark that instead of describing parameterized problems using families, a more standard and at the same time more flexible way is to use reductions to model checking problems. Clearly, if a family $(\phi_k)_{k\in\mathbb{N}}$ of $\mathcal{L}$-formulas describes $Q \subseteq \text{STRUC}[\tau] \times \mathbb{N}$, then there is a very simple parameterized reduction from Q to the model checking problem $\text{p}_\phi\text{-MC}(\mathcal{L})$, where the input is a pair $(\mathcal{A}, \text{num}(\phi))$ and the question is whether both $\mathcal{A} \models \phi$ and $\phi \in \mathcal{L}$ hold. (The function num encodes mathematical objects like ϕ or later tuples like (ϕ, δ) as unique natural numbers.) The reduction simply maps a pair $(\mathcal{A}, k)$ to $(\mathcal{A}, \text{num}(\phi_k))$. Even more interestingly, without going into any of the technical details, it is also not hard to see that as long as a reduction is sufficiently simple, the reverse implication holds, that is, we can replace a reduction to the model checking problem by a family of formulas that describe the problem. We can, thus, use whatever formalism seems more appropriate for the task at hand and – as we hope that this paper shows – it is sometimes quite natural to write down a family that describes a problem.

Parameterized Circuits. For our descriptive setting, we need to slightly adapt the definition of the circuit classes para-AC^0 and para-$\text{AC}^{0\uparrow}$ from [2, 3]: Let us say that a problem $Q \subseteq \text{STRUC}[\tau] \times \mathbb{N}$ is in para-AC^0, if there is a family $(C_{n,k})_{n,k\in\mathbb{N}}$ of AC-circuits (Boolean

circuits with unbounded fan-in) such that for all $(\mathcal{A}, k) \in \text{STRUC}[\tau] \times \mathbb{N}$ we have, first, $(\mathcal{A}, k) \in Q$ if, and only if, $C_{|x|,k}(x) = 1$ where x is a binary encoding of $\mathcal{A}$; second, the size of $C_{n,k}$ is at most $f(k) \cdot n^c$ for some computable function f; third, the depth of $C_{n,k}$ is bounded by a constant; and, fourth, the circuit family satisfies a DLOGTIME-uniformity condition. The class para-AC$^{0\uparrow}$ is defined the same way, but the depth may be $g(k)$ for some computable g instead of only $O(1)$. The following fact and theorem show how these two circuit classes are closely related to descriptions of parameterized problems using formulas:

▶ **Fact 2.2** ([6]). $\text{para-AC}^0 = \text{X}\big\{(\phi_k)_{k\in\mathbb{N}} \;\big|\; \phi_k \in \text{FO}[+,\times], \max_k \text{qr}(\phi_k) < \infty\big\}$.

▶ **Theorem 2.3.** $\text{para-AC}^{0\uparrow} = \text{X}\big\{(\phi_k)_{k\in\mathbb{N}} \;\big|\; \phi_k \in \text{FO}[+,\times], \max_k |\text{bound}(\phi_k)| < \infty\big\}$.

Proof Sketch. Basically, this follows from the well-known link between circuit depth and size and the number of variables used in a formula, see for instance [15]: The *quantifier rank* of a first-order formula naturally corresponds to the *depth* of a circuit that solves the model checking problem for the formula. The *number of variables* corresponds to the *exponent of the polynomial* that bounds the size of the circuit (the paper [13] is actually entitled "DSPACE$[n^k]$ = VAR$[k+1]$"). A simple new observation (but needed for the theorem – usually only one formula is considered) is that the *length* of the formula is linked *multiplicatively* to the size of the circuit. ◀

3 Syntactic Properties Allowing Color Coding

The color coding technique [1] is a powerful method from parameterized complexity theory for "discovering small objects" in larger structures. Recall the example from the introduction: While finding k disjoint triangles in a graph is difficult in general, it is easy when the graph is colored with k colors and the objective is to find for each color one triangle having this color. The idea behind color coding is to reduce the (hard) uncolored version to the (easy) colored version by *randomly* coloring the graph and then "hoping" that the coloring assigns a different color to each triangle. Since the triangles are "small objects," the probability that they do, indeed, get different colors depends only on k. Even more importantly, Alon et al. noticed that we can derandomize the coloring procedure simply by coloring each vertex by its hash value with respect to a simple family of universal hash functions that only use addition and multiplication [1]. This idea is beautiful and works surprisingly well in practice [12], but using the method inside proofs can be tricky: On the one hand, we need to "keep the set sizes under control" (they must stay roughly logarithmic in size) and we "need to actually identify the small set based just on its random coloring." Especially for more complex proofs this can lead to rather subtle arguments.

In the present section, we identify *syntactic* properties of formulas that guarantee that the color coding technique can be applied. The property is that the colors (the predicates C_i in the formulas) are not in the scope of a universal quantifier (this restriction is necessary, as the example of the formula describing 3-colorability shows).

As mentioned already in the introduction, the main "job" of the colors in proofs based on color coding is to ensure that vertices of a graph are different from other vertices. This leads us to the idea of focusing entirely on the notion of distinctness in the second half of this section. This time, there will be syntactic properties of existentially bounded first-order variables that will allow us to apply color coding to them.

3.1 Formulas With Color Predicates

In graph theory, a *coloring* of a graph can either refer to an arbitrary assignment that maps each vertex to a color or to such an assignment in which vertices connected by an edge must get different colors (sometimes called *proper* colorings). For our purposes, colorings need not be proper and are thus partitions of the vertex set into *color classes.* From the logical point of view, each color class can be represented by a unary predicate. A *k-coloring of a τ-structure $\mathcal{A}$* is a structure $\mathcal{B}$ over the signature $\tau_{k\text{-colors}} = \tau \cup \{C_1^1, \dots, C_k^1\}$, where the C_i are fresh unary relation symbols, such that $\mathcal{A}$ is the τ-restriction of $\mathcal{B}$ and such that the sets $C_1^{\mathcal{B}}$ to $C_k^{\mathcal{B}}$ form a partition of the universe $|\mathcal{A}|$ of $\mathcal{A}$.

Let us now formulate and prove the first syntactic version of color coding. An example of a possible formula ϕ in the theorem is $\bigwedge_{i=1}^{k} \exists x \exists y \exists z (Exy \wedge Eyz \wedge Exz \wedge C_i x \wedge C_i y \wedge C_i z)$, for which the theorem tells us that there is a formula ϕ' of constant quantifier rank that is true exactly when there are pairwise disjoint sets C_i that make ϕ true.

▶ **Theorem 3.1.** *Let τ be an arithmetic signature and let k be a number. For each first-order $\tau_{k\text{-colors}}$-sentence ϕ in negation normal form in which no C_i is inside a universal scope, there is a τ-sentence ϕ' such that:*

1. *For all $\mathcal{A} \in \text{STRUC}[\tau]$ we have $\mathcal{A} \models \phi'$ if, and only if, there is a k-coloring $\mathcal{B}$ of $\mathcal{A}$ with $\mathcal{B} \models \phi$.*
2. $\text{qr}(\phi') = \text{qr}(\phi) + O(1)$.
3. $|\text{bound}(\phi')| = |\text{bound}(\phi)| + O(1)$.

(Let us clarify that $O(1)$ represents a global constant that is independent of τ and k.)

Proof. Let τ, k, and ϕ be given as stated in the theorem. If necessary, we modify ϕ to ensure that there is no literal of the form $\neg C_i x_j$, by replacing each such literal by the equivalent $\bigvee_{l \neq i} C_l x_j$. After this transformation, the C_i in ϕ are neither in the scope of universal quantifiers nor of negations – and this is also true for all subformulas α of ϕ. We will now show by structural induction that all these subformulas (and, hence, also ϕ) have two *semantic* properties, which we call the *monotonicity property* and the *small witness property* (with respect to the C_i). Afterwards, we will show that these two properties allow us to apply the color coding technique.

Establishing the Monotonicity and Small Witness Properties. Some notations will be useful: Given a τ-structure $\mathcal{A}$ with universe A and given sets $A_i \subseteq A$ for $i \in \{1, \dots, k\}$, let us write $\mathcal{A} \models \phi(A_1, \dots, A_k)$ to indicate that $\mathcal{B}$ is a model of ϕ where $\mathcal{B}$ is the $\tau_{k\text{-colors}}$-structure with universe A in which all symbols from τ are interpreted as in $\mathcal{A}$ and in which the symbol C_i is interpreted as A_i, that is, $C_i^{\mathcal{B}} = A_i$. Subformulas γ of ϕ may have free variables and suppose that x_1 to x_m are the free variables in γ and let $a_i \in A$ for $i \in \{1, \dots, m\}$. We write $\mathcal{A} \models \gamma(A_1, \dots, A_k, a_1, \dots, a_m)$ to indicate that γ holds in the just-described structure $\mathcal{B}$ when each x_i is interpreted as a_i.

▶ **Definition 3.2.** *Let γ be a $\tau_{k\text{-colors}}$-formula with free variables x_1 to x_m. We say that γ has the* monotonicity and the small witness properties *with respect to the C_i if for all τ-structures $\mathcal{A}$ with universe A and all values $a_1, \dots, a_m \in A$ the following holds:*

1. Monotonicity property: *Let $A_1, \dots, A_k \subseteq A$ and $B_1, \dots, B_k \subseteq A$ be sets with $A_i \subseteq B_i$ for all $i \in \{1, \dots, k\}$. Then $\mathcal{A} \models \gamma(A_1, \dots, A_k, a_1, \dots, a_m)$ implies $\mathcal{A} \models \gamma(B_1, \dots, B_k, a_1, \dots, a_m)$.*

2. Small witness property: *If there are any pairwise disjoint sets $B_1, \dots, B_k \subseteq A$ with $\mathcal{A} \models \gamma(B_1, \dots, B_k, a_1, \dots, a_m)$, then there are sets $A_i \subseteq B_i$ whose sizes $|A_i|$ depend only on γ for $i \in \{1, \dots, k\}$, such that $\mathcal{A} \models \gamma(A_1, \dots, A_k, a_1, \dots, a_m)$.*

We now show that ϕ has these two properties (for $m = 0$). For monotonicity, just note that the C_i are not in the scope of any negation and, thus, if some A_i make ϕ true, so will all supersets B_i of the A_i.

To see that the small witness property holds, we argue by structural induction: If ϕ is any formula that does not involve any C_i, then ϕ is true or false independently of the B_i and, in particular, if it is true at all, it is also true for $A_i = \emptyset$ for $i \in \{1, \dots, k\}$. If ϕ is the atomic formula $C_i x_j$, then setting $A_i = \{a_j\}$ and $A_{i'} = \emptyset$ for $i' \neq i$ makes the formula true.

If $\phi = \alpha \wedge \beta$, then α and β have the small witness property by the induction hypothesis. Let $B_1, \dots, B_k \subseteq A$ make ϕ hold in $\mathcal{A}$. Then they also make both α and β hold in $\mathcal{A}$. Let $A_1^\alpha, \dots, A_k^\alpha \subseteq A$ with $A_i^\alpha \subseteq B_i$ be the witnesses for α and let $A_1^\beta, \dots, A_k^\beta \subseteq A$ be the witnesses for β. Then by the monotonicity property, $A_1^\alpha \cup A_1^\beta, \dots, A_k^\alpha \cup A_k^\beta$ makes both α and β true, that is

$$\mathcal{A} \models \alpha(A_1^\alpha \cup A_1^\beta, \dots, A_k^\alpha \cup A_k^\beta, a_1, \dots, a_m)$$

and the same holds for β. Note that $A_i^\alpha \cup A_i^\beta \subseteq B_i$ still holds and that they have sizes depending only on α and β and thereby on ϕ.

For $\phi = \alpha \vee \beta$ we can argue in exactly the same way as for the logical and.

The last case for the structural induction is $\phi = \exists x_m(\alpha)$. Consider pairwise disjoint $B_1, \dots, B_k \subseteq A$ that make ϕ true. Then there is a value $a_m \in A$ such that $\mathcal{A} \models \alpha(B_1, \dots, B_k, a_1, \dots, a_m)$. Now, since α has the small witness property by the induction hypothesis, we get $A_i \subseteq B_i$ of size depending on α for which we also have $\mathcal{A} \models \alpha(A_1, \dots, A_k, a_1, \dots, a_m)$. But then, by the definition of existential quantifiers, these A_i also witness $\mathcal{A} \models \exists x_m \phi(A_1, \dots, A_k, a_1, \dots, a_{m-1})$. (Observe that this is the point where the argument would *not* work for a universal quantifier: Here, for each possible value of a_m we might have a different set of A_i's as witnesses and their union would then no longer have small size.)

Applying Color Coding. Our next step in the proof is to use color coding to produce the partition. First, let us recall the basic lemma on universal hash functions formulated below in a way equivalent to [11, page 347]:

▶ **Lemma 3.3.** *There is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ and all subsets $X \subseteq \{0, \dots, n-1\}$ there exist a prime $p < |X|^2 \log_2 n$ and a number $q < p$ such that the function $h_{p,q}(m) = (q \cdot m \bmod p) \bmod |X|^2$ is injective on X.*

As has already been observed by Chen et al. [6], if we set $k = |X|$ we can easily express the computation underlying $h_{p,q} \colon \{0, \dots, n-1\} \to \{0, \dots, k^2 - 1\}$ using a fixed FO[+, ×]-formula $\rho(k, p, q, x, y)$. That is, if we encode the numbers $k, p, q, x, y \in \{0, \dots, n-1\}$ as corresponding elements of the universe with respect to the ordering of the universe, then $\rho(k, p, q, x, y)$ holds if, and only if, $h_{p,q}(x) = y$. Note that the p and q from the lemma could exceed n for very large X (they can reach up to $n^2 \log_2 n \leq n^3$), but, first, this situation will not arise in the following and, second, this could be fixed by using three variables to encode p and three variables to encode q. Trivially, $\rho(k, p, q, x, y)$ has some constant quantifier rank (the formula explicitly constructed by Chen et al. has $\mathrm{qr}(\rho) = 9$, assuming $k^2 < n$).

Next, we will need the basic idea or "trick" of Alon et al.'s [1] color coding technique: While for appropriate p and q the function $h_{p,q}$ will "just" be injective on $\{0, \dots, k^2 - 1\}$,

we actually want a function that maps each element $x \in X$ to a specific element ("the color of x") of $\{1, \dots, k\}$. Fortunately, this is easy to achieve by concatenating $h_{p,q}$ with an appropriate function $g\colon \{0, \dots, k^2 - 1\} \to \{1, \dots, k\}$.

In detail, to construct ϕ' from the claim of the theorem, we construct a family of formulas $\phi^g(p, q)$ where p and q are new free variables and the formulas are indexed by all possible functions $g\colon \{0, \dots, k^2 - 1\} \to \{1, \dots, k\}$: In ϕ, replace every occurrence of $C_i x_j$ by the following formula $\pi_i^g(p, q, x_j)$:

$$\bigvee_{y \in \{0,\dots,k^2-1\}, g(y)=i} \exists \hat{k} \exists \hat{y} \big(\textsc{succ}^k(0) = \hat{k} \wedge \textsc{succ}^y(0) = \hat{y} \wedge \rho(\hat{k}, p, q, x_j, \hat{y}) \big)$$

where $\hat{k}$ and $\hat{y}$ are fresh variables that we bind to the numbers k and y (if the universe is large enough). Note that the formula $C_i x_j$ has x_j as a free variable, while $\pi_i^g(p, q, x_j)$ additionally has p and q as free variables. As an example, for the formula $\phi = \exists x(C_2 x \vee \exists y C_5 y)$ we would have $\phi^g = \exists x(\pi_2^g(p, q, x) \vee \exists y \pi_5^g(p, q, y))$. Clearly, each ϕ^g has the property $\operatorname{qr}(\phi^g) = \operatorname{qr}(\phi) + O(1)$.

The desired formula ϕ' is (almost) simply $\bigvee_{g:\{0,\dots,k^2-1\}\to\{1,\dots,k\}} \exists p \exists q(\phi^g(p, q))$. The "almost" is due to the fact that this formula works only for structures with a sufficiently large universe – but by Lemma 2.1 it suffices to consider only this case. Let us prove that for every σ-structure $\mathcal{A}$ with universe $A = \{0, \dots, n - 1\}$ and $n \geq c$ for some to-be-specified constant c, the following two statements are equivalent:

1. There is a k-coloring $\mathcal{B}$ of $\mathcal{A}$ with $\mathcal{B} \models \phi$.
2. $\mathcal{A} \models \bigvee_{g:\{0,\dots,k^2-1\}\to\{1,\dots,k\}} \exists p \exists q(\phi^g(p, q))$.

Let us start with the implication of item 2 to 1. Suppose there is a function $g\colon \{0, \dots, k^2 - 1\} \to \{1, \dots, k\}$ and elements $p, q \in \{0, \dots, n - 1\}$ such that $\mathcal{A} \models \phi^g(p, q)$. We define a partition $A_1 \dot\cup \cdots \dot\cup A_k = A$ by $A_i = \{x \in A \mid g(h_{p,q}(x)) = i\}$. In other words, A_i contains all elements of A that are first hashed to an element of $\{0, \dots, k^2 - 1\}$ that is then mapped to i by the function g. Trivially, the A_i form a partition of the universe A.

Assuming that the universe size is sufficiently large, namely for $k^2 \log_2 n < n$, inside ϕ^g all uses of $\rho(\hat{k}, p, q, x, \hat{y})$ will have the property that $\mathcal{A} \models \rho(\hat{k}, p, q, x, \hat{y})$ if, and only if, $h_{p,q}(x) = \hat{y}$. Clearly, there is a constant c depending only on k such that for all $n > c$ we have $k^2 \log_2 n < n$.

With the property established, we now see that $\pi_i^g(p, g, x_j)$ holds inside the formula ϕ^g if, and only if, the interpretation of x_j is an element of A_i. This means that if we interpret each C_i by A_i, then we get $\mathcal{A} \models \phi(A_1, \dots, A_k)$ and the A_i form a partition of the universe. In other words, we get item 1.

Now assume that item 1 holds, that is, there is a partition $B_1 \dot\cup \cdots \dot\cup B_k = A$ with $\mathcal{A} \models \phi(B_1, \dots, B_k)$. We must show that there are a $g\colon \{0, \dots, k^2 - 1\} \to \{1, \dots, k\}$ and $p, q \in A$ such that $\mathcal{A} \models \phi^g(p, q)$.

At this point, we use the small witness property that we established earlier for the partition. By this property there are pairwise disjoint sets $A_i \subseteq A$ such that, first, $|A_i|$ depends only on ϕ and, second, $\mathcal{A} \models \phi(A_1, \dots, A_k)$. Let $X = \bigcup_{i=1}^k A_i$. Then $|X|$ depends only on ϕ and let s_ϕ be a ϕ-dependent upper bound on this size. By the universal hashing lemma, there are now p and q such that $h_{p,q}\colon \{0, \dots, n - 1\} \to \{0, \dots, s_\phi^2 - 1\}$ is injective on X. But, then, we can set $g\colon \{0, \dots, s_\phi^2 - 1\} \to \{1, \dots, k\}$ to $g(v) = i$ if there is an $x \in A_i$ with $h_{p,q}(x) = v$ and setting $g(v)$ arbitrarily otherwise. Note that this is, indeed, a valid definition of g since $h_{p,q}$ is injective on X.

With these definition, we now define the following sets D_1 to D_k: Let $D_i = \{x \in A \mid g(h_{p,q}(\hat{x})) = i\}$ where $\hat{x}$ is the index of x in A with respect to the ordering (that is, $\hat{x} = |\{y \in A \mid y <^{\mathcal{A}} x\}|$ and for the special case that $A = \{0, \dots, n - 1\}$ and that $<^{\mathcal{A}}$ is

the natural ordering, $\hat{x} = x$). Observe that $D_i \supseteq A_i$ holds for all D_i and that the D_i form a partition of the universe A. By the monotonicity property, $\mathcal{A} \models \phi(A_1, \dots, A_k)$ implies $\mathcal{A} \models \phi(D_1, \dots, D_k)$. However, by definition of the D_i and of the formulas π_i^g, for a sufficiently large universe size n (namely $s_\phi^2 \log_2 n < n$), we now also have $\mathcal{A} \models \phi^g(p, q)$, which in turn implies $\mathcal{A} \models \bigvee_g \exists p \exists q \phi^g$. ◀

In the theorem we assumed that ϕ is a sentence to keep the notation simple, both the theorem and later theorems still hold when $\phi(x_1, \dots, x_n)$ has free variables x_1 to x_n. Then there is a corresponding $\phi'(x_1, \dots, x_n)$ such that first item becomes that for all $\mathcal{A} \in \text{STRUC}[\tau]$ and all $a_1, \dots, a_n \in |\mathcal{A}|$ we have $\mathcal{A} \models \phi'(a_1, \dots, a_n)$ if, and only if, there is a k-coloring $\mathcal{B}$ of $\mathcal{A}$ with $\mathcal{B} \models \phi(a_1, \dots, a_n)$. Note that the syntactic transformations in the theorem do not add dependencies of universal quantifiers on the free variables.

3.2 Formulas With Weak Quantifiers

If one has a closer look at proofs based on color coding, one cannot help but notice that the colors are almost exclusively used to ensure that certain vertices in a structure are distinct from certain other vertices: recall the introductory example $\bigwedge_{j=1}^{k} \exists x \exists y \exists z (Exy \wedge Eyz \wedge Exz \wedge C_{3j-2}x \wedge C_{3j-1}y \wedge C_{3j}z)$, which describes the triangle packing problem when we require that the C_i form a partition of the universe. Since the C_i are only used to ensure that the many different x, y, and z are different, we already rewrote the formula in (4) as $\exists x_1 \cdots \exists x_{3k} \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{j=1}^{k} \exists x \exists y \exists z (Exy \wedge Eyz \wedge Exz \wedge x_{3j-2} = x \wedge x_{3j-1} = y \wedge x_{3j} = z)$. While this rewriting gets rid of the colors and moves us back into the familiar territory of simple first-order formulas, the quantifier rank and the number of variables in the formula have now "exploded" (from the constant 3 to the parameter-dependent value $3k + 3$) – which is exactly what we need to avoid in order to apply Fact 2.2 or Theorem 2.3.

We now define a syntactic property that the x_i have that allows us to remove them from the formula and, thereby, to arrive at a family of formulas of constant quantifier rank. For a (sub)formula α of the form $\forall d(\phi)$ or $\exists d(\phi)$, we say that d *depends* on all free variables in ϕ (at the position of α in a larger formula). For instance, in $Exy \wedge \forall b(Exb \wedge \exists z(Eyz)) \wedge \exists b(Exx)$, the variable b depends on x and y at its first binding ($\forall b$) and on x at the second binding ($\exists b$).

▶ **Definition 3.4.** *We call the leading quantifier in a formula $\exists x(\phi)$ in negation normal form* strong *if*

1. *some universal binding inside ϕ depends on x or*
2. *there is a subformula $\alpha \wedge \beta$ of ϕ such that both α and β contain x in literals that are not of the form $x \neq y$ for some variable y.*

If neither of the above hold, we call the quantifier weak. *The* strong quantifier rank strong-qr(ϕ) *is the quantifier rank of ϕ, where weak quantifiers are ignored;* strong-bound(ϕ) *contains all variables of ϕ that are bound by non-weak quantifiers.*

(Later on we extend the definition to the dual notion of weak *universal* quantifiers, but for the moment let us only call existential quantifiers weak.)

We place a dot on the variables bound by weak quantifiers to make them easier to spot. For example, in $\phi = \exists x \exists y \exists \dot{z} (Rxx\dot{z}\dot{z} \wedge x \neq y \wedge y \neq \dot{z} \wedge Px \wedge \forall w\, Ewyy)$ the quantifier $\exists \dot{z}$ is weak, but neither are $\exists x$ (since x is used in two literals joined by a conjunction, namely in $Rxx\dot{z}\dot{z}$ and Px) nor $\exists y$ (since w depends on y in $\forall w\, Ewyy$). We have $\text{qr}(\phi) = 4$, but strong-qr$(\phi) = 3$, and bound$(\phi) = \{x, y, \dot{z}\}$, but strong-bound$(\phi) = \{x, y\}$.

Admittedly, the definition of weakness is a bit technical, but note that there is a rather simple sufficient condition for a variable x to be weak: If it not used in universal binding

and used in only one literal that is not an inequality, then x is weak. This condition almost always suffices for identifying the weak variables, although there are of course exceptions like $\exists\dot{x}(P\dot{x} \vee Q\dot{x})$.

▶ **Theorem 3.5.** *Let τ be an arithmetic signature. Then for every τ-formula ϕ in negation normal form there is a τ-formula ϕ' such that*

1. *ϕ' is equivalent to ϕ on finite structures,*
2. $\mathrm{qr}(\phi') = 3 \cdot \text{strong-qr}(\phi) + O(\mathrm{arity}(\tau))$*, and*
3. $|\mathrm{bound}(\phi')| = |\text{strong-bound}(\phi)| + O(\mathrm{arity}(\tau))$.

Proof Sketch. Using simple transformations, we can ensure that all weak quantifiers follow in blocks after universal quantifiers. We can also ensure that inequality literals directly follow the blocks of weak quantifiers and are joined by conjunctions. If the inequality literals following a block happen to require that all weak variables from the block are different (that is, if for all pairs $\dot{x}_i$ and $\dot{x}_j$ of different weak variables there is an inequality $\dot{x}_i \neq \dot{x}_j$), then we can remove the weak quantifiers $\exists\dot{x}_i$ and at the (single) place where $\dot{x}_i$ is used, we use a color C_i instead. For instance, if $\dot{x}_i$ is used in the literal $\dot{x}_i = y$, we replace the literal by $C_i y$. If $\dot{x}_i$ is used for instance in $\neg E\dot{x}_i y$, we replace this by $\exists x(C_i x \wedge \neg Exy)$. In this way, for each block we get an equivalent formula to which we can apply Theorem 3.1. A more complicated situation arises when the inequality literals in a block "do not require complete distinctness," but this case can also be handled by considering all possible ways in which the inequalities can be satisfied in parallel. In result, all weak quantifiers get removed and for each block a constant number of new quantifiers are introduced. Since each block follows a different universal quantifier, the new total quantifier rank is at most the strong quantifier rank times a constant factor; and the new number of variables is only a constant above the number of original strong variables. ◀

We already mentioned that the notion of weak existential quantifiers begs a dual: By Theorem 3.5, for $\phi = \exists\dot{x}_1 \cdots \exists\dot{x}_k(\psi)$ there is an equivalent formula ϕ' with $\mathrm{qr}(\phi') = O(\text{strong-qr}(\phi))$. Since, trivially, $\mathrm{qr}(\neg\phi') = \mathrm{qr}(\phi')$, the formula $\neg\phi$ is also equivalent to a formula of quantifier rank $O(\text{strong-qr}(\phi))$. The normal form of $\neg\phi$ starts with $\forall x_1 \cdots \forall x_k$ to which Theorem 3.5 does not apply "at all" – but the dual of the theorem applies, where we call the leading quantifier in a (sub)formula $\forall x(\phi)$ *weak* if no *existential* binding inside ϕ depends on x and in all subformulas of ϕ of the form $\alpha \vee \beta$ at most one of α and β may contain a literal that contains x and is not of the form $x = y$ (note that this is now an equality). More interestingly, we can even show that both kinds of weak quantifiers may be present:

▶ **Theorem 3.6.** *Theorem 3.5 still holds when ϕ may contain both existential and universal weak variables, none of which count towards the strong quantifier rank nor count as strong bound variables.*

Proof Sketch. As in the proof of Theorem 3.5, we syntactically transform ϕ so that the weak existential quantifiers follow strong universal quantifiers in block and – this is new – that the weak universal quantifiers follow strong existential quantifiers. The key observation that makes these transformations possible in the mixed case is that weak existential and weak universal quantifiers commute: For instance, $\exists\dot{x}(\alpha \wedge \forall\dot{y}(\beta)) \equiv \forall\dot{y}(\beta \wedge \exists\dot{x}(\alpha))$ since $\dot{x}$ and $\dot{y}$ cannot depend on one another by the core property of weak quantifiers (α cannot contain $\dot{y}$ and β cannot contain $\dot{x}$). ◀

4 Syntactic Proofs and Natural Problems

The special allure of descriptive complexity theory lies in the possibility of proving that a problem has a certain complexity just by describing the problem in the right way. The "right way" is, of course, a logical description that has a certain syntax (such as having a bounded strong quantifier rank). In the following we present such descriptions for several natural problems and thereby bound their complexity "in a purely syntactic way." First, however, we present "syntactic tools" for describing problems more easily. These tools are built on top of the notion of strong and weak quantifiers.

4.1 Syntactic Tools: New Operators

It is common in mathematical logic to distinguish between the core syntax and additional "shorthands" built on top of the core syntax. For instance, while $\neg$ and $\vee$ are typically considered to be part of the core syntax of propositional logic, the notation $a \to b$ is often seen as a shorthand for $\neg a \vee b$. In a similar way, we now consider the notions of weak variables and quantifiers introduced in the previous section as our "core syntax" and build a number of useful shorthands on top of them. Of course, just as $a \to b$ has an intended semantic meaning that the expansion $\neg a \vee b$ of the shorthand must reflect, the shorthands we introduce also have an intended semantic meaning, which we specify.

As a first example, consider the common notation $\exists^{\geq k} x(\phi(x))$, whose intended semantics is "there are at least k different elements in the universe that make $\phi(x)$ true." While this notation is often considered as a shorthand for $\exists x_1 \cdots \exists x_k \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i=1}^{k} \phi(x_i)$ we will consider it a shorthand for the equivalent, but slightly more complicated formula $\exists \dot{x}_1 \cdots \exists \dot{x}_k \bigwedge_{i \neq j} \dot{x}_i \neq \dot{x}_j \wedge \bigwedge_{i=1}^{k} \exists x(x = \dot{x}_i \wedge \phi(x))$. The difference is, of course, that the strong quantifier rank is now much lower and, hence, by Theorem 3.5 we can replace any occurrence of $\exists^{\geq k} x(\phi(x))$ by a formula of quantifier rank $\mathrm{qr}(\phi) + O(1)$. In all of the following notations, k and s are arbitrary values. The indicated strong quantifier rank for the notation is that of its expansion. The *semantics* describe which structures $\mathcal{A}$ are models of the formula.

▶ **Notation** $(\exists^{\geq k} x(\phi(x)))$. **Strong-qr:** $1 + \text{strong-qr}(\phi)$
Semantics There are k distinct $a_1, \ldots, a_k \in |\mathcal{A}|$ with $\mathcal{A} \models \phi(a_i)$ for all i.
Expansion $\exists \dot{x}_1 \cdots \exists \dot{x}_k \bigwedge_{i \neq j} \dot{x}_i \neq \dot{x}_j \wedge \bigwedge_{i=1}^{k} \exists x(x = \dot{x}_i \wedge \phi(x))$

▶ **Notation** $(\exists^{\leq k} x(\phi(x)))$. **Strong-qr:** $1 + \text{strong-qr}(\phi)$
Semantics There are at most k distinct $a_1, \ldots, a_k \in |\mathcal{A}|$ with $\mathcal{A} \models \phi(a_i)$ for all i.
Expansion $\forall \dot{x}_1 \cdots \forall \dot{x}_{k+1} \bigvee_{i \neq j} \dot{x}_i = \dot{x}_j \vee \bigvee_{i=1}^{k+1} \forall x(x \neq \dot{x}_i \vee \neg\phi(x))$ $(\equiv \neg\exists^{\geq k+1} x(\phi(x)))$

▶ **Notation** $(\exists^{=k} x(\phi(x)))$. **Strong-qr:** $1 + \text{strong-qr}(\phi)$
Semantics There are exactly k distinct $a_1, \ldots, a_k \in |\mathcal{A}|$ with $\mathcal{A} \models \phi(a_i)$ for all i.
Expansion $\exists^{\geq k} x(\phi(x)) \wedge \exists^{\leq k} x(\phi(x))$

The next notation is useful for "binding" a set of vertices to weak or strong variables. The binding contains the allowed "single use" of the weak variables in the sense of Definition 3.4, but they can still be used in inequality literals. Let $\mathring{x}$ indicate that x may be weak or strong.

▶ **Notation** $(\{\mathring{x}_1, \ldots, \mathring{x}_k\} = \{x \mid \phi(x)\})$. **Strong-qr:** $1 + \text{strong-qr}(\phi)$
Semantics Let $a_1, \ldots, a_k \in |\mathcal{A}|$ be the assignments to the $\mathring{x}_i$ (note that they need *not* be distinct). Then $\{a_1, \ldots, a_k\} = \big\{a \in |\mathcal{A}| \;\big|\; \mathcal{A} \models \phi(a)\big\}$ must hold.
Expansion

$$\begin{aligned}
&\textstyle\bigwedge_{i=1}^{k} \exists x\big(x = \mathring{x}_i \wedge \phi(x)\big) \wedge && \text{// ensure } \{\mathring{x}_1, \ldots, \mathring{x}_k\} \subseteq \{x \mid \phi(x)\}\\
&\textstyle\bigvee_{s=1}^{k} \big(\exists^{=s} x(\phi(x)) \wedge && \text{// bind } s \text{ to } |\{x \mid \phi(x)\}|\\
&\quad \textstyle\bigvee_{I \subseteq \{1,\ldots,k\}, |I|=s} \bigwedge_{i,j \in I, i \neq j} \mathring{x}_i \neq \mathring{x}_j\big). && \text{// ensure } |\{\mathring{x}_1, \ldots, \mathring{x}_k\}| \geq s
\end{aligned}$$

The final notation can be thought of as a "generalization of $\exists^{=k}$" where we not only ask whether there are exactly k distinct a_i with a property ϕ, but whether these a_i then also have an *arbitrary* special additional property. Formally, let $Q \subseteq \text{STRUC}[\tau]$ be an arbitrary τ-problem. We write $\mathcal{A}[I]$ for the substructure of $\mathcal{A}$ induced on a subset $I \subseteq |\mathcal{A}|$.

▶ **Notation** ($\text{INDUCED}^{\text{size}=k}\{x \mid \phi(x)\} \in Q$). **Strong-qr:** $1 + \text{strong-qr}(\phi) + \text{arity}(\tau)$
Semantics The set $I = \{a \in |\mathcal{A}| \mid \mathcal{A} \models \phi(a)\}$ has size exactly k and $\mathcal{A}[I] \in Q$.
Expansion Assuming for simplicity that τ contains only E^2 as non-arithmetic predicate:

$$\exists^{=k} x(\phi(x)) \wedge \bigvee\nolimits_{\mathcal{A}\in Q, |\mathcal{A}|=\{1,\dots,k\}} \bigwedge\nolimits_{(i,j)\in E^{\mathcal{A}}} \exists x \exists y(\pi_i(x) \wedge \pi_j(y) \wedge Exy) \wedge \bigwedge\nolimits_{(i,j)\notin E^{\mathcal{A}}} \exists x \exists y(\pi_i(x) \wedge \pi_j(y) \wedge \neg Exy),$$

where $\pi_i(x)$ is a shorthand for $\phi(x) \wedge \exists^{=i-1} z(z < x \wedge \phi(z))$, which binds x to the ith element of the universe with property ϕ.

▶ **Notation** ($\text{INDUCED}^{\text{size}\leq k}\{x \mid \phi(x)\} \in Q$). **Strong-qr:** $1 + \text{strong-qr}(\phi) + \text{arity}(\tau)$
Semantics The set $I = \{a \in |\mathcal{A}| \mid \mathcal{A} \models \phi(a)\}$ has size at most k and $\mathcal{A}[I] \in Q$.
Expansion $\bigvee_{s=0}^{k} \text{INDUCED}^{\text{size}=s}\{x \mid \phi(x)\} \in Q$

4.2 Describing Classical Problems

A *vertex cover* of a graph $G = (V, E)$ is a subset $X \subseteq V$ with $e \cap X \neq \emptyset$ for all $e \in E$. The problem p_k-VERTEX-SET asks whether a graph has a cover X with $|X| \leq k$.

▶ **Theorem 4.1** ([2, 6]). p-VERTEX-COVER $\in$ para-AC^0.

Proof. We describe the problem using a family $(\phi_k)_{k\in\mathbb{N}}$ of constant strong quantifier rank that expresses the well-known Buss kernelization "using logic": Let $\text{HIGH}(x) = \exists^{\geq k+1} y(Exy)$ expresses that x is a *high-degree vertex.* Buss observed that all high-degree vertices must be part of a vertex cover of size at most k. Thus, $h \leq k$ must hold for the unique h with $\exists^{=h} x(\text{HIGH}(x))$. A remaining vertex is *interesting* if it is connected to at least one non-high-degree vertex: $\text{INTERESTING}(x) = \neg\,\text{HIGH}(x) \wedge \exists y(Exy \wedge \neg\,\text{HIGH}(y))$. If there are more than $(k-h)(k+1) \leq k^2 + k$ interesting vertices, there cannot be a vertex cover – and if there are less, the graph induced on the interesting vertices must have a vertex cover of size $k - h$. In symbols: $\phi_k = \bigvee_{h=0}^{k} \big(\exists^{=h} x(\text{HIGH}(x)) \wedge \text{INDUCED}^{\text{size}\leq k^2+k}\{x \mid \text{INTERESTING}(x)\} \in Q_{k-h}\big)$ for $Q_s = \{\mathcal{G} \mid \mathcal{G} \text{ has a vertex cover of size } s\}$. ◀

Hitting sets generalize the notion of vertex covers to hypergraphs (V, E). They are still sets $X \subseteq V$ with $e \cap X \neq \emptyset$ for all $e \in E$. The problem $\text{p}_{k,d}$-HITTING-SET asks whether a hypergraph with $\max_{e\in E} |e| \leq d$ has a hitting set X with $|X| \leq k$. Note that p-VERTEX-COVER is exactly this problem restricted to $d = 2$.

▶ **Theorem 4.2** ([4]). $\text{p}_{k,d}$-HITTING-SET $\in$ para-AC^0.

Proof Sketch. Instead of the Buss kernelization, "using logic" we describe the kernelization presented by us in [4] for the hitting set problem. While this kernelization is *considerably* more complex, it turns out that it can be expressed quite naturally using weak variables. ◀

Next, we show that the result by Flum and Grohe [10] that the model checking problem for first-order logic lies in FPT on structures whose Gaifman graph has bounded degree can be obtained "syntactically." For simplicity, we only consider graphs and let $\text{p}_{\psi,\delta}\text{-MC}(\text{FO}) = \big\{(\mathcal{G}, \text{num}(\psi, \delta)) \;\big|\; \mathcal{G} \in \text{STRUC}[(E^2)], \psi \in \text{FO}, \mathcal{G} \models \psi, \text{max-degree}(\mathcal{G}) \leq \delta\big\}$.

▶ **Theorem 4.3** ([2, 10]). $\mathrm{p}_{\psi,\delta}\text{-MC}(\mathrm{FO}) \in \text{para-AC}^{0\uparrow}$.

Proof Sketch. There is a family $(\phi_{\psi,\delta})_{\psi\in\mathrm{FO},\delta\in\mathbb{N}}$ with a bound on the number of strong variables that describes $\mathrm{p}_{\psi,\delta}$-MC(FO): For fixed ψ and δ, using Gaifman's Theorem, rewrite ψ as $\exists x_1 \cdots \exists x_k \big(\bigwedge_{i\neq j} \gamma_{\mathrm{dist}(x_i,x_j)>2r} \wedge \bigwedge_i \rho(x_i)\big)$ where ρ is r-local. Because of the bounded degree, a ball of radius r can have maximum size δ^r. We can now verify the disjointness of the balls surrounding the x_i by using one weak variable for each element in such a ball. The second part can then be verified by $\textsc{induced}^{\mathrm{size}\leq\delta^r}\{x \mid \gamma_{\mathrm{dist}(x,x_i)\leq r}\} \in \{\mathcal{G} \mid \mathcal{G} \models \rho(x_i)\}$. ◀

In our final example, $\mathrm{td}(H)$ is the *tree depth* of H and $\mathrm{tw}(H)$ is the *tree width* (see the appendix for detailed definitions). A graph $H = (V(H), E(H))$ *embeds into* a graph $G = (V(G), E(G))$ if there is an injective mapping $\iota\colon V(H) \to V(G)$ such that for all $(u, v) \in E(H)$ we have $(\iota(u), \iota(v)) \in E(G)$. Let $\text{p-EMB}_{\mathrm{td}\leq c}$ be $\{(G, \mathrm{num}(H)) \mid \mathrm{td}(H) \leq c$ and H embeds into $G\}$ and define $\text{p-EMB}_{\mathrm{tw}\leq c}$ similarly.

▶ **Theorem 4.4** ([2, 5]). $\text{p-EMB}_{\mathrm{td}\leq c} \in \text{para-AC}^0$ *and* $\text{p-EMB}_{\mathrm{tw}\leq c} \in \text{para-AC}^{0\uparrow}$ *for each* c.

Proof Sketch. For each graph H together with a tree decomposition (T, B) of H, we present a formula $\phi_{H,T,B}$ with

1. $\text{strong-qr}(\phi_{H,T,B}) = O(\mathrm{depth}(T))$ and
2. $|\text{strong-bound}(\phi_{H,T,B})| = O(\mathrm{width}(B))$,

such that for all graphs $\mathcal{G}$ we have $\mathcal{G} \models \phi_{H,T,B}$ if, and only if, H embeds into $\mathcal{G}$. The idea is to use $|H|$ distinct weak variables to bind the embedding and $\mathrm{width}(B) + 1$ strong variables to keep track of the vertices in the bags. Each time a vertex enters the bags for the first time, bind the corresponding weak variable to one of the strong ones. Recycle strong variables when a vertex leaves a bag. Build a nested formula whose structure mirrors the tree decomposition and check for each bag whether the necessary edges are present. ◀

5 Conclusion

In the present paper, we showed how the color coding technique can be turned into a powerful tool for parameterized descriptive complexity theory. This tool allows us to show that important results from parameterized complexity theory – like the fact that the embedding problem for graphs of bounded tree width lies in FPT – follow just from the syntactic structure of the formulas that describe the problem.

In all our syntactic characterizations it was important that variables or color predicates were not allowed to be within a universal scope. The reason was that literals, disjunctions, conjunctions, and existential quantifiers all have what we called the *small witness property,* which universal quantifiers do *not* have. However, there are other quantifiers, from more powerful logics that we did not explore, that also have the small witness property. An example are operators that test whether there is a path of length at most k from one vertex to another for some fixed k: if such a path exists, its vertices form a "small witness." Weak variables may be used inside these operators, leading to broader classes of problems that can be described by families of bounded strong quantifier rank. On the other hand, we cannot add the full transitive closure operator TC (for which it is well-known that FO[TC] = NL) and hope that Theorems 3.1 and 3.5 still hold: If this were the case, we should be able to turn a formula that uses two colors C_1 and C_2 to express that there are two vertex-disjoint paths between two vertices into a FO[TC] formula – thus proving the unlikely result that the NP-hard disjoint path problem is in NL.

Another line of inquiry into the descriptive complexity of parameterized problems was already started in the repeatedly cited paper by Chen et al. [6]: They give first syntactic properties for families of formulas describing weighted model checking problems that imply

membership in para-AC^0. We believe that it might be possible to base an alternative notion of weak quantifiers on these syntactic properties. Ideally, we would like to prove a theorem similar to Theorem 3.5 in which there are just more quantifiers that count as weak and, hence, even more families have bounded strong quantifier rank. This would allow us to prove for even more problems that they lie in FPT just because of the syntactic structure of the natural formula families that describe them.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Max Bannach, Christoph Stockhusen, and Till Tantau. Fast Parallel Fixed-parameter Algorithms via Color Coding. In *Proceedings of the Tenth International Symposium on Parameterized and Exact Computation (IPEC 2015)*, pages 224–235, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.

3 Max Bannach and Till Tantau. Parallel Multivariate Meta-Theorems. In *Proceedings of the Eleventh International Symposium on Parameterized and Exact Computation (IPEC 2016)*, pages 4:1–4:17, 2016. doi:10.4230/LIPIcs.IPEC.2016.4.

4 Max Bannach and Till Tantau. Computing Hitting Set Kernels By AC^0-Circuits. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, pages 9:1–9:14, 2018. doi:10.4230/LIPIcs.STACS.2018.9.

5 Hubie Chen and Moritz Müller. The Fine Classification of Conjunctive Queries and Parameterized Logarithmic Space. *ACM Transactions on Computation Theory*, 7(2):7:1–7:27, 2015. doi:10.1145/2751316.

6 Yijia Chen, Jörg Flum, and Xuangui Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In *Proceedings of the 26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, pages 19:1–19:16, 2017. doi:10.4230/LIPIcs.CSL.2017.19.

7 Yijia Chen and Jörg Flum. Tree-depth, quantifier elimination, and quantifier rank. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 225–234. ACM, 2018. doi:10.1145/3209108.3209160.

8 Bireswar Das, Murali Krishna Enduri, and I. Vinod Reddy. On the Parallel Parameterized Complexity of the Graph Isomorphism Problem. In *Proceedings of the Twelfth International Conference and Workshop on Algorithms and Computation (WALCOM 2018)*, pages 252–264. Springer, 2018. doi:10.1007/978-3-319-75172-6_22.

9 Ronald Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. *Complexity of Computation*, 7:43–74, 1974.

10 Jörg Flum and Martin Grohe. Describing Parameterized Complexity Classes. *Information and Computation*, 187(2):291–319, December 2003. doi:10.1016/S0890-5401(03)00161-5.

11 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006. doi:10.1007/3-540-29953-X.

12 Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection. *Algorithmica*, 52(2):114–132, 2008. doi:10.1007/s00453-007-9008-7.

13 Neil Immerman. $\mathrm{DSPACE}[n^k] = \mathrm{VAR}[k+1]$. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 334–340, 1991. doi:10.1109/SCT.1991.160278.

14 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 789–798, 2018. doi:10.1145/3209108.3209136.

15 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

Bounding Quantum-Classical Separations for Classes of Nonlocal Games

Tom Bannink
CWI, QuSoft, Science Park 123, 1098 XG Amsterdam, Netherlands
bannink@cwi.nl

Jop Briët
CWI, QuSoft, Science Park 123, 1098 XG Amsterdam, Netherlands
j.briet@cwi.nl

Harry Buhrman
CWI, University of Amsterdam & QuSoft, Science Park 123, 1098 XG Amsterdam, Netherlands
buhrman@cwi.nl

Farrokh Labib
CWI, QuSoft, Science Park 123, 1098 XG Amsterdam, Netherlands
labib@cwi.nl

Troy Lee
Centre for Quantum Software and Information, School of Software, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia
troyjlee@gmail.com

Abstract

We bound separations between the entangled and classical values for several classes of nonlocal t-player games. Our motivating question is whether there is a family of t-player XOR games for which the entangled bias is 1 but for which the classical bias goes down to 0, for fixed t. Answering this question would have important consequences in the study of multi-party communication complexity, as a positive answer would imply an unbounded separation between randomized communication complexity with and without entanglement. Our contribution to answering the question is identifying several general classes of games for which the classical bias can not go to zero when the entangled bias stays above a constant threshold. This rules out the possibility of using these games to answer our motivating question. A previously studied set of XOR games, known not to give a positive answer to the question, are those for which there is a quantum strategy that attains value 1 using a so-called Schmidt state. We generalize this class to mod-m games and show that their classical value is always at least $\frac{1}{m} + \frac{m-1}{m} t^{1-t}$. Secondly, for free XOR games, in which the input distribution is of product form, we show $\beta(G) \geq \beta^*(G)^{2^t}$ where $\beta(G)$ and $\beta^*(G)$ are the classical and entangled biases of the game respectively. We also introduce so-called line games, an example of which is a slight modification of the Magic Square game, and show that they can not give a positive answer to the question either. Finally we look at two-player unique games and show that if the entangled value is $1 - \epsilon$ then the classical value is at least $1 - \mathcal{O}(\sqrt{\epsilon \log k})$ where k is the number of outputs in the game. Our proofs use semidefinite-programming techniques, the Gowers inverse theorem and hypergraph norms.

2012 ACM Subject Classification Theory of computation → Quantum communication complexity

Keywords and phrases Nonlocal games, communication complexity, bounded separations, semidefinite programming, pseudorandomness, Gowers norms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.12

Related Version A full version of this paper is available at `https://arxiv.org/abs/1811.11068`.

Funding Tom Bannink, Jop Briët, Harry Buhrman and Farrokh Labib are supported by the NWO Gravitation-grant NETWORKS-024.002.003.
Jop Briët: Supported by a VENI grant.

Harry Buhrman: Supported by the NWO Gravitation-grant QSC-024.003.037, also supported by EU grant QuantAlgo.
Troy Lee: Part of this work was done while at the School for Physical and Mathematical Sciences, Nanyang Technological University and the Centre for Quantum Technologies, Singapore, supported by the Singapore National Research Foundation under NRF RF Award No. NRF-NRFF2013-13.

Acknowledgements We thank Peter Høyer, Serge Massar, and Henry Yuen for useful discussions. We thank Shravas Rao for providing a proof of one of the lemmas.

1 Introduction

The study of multiplayer games has been extremely fruitful in theoretical computer science across diverse areas including the study of complexity classes [3], hardness of approximation [21], and communication complexity [20]. They are also a great framework in which to study Bell inequalities [2] and analyze the nonlocal properties of entanglement. A particularly simple kind of multiplayer game is an XOR game. An XOR game $G = (f, \pi)$ between t-players is defined by a function $f : X_1 \times X_2 \times \cdots \times X_t \to \{0, 1\}$ and a probability distribution π over $X_1 \times \cdots \times X_t$. An input $(x_1, \ldots, x_t) \in X_1 \times \cdots \times X_t$ is chosen by a referee according to π, who then gives x_i to player i. Without communicating, player i then outputs a bit $a_i \in \{0, 1\}$ with the collective goal of the players being that $a_1 \oplus \cdots \oplus a_t = f(x_1, \ldots, x_t)$. In a classical XOR game, the players' strategies are deterministic. In an XOR game with entanglement, players are allowed to share a quantum state and make measurements on this state to inform their outputs.

As players can always win an XOR game with probability at least $\frac{1}{2}$, it is common to study the *bias* of an XOR game, the probability of winning minus the probability of losing. We use $\beta(G)$ to denote the largest bias achievable by a classical protocol for the game G, and $\beta^*(G)$ to denote the best bias achievable by a protocol using shared entanglement for the game G.

Our motivating question in this paper is:

▶ **Question 1.** Is there a family of t-player XOR games $(G_n)_{n\in\mathbb{N}}$ such that $\beta^*(G_n) = 1$ and $\beta(G_n) \to 0$ as $n \to \infty$?

This question has important implications for multi-party communication complexity. For a function $f : X_1 \times \cdots X_t \to \{0, 1\}$, let $R(f)$ denote the t-party randomized communication complexity of f (in the number-in-the-hand model), and let $R^*(f)$ denote the t-party randomized communication complexity of f where the parties are allowed to share entanglement. A positive answer to Question 1 gives a family of functions $(f_n)_{n\in\mathbb{N}}$ with $R^*(f_n) = O(1)$ and $R(f_n) = \omega(1)$, i.e. an unbounded separation between these two communication models.

In the reverse direction, a family of functions $(f_n)_{n\in\mathbb{N}}$ with $R^*(f_n) = O(1)$ and $R(f_n) = \omega(1)$ gives a family of games $G_n = (f_n, \pi_n)$ with $\beta^*(G_n) \geq c$ for some constant c and $\beta(G_n) \to 0$ as $n \to \infty$. Thus there is a very close connection between Question 1 and the existence of an unbounded separation between randomized communication complexity with and without entanglement.

For the two-player case, it is known that the answer to Question 1 is negative. It was observed by Tsirelson [30] that Grothendieck's inequality [15], a fundamental result from Banach space theory, is equivalent to the assertion that $\beta^*(G) \leq K_G \cdot \beta(G)$, where $K_G \leq 1.78\ldots$ [24, 6] is Grothendieck's constant.

Linial and Shraibman [25] and Shi and Zhu [28] realized that the XOR bias of a game (f, π) can be used to lower bound the communication complexity of f, both in the randomized setting and the setting with entanglement. Together with Grothendieck's inequality they

used this to show that $R(f) = O(2^{2R^*(f)})$ for any partial two-party function f. Thus in the two-party case an unbounded communication separation is not possible between the randomized model with and without entanglement. Raz has given an example of a partial function f with $R(f) = 2^{\Omega(R^*(f))}$ [27], thus the upper bound of Linial-Shraibman and Shi-Zhu is essentially optimal.

In the case of three or more parties, Question 1 and the corresponding question of an unbounded separation between the entangled and non-entangled communication complexity models remain open. A striking result of Peréz-García et al. [26] shows that there is no analogue of Grothendieck's inequality in the three-player setting. In particular, they showed that there exists an infinite family of three-player XOR games $(G_n)_{n\in\mathbb{N}}$ with the property that the ratio of the entangled and classical biases of G_n goes to infinity with n. This result was later quantitatively improved by Briët and Vidick [8]. Both results rely crucially on non-constructive (probabilistic) methods, and in both separating examples the entangled bias $\beta^*(G_n)$ also goes to zero with increasing n. These works leave open the question, posed explicitly in [8], of whether there is such a family of games in which the entangled bias does not vanish with n, but instead stays above a fixed positive threshold while the classical bias decays to zero. Crucially, having a separation in XOR bias where $\beta^*(G_n)$ remains constant is what is needed to also obtain an unbounded separation between randomized communication complexity with and without entanglement.

Our contribution to answering Question 1

One approach to Question 1 is to look at different classes of games and identify which ones could possibly lead to a positive answer.

Peréz-García et al. [26] show that in any XOR game where the entangled strategy uses a GHZ state, there is a bounded gap between the classical and entangled bias: namely, the bias with a GHZ state in a t-player XOR game G is at most $K_G(2\sqrt{2})^{t-1}\beta(G)$. This bound is essentially tight as there are examples of t-player XOR games achieving a ratio between the GHZ state bias and classical bias of $\frac{\pi}{2}^t$ [32]. Briët et al. [7] later extended the Grothendieck-type inequality of Peréz-García et al. to a larger class of entangled states called Schmidt states (see Equation 1). Thus any game where there is a perfect strategy where the players share a Schmidt state cannot give a positive answer to Question 1.

Watts et al. [31] recently investigated Question 1 and found that a t-player XOR game G that is symmetric, i.e. invariant under the renaming of players, and where $\beta^*(G) = 1$ always has a perfect entangled strategy where the players share a GHZ state. Thus symmetric games also cannot give a positive answer to Question 1.

We further study games that have a perfect strategy where players share a GHZ or Schmidt state. We do this for a generalization of XOR games called mod m games. In a mod m game the players output an integer between 0 and $m-1$ and the goal is for the sum of the outputs mod m to equal a target value determined by their inputs. We show that the classical advantage over random guessing is at least $\frac{m-1}{m}t^{1-t}$ in any t player mod m game that can be won perfectly by sharing a Schmidt state (see Theorem 2).

We show this by introducing *angle games*, a class of games that can be won perfectly sharing a GHZ state and are the *hardest* of all such games. Thus a classical strategy in an angle game can be used to lower bound the winning probability of any mod m game that has a perfect Schmidt state strategy.

For small values of t we can directly analyze angle games to give bounds that are sometimes tight. One interesting consequence of our result is the following. The Mermin game G is a three-party XOR game where by sharing a GHZ state players can play perfectly,

$\beta^*(G) = 1$, while classically $\beta(G) = \frac{1}{2}$. We show that this is the maximal possible separation of any 3-party XOR game where $\beta^*(G) = 1$ via a GHZ state. In particular, this means that when one looks at the XOR repetition of the Mermin game the classical bias *does not go down at all*.

We rule out other types of games that could positively answer Question 1 as well. A t-player *free* XOR game $G = (f, \pi)$ is a game where π is a product distribution. For such games we show that $\beta(G) \geq \beta^*(G)^{2^t}$, and thus they cannot be used for a positive answer to Question 1.

Another class of XOR games we consider are *line games*, where the questions asked to the players are related by a geometric property. An example of a line game is a slight modification of the Magic Square game [18]. We show that line games cannot give a positive answer to Question 1 either.

Finally, we look at extensions of Question 1 beyond XOR games to more general classes of games like unique games [21], which have been deeply studied because of their application in hardness of approximation. For unique games we show that in fact if there is strategy with entanglement that can win a unique game perfectly, then there is a perfect classical strategy as well. This can be compared with the result of Cleve et al. [10] that if a two-player game with binary outputs has a perfect strategy with entanglement then it also has a perfect classical strategy. More generally, we show that if the winning probability with entanglement is $1 - \epsilon$ in a unique game with k outputs, then there is a classical strategy that wins with probability $1 - C\sqrt{\epsilon \log k}$.

In the next subsections, we discuss our results in more detail.

1.1 Perfect Schmidt strategies for MOD games

A MOD-m game is a generalization of XOR games to non-binary outputs. A nonlocal game is a MOD-m game if the players are required to answer with integers from 0 to $m-1$, and win if and only if the sum of their answers modulo m equals the target value determined by their inputs. We denote the optimal winning probability using classical strategies by $\omega(G)$, and we write $\omega^*(G)$ for the entangled winning probability. Random play in such a game ensures that the players can always win with probability at least $\frac{1}{m}$. As with XOR games, in a MOD-m game one often considers the *bias* given by the maximum amount by which the value can exceed $\frac{1}{m}$, scaled to be in the $[0, 1]$ range. The bias is $\beta(G) = \frac{m}{m-1}(\omega(G) - \frac{1}{m})$, and similar for the entangled version. This generalizes the definition given for XOR games above.

Define a t-partite *Schmidt state* as a t-partite quantum state that can be written in the form

$$|\psi\rangle = \sum_{i=0}^{d-1} c_i |e_i^{(1)}\rangle |e_i^{(2)}\rangle \cdots |e_i^{(t)}\rangle, \tag{1}$$

where $c_i > 0$ and where the $|e_i^{(j)}\rangle$ $(i = 0, 1, ..., d-1)$ are orthogonal vectors in the j-th system. For $t = 2$ any state can be written this way, something commonly known as the Schmidt decomposition. Note that the well-known GHZ state is a Schmidt state where all the c_i are equal to $1/\sqrt{d}$. In the context of nonlocal games, define a *Schmidt strategy* as a quantum strategy that uses (only) a Schmidt state. We say a strategy is *perfect* if it achieves winning probability 1.

We consider t-player MOD-m games for which there is a perfect Schmidt strategy ("perfect Schmidt games") and for such games we give lower bounds on the classical winning probabilities. One particular set of games with this property is described by Boyer [5]. Their

entangled value is 1 but their classical value goes to 0 as the number of players goes to infinity. The authors of [31] define a closely-related class of games called *noPREF games*. This set of games is equal to the set of perfect Schmidt games when $m = 2$ and the distribution on the inputs is uniform. In [31] it is shown that checking whether a game is in this class can be done in polynomial time. Furthermore, for *symmetric* t-player XOR games they show that a game has entangled value 1 if and only if it falls in this class of perfect Schmidt games. They also provide an explicit non-symmetric XOR game with entangled value 1 that is not in this class. We introduce a t-player MOD-m game called the *uniform angle game*, denoted $\mathrm{UAG}_{t,m}$ (defined in the full version of this paper) for which there is a perfect Schmidt strategy and show a lower bound on the classical winning probability.

▶ **Theorem 2.** *Any t-player MOD-m game G with perfect Schmidt strategy satisfies $\omega(G) \geq \omega(\mathrm{UAG}_{t,m})$. Furthermore we have $\beta(\mathrm{UAG}_{t,m}) \geq t^{1-t}$.*

For $t = 3, m = 2$ (3-player XOR games) we have $\omega(\mathrm{UAG}_{3,2}) = 3/4$. In the full version we provide bounds on $\omega(\mathrm{UAG}_{t,m})$ for other values of t, m.

Let the inputs to a game come from a set $X = X_1 \times X_2 \times ... \times X_t$ where X_i is the set of inputs for the i-th player. We say a game is *total* when all elements of X have a non-zero probability of being asked (sometimes also called having *full support*), similar to total functions in the setting of communication complexity. On the other hand, we say that a game has a *promise* on the inputs when it is not total. For the class of perfect Schmidt games we show that total games are trivial.

▶ **Lemma 3.** *When a t-player MOD-m game G with perfect Schmidt strategy is total then $\omega(G) = 1$.*

1.2 Free XOR games

In this subsection we identify two types of games, namely *free games* and *line games*, for which either the ratio of the entangled and classical biases is small, or the entangled bias itself is small. Thus these games will not be able to give a positive answer to Question 1. Free games are a general and natural class of games in which the players' questions are independently distributed. Line games appear to be less studied (see below for their definition), but turn out to be relevant in the context of parallel repetition (also see below). The main idea behind these results is that a large entangled bias implies that the games are in a sense far from random. This is quantified by the magnitude of certain norms of the game tensors. The particular norms of interest here are related to norms used in Gowers' celebrated hypergraph- and Fourier-analytic proofs of Szemerédi's Theorem. A crucial fact of these norms is that they are large if and only if there is "correlation with structure", the opposite of what one would expect from randomness. We show that this structure can be turned into good classical strategies, thus establishing a relationship between the entangled and classical biases.

▶ **Theorem 4** (Polynomial bias relation for free XOR games)**.** *For integer $t \geq 2$ and any free t-player XOR game with entangled bias β, the classical bias is at least β^{2^t}.*

This result may be considered as an analogue of a well-known result on quantum query algorithms for total functions. It is shown in [1] that the bounded-error quantum and classical query complexities of total functions are polynomially related.

1.3 Line games

Line games are not free, but have a simple geometric structure. For a finite field $\mathbb{F}$ of characteristic at least t and positive integer n, a t-player line game is given by a map $\tau : \mathbb{F}^n \to \{0,1\}$. In the game, the referee independently samples two uniformly random points $x, y \in \mathbb{F}^n$ and sends the point $x + (i-1)y$ to the ith player. The players win the game if and only if the XOR of their answers equals $\tau(y)$. In other words, the players' questions correspond to consecutive points (or an arithmetic progression) on a random affine line through $\mathbb{F}^n$ and the winning criterion depends only on the direction of the line. Refer to this as a line games *over* $\mathbb{F}^n$.

A small example of a line game can be obtained from a slight modification of the three-player Magic Square game, which was analyzed in [18]. The line game is played over the plane $\mathbb{F}_3^2$ and the predicate is zero only on the horizontal lines (with $y \in \{(1,0),(2,0)\}$. In the Magic Square game, the referee restricts only to horizontal and vertical lines.[1]

▶ **Theorem 5.** *For any $\varepsilon \in (0,1]$, integer $t \geq 2$ and finite field $\mathbb{F}$ of characteristic at least t, there exists a $\delta(\varepsilon, t, \mathbb{F}) \in (0,1]$ such that the following holds. For any positive integer n and any t-player line game over $\mathbb{F}^n$ with entangled bias ε, the classical bias is at least $\delta(\varepsilon, t, \mathbb{F})$.*

Note that in the above result, the value of the classical bias is independent of the dimension n of the vector space determining the players' question sets.

While it is not relevant to Question 1, the proof techniques used for Theorem 5 allow us to prove a parallel repetition theorem for a class of games that include line games. It is known that the value of free games and so-called anchored games decays exponentially under parallel repetition. Dinur et al. [13] identified a general criterion of multi-player games to behave like this, encompassing free and anchored games. They showed that it is sufficient for a certain graph that can be obtained from a game to be expanding, a well-known pseudorandom property that gives a measure of graph connectivity. Line games do not belong to this class, as their graphs are not even connected. However, we show that if a map $\tau : \mathbb{F}^n \to \{0,1\}^n$ is pseudorandom in a different sense, then a line game defined by τ has exponential decaying value under parallel repetition. More generally, we show that this is the case for a family of XOR games over an arbitrary finite abelian group Γ. These games are given by a positive integer m, a family of affine linear maps $\psi_0, \ldots, \psi_t : \Gamma^m \to \Gamma$ such that no two are multiples of each other, and a "game map" $\rho : \Gamma \to \{0,1\}$. In the game, the referee samples a uniform random element x from Γ^m and sends the group element $\psi_i(x)$ to the ith player. The winning criterion is given by $\rho(\psi_0(x))$. The relevant notion of pseudoranomness is quantified by the Gowers t-uniformity norm of the map $(-1)^\rho : x \mapsto (-1)^{\rho(x)}$, denoted $\|(-1)^\rho\|_{U^t}$.

▶ **Lemma 6.** *Let m, t be positive integers and let Γ be a finite abelian group. Let $\psi_0, \ldots, \psi_t : \Gamma^m \to \Gamma$ be affine linear maps such that no two are multiples of each other and let $\rho : \Gamma \to \{0,1\}$. Let G be the t-player XOR game given by the system $\{\psi_0, \ldots, \psi_t, \rho\}$. Then, for every positive integer k,*

$$\omega(G^k) \leq \Big(\frac{1 + \|(-1)^\rho\|_{U^t}}{2}\Big)^k.$$

[1] Though this is not the typical description of the game, it is easily seen to be equivalent.

1.4 Unique games

We know that the answer to Question 1 is negative in the two-player case, but we can generalize the question by dropping the XOR restriction. The set of XOR games is part of a larger class of games called unique games for which we investigate the relation between classical and entangled values. A two-player nonlocal game is a unique game if for every pair of questions, for every possible answer of the first player there is exactly one answer of the second player that lets them win, and vice versa. Stated differently, for every question there is a matching between the answers of the two players such that only the matching pairs of answers let the player win.

The Unique Games Conjecture (UGC) of Khot [21] states that for any $\epsilon, \delta > 0$, for any $k > k(\epsilon, \delta)$, it is NP-hard to distinguish instances of unique games with winning probability at least $1 - \epsilon$ from those with winning probability at most δ, where k is the number of possible answers. This conjecture has important consequences because it implies several hardness of approximation results. For example, for the Max-Cut problem, Khot et al. [22] showed that the UGC implies that obtaining an approximation ratio better than ≈ 0.878 is NP-hard. Other results include inapproximability for Vertex Cover [23] and graph coloring problems [14].

Our results relate the quantum and classical winning probabilities in the regime of near-perfect play and are based on a result in [9].

▶ **Theorem 7.** *Let $\epsilon \geq 0$. There is an efficient algorithm that, given any two-player unique game with entangled value $1 - \epsilon$, outputs a classical strategy with winning probability at least $1 - C\sqrt{\epsilon \log k}$, where C is a constant independent of the game.*

Note that for $\epsilon = 0$ this means a perfect quantum strategy implies a perfect classical strategy. Furthermore, the above result only beats a trivial strategy when $\epsilon = \mathcal{O}(1/\log k)$.

Work in a similar direction includes [19]. They show that entangled version of the UGC is false, by providing an efficient algorithm that gives an explicit quantum strategy with winning probability at least $1 - 6\epsilon$ when the true entangled value is $1 - \epsilon$. In the classical case, [9] gives an algorithm that outputs a classical strategy with winning probability $1 - \mathcal{O}(\sqrt{\epsilon \log k})$ when the true *classical value* is $1 - \epsilon$. We extend this result by showing that this classical strategy also does the job when, not the classical, but the entangled value is $1 - \epsilon$.

2 Techniques

This section provides an overview of the proof techniques that we employed. We give sketches of the main ideas which are worked out in full detail in later sections.

2.1 Reduction to angle games

To prove Theorem 2 we introduce a new set of t-player MOD-m games that we call *angle games*. We define a particular angle game called the *uniform angle game*, denoted by $\mathrm{UAG}_{t,m}$ and show that it is the hardest of these games. In an angle game, players receive complex phases $e^{i\phi}$ (angles) satisfying a promise, and the winning answer depends only on the product of the inputs $e^{i\phi_1} \cdot e^{i\phi_2} \cdots e^{i\phi_t}$. We prove the theorem by extracting from any perfect Schmidt strategy a set of complex phases that satisfy such a promise, and thereby reducing any such game to the $\mathrm{UAG}_{t,m}$ game. Let us sketch how this is accomplished. Assume that a perfect Schmidt strategy exists, and let $\{P_1^{(j,x_j)}, ..., P_m^{(j,x_j)}\}$ be the projective measurement

done by player j on input x_j so that $P_i^{(j,x_j)}$ corresponds to output i. Now define unitaries $U^{(j,x_j)} = \sum_i \omega_m^i P_i^{(j,x_j)}$, where $\omega_m = e^{2\pi i/m}$ is an m-th root of unity. Since the strategy is perfect we have for every input $(x_1, ..., x_t)$ that

$$\omega_m^{M(x_1,...,x_t)} = \langle\psi|U^{(1,x_1)} \otimes U^{(2,x_2)} \otimes ... \otimes U^{(t,x_t)}|\psi\rangle.$$

Using the definition of a Schmidt state, we show that this equality implies that these unitaries must be of a simple form and their entries satisfy the promise of an angle game. We prove Theorem 2 and Lemma 3 in the full version, where we also provide classical strategies for the uniform angle game and show that these are tight in the case of 3-player XOR games.

2.2 Norming hypergraphs and quasirandomness

Our main tool for proving Theorem 4 is a relation between the entangled and classical biases and a norm on the set of game tensors. For t-tensors, this norm is given in terms of a certain t-partite t-uniform hypergraph H. Recall that such a hypergraph consists of t finite and pairwise disjoint vertex sets $V_1, \ldots, V_t$ and a collection of t-tuples $E(H) \subseteq V_1 \times \cdots \times V_t$, referred to as the edge set of H. For a t-tensor $T \in \mathbb{R}^{n_1 \times \cdots \times n_t}$, the norm has the following form:

$$\|T\|_H = \Big(\mathbf{E}_{\phi_i : V_i \to [n_i]}\Big[\prod_{(v_1,\ldots,v_t)\in E(H)} T\big(\phi_1(v_1), \ldots, \phi_t(v_t)\big)\Big]\Big)^{\frac{1}{|E(H)|}}, \tag{2}$$

where the expectation taken with respect to the uniform distribution over all t-tuples of mappings ϕ_i from V_i to $[n_i]$. Expressions such as (2) play an important role in the context of graph homomorphisms [4]. If T is the adjacency matrix of a bipartite graph with left and right node sets $[n_1]$ and $[n_2]$ respectively, then each product in (2) is 1 if and only if the maps ϕ_1 and ϕ_2 preserve edges.

Criteria for H under which (2) defines a norm or a semi-norm were determined by Hatami [17, 16] and Conlon and Lee [12]. Famous examples of graph norms include the Schatten-p norms for even $p \geq 4$ (in which case H is a p-cycle) and a well-known family of hypergraph norms are the Gowers octahedral norms. The latter were introduced for the purpose of quantifying a notion of quasirandomness of hypergraphs as an important part of Gowers' graph-theoretic proof of Szemerédi's theorem on arithmetic progressions. Having large Gowers norm turns out to imply *correlation with structure*, as opposed to quasirandomness. This is true also for the norm relevant for our setting. In particular, it turns out that the structure with which a game tensor correlates can be turned into a classical strategy for the game. As such, a large norm of the game tensor implies a large classical bias of the game itself. At the same time, we show that the entangled bias is bounded from above by the norm of the game tensor, provided the game is free. Putting these observations together gives the proof of Theorem 4, which we give in the full version of this paper.

The particular hypergraph norm relevant in our setting was introduced in [11] and can be obtained recursively as follows. Starting with a t-partite t-uniform hypergraph H with vertex set $V_1 \cup \cdots \cup V_t$, write $\mathrm{db}_i(H)$ for the t-partite t-uniform hypergraph obtained by making two vertex-disjoint copies of H and gluing them together so that the vertices in the two copies of V_i are identified. We obtain our hypergraph by starting with a single edge $e = (v_1, \ldots, v_t)$ (and vertex sets of size 1), and applying this operation to all parts, forming the hypergraph $\mathrm{db}_1(\mathrm{db}_2(\ldots \mathrm{db}_t(e)))$ with vertex sets of size 2^{t-1} and 2^t edges. The fact that this hypergraph defines a norm via (2) was proved in [12].

2.3 Line games and Gowers uniformity norms

The proof of Theorem 5 is based on two fundamental results from additive combinatorics: the generalized von Neumann inequality and the Gowers Inverse Theorem. The former easily shows that the classical bias of a line game is bounded from above by the Gowers t-uniformity norm of the game map. We show that in fact the same upper bound holds for the entangled bias as well. A large entangled bias thus implies a large uniformity norm for the game map. Analogous to the above-mentioned octahedral norms for tensors, uniformity norms were introduced to quantify a notion of pseudorandomness for bounded maps over abelian groups as an important step in Gowers' other proof of Szemerédi's Theorem, based on higher-order Fourier analysis. The highly non-trivial Gowers Inverse Theorem of Tao and Ziegler [29] establishes that high uniformity norm again implies correlation with structure. Although structure in this context means something quite different than for tensors, we show that it still implies a lower bound on the classical bias. The above observations together prove Theorem 5, details of which can be found in the full version.

2.4 Semidefinite programming relaxation

The proof of Theorem 7 is a small modification of a proof in [9]. They consider a semidefinite programming (SDP) relaxation of the optimization problem for the classical value and then give two algorithms for rounding the result of the SDP to a classical strategy. In the SDP relaxation the objective is to optimize $\mathbb{E}_{x,y}\sum_{i=1}^{k}\langle u_i^{(x)} \mid v_{\pi_{xy}(i)}^{(y)}\rangle$ where $u_i^{(x)}, v_j^{(y)} \in \mathbb{R}^d$ are vectors corresponding to questions x, y and answers i, j. Furthermore, π_{xy} is the matching of correct answers on questions x, y. A classical strategy would correspond to the case where the vectors are integers instead, such that for each x exactly one $u_i^{(x)}$ is equal to 1 and all other $u_i^{(x)}$ are equal to zero and similar for the $v_j^{(y)}$. A quantum strategy also gives rise to a set of vectors, but satisfying different constraints [19]. One of the constraints of the SDP considered in [9] is $0 \leq \langle u_i \mid v_{\pi_{xy}(i)}\rangle \leq |u_i|^2$ which is valid for classical strategies, but in general not for quantum strategies. For our proof, we consider the same SDP but with this constraint dropped. In that case it is also a relaxation for the entangled case and with a few changes one of the rounding algorithms in [9] is also valid when the constraint is dropped. Note that the result only beats a trivial strategy when $\epsilon = \mathcal{O}(1/\log k)$ whereas the other rounding algorithm in [9] is non-trivial for any ϵ. However this other algorithm is more dependent on the extra constraint and it is not clear if it can be dropped there as well.

To get some intuition for the rounding algorithm, we sketch a solution for $\epsilon = 0$ here. In this case one can show that for each question pair x, y the set of vectors $|u_i^{(x)}\rangle$ $(i = 1, ..., k)$ known by the first player is the same set of vectors as the set $|v_i^{(y)}\rangle$ $(i = 1, ..., k)$ known to the second player. In particular, the vector $|u_i^{(x)}\rangle$ is the same as the matching vector $|v_{\pi_{xy}(i)}^{(y)}\rangle$ of the other player. Using shared randomness they can sample a random vector $|g\rangle$ and compute the overlaps $\xi_i^{(x)} = \langle g|u_i^{(x)}\rangle$ and $\xi_i^{(y)} = \langle g|v_i^{(y)}\rangle$ respectively. As they have the same vectors, the players will have the same values for answers in the matching: $\xi_i^{(x)} = \xi_{\pi_{xy}(i)}^{(y)}$. Now both players simply output the answer i for which $|\xi_i^{(x)}|$ (and $|\xi_i^{(y)}|$ for the other player) has the largest value. With probability one this will yield correct answers. For $\epsilon > 0$ the sets of vectors will not be exactly equal and therefore the values $\xi_i^{(x)}, \xi_{\pi_{xy}(i)}^{(y)}$ will be close but not exactly equal. The discrepancy in these values will be bigger for vectors $|u_i^{(x)}\rangle$ with a small norm. In the full version we provide the rounding algorithm in full detail and show how this issue is solved.

References

1 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum Lower Bounds by Polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Earlier version in FOCS'98.

2 J.S. Bell. On the Einstein Podolsky Rosen Paradox. *Physics*, 1(3):195–200, 1964.

3 Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *20th Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 113–131, 1988.

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 Michel Boyer. Extended GHZ n-player games with classical probability of winning tending to 0. *arXiv*, September 2004. `arXiv:quant-ph/0408090`.

6 Mark Braverman, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. The Grothendieck constant is strictly smaller than Krivine's bound. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS '11)*, pages 453–462, 2011.

7 Jop Briët, Harry Buhrman, Troy Lee, and Thomas Vidick. Multipartite entanglement in XOR games. *Quantum Information & Computation*, 13(3-4):334–360, 2013.

8 Jop Briët and Thomas Vidick. Explicit lower and upper bounds on the entangled value of multiplayer XOR games. *Communications in Mathematical Physics*, 321(1):181–207, 2013.

9 Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Near-optimal Algorithms for Unique Games. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 205–214, New York, NY, USA, 2006. ACM. `doi:10.1145/1132516.1132547`.

10 Richard Cleve, Peter Høyer, Benjamin Toner, and John Watrous. Consequences and Limits of Nonlocal Strategies. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 236–249, 2004. `doi:10.1109/CCC.2004.1313847`.

11 David Conlon, Hiêp Hàn, Yury Person, and Mathias Schacht. Weak quasi-randomness for uniform hypergraphs. *Random Structures & Algorithms*, 40(1):1–38, 2012.

12 David Conlon and Joonkyung Lee. Finite reflection groups and graph norms. *Advances in Mathematics*, 315:130–165, 2017.

13 Irit Dinur, Prahladh Harsha, Rakesh Venkat, and Henry Yuen. Multiplayer parallel repetition for expander games. *arXiv preprint*, 2016. `arXiv:1610.08349`.

14 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional Hardness for Approximate Coloring. *SIAM J. Comput.*, 39(3):843–873, 2009. `doi:10.1137/07068062X`.

15 Alexander Grothendieck. Résumé de la théorie métrique des produits tensoriels topologiques (French). *Bol. Soc. Mat. São Paulo*, 8:1–79, 1953.

16 Hamed Hatami. *On generalizations of Gowers norms*. PhD thesis, University of Toronto, 2009.

17 Hamed Hatami. Graph norms and Sidorenko's conjecture. *Israel Journal of Mathematics*, 175(1):125–150, 2010.

18 Tsuyoshi Ito, Hirotada Kobayashi, Daniel Preda, Xiaoming Sun, and Andrew C. C. Yao. Generalized Tsirelson Inequalities, Commuting-Operator Provers, and Multi-prover Interactive Proof Systems. In *Proceedings of the 2008 IEEE 23rd Annual Conference on Computational Complexity*, CCC '08, pages 187–198. IEEE Computer Society, 2008.

19 Julia Kempe, Oded Regev, and Ben Toner. Unique Games with Entangled Provers are Easy. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 457–466, 2008. `doi:10.1109/FOCS.2008.9`.

20 Iordanis Kerenidis, Sophie Laplante, Virginie Lerays, Jérémie Roland, and David Xiao. Lower bounds on information complexity via zero-communication protocols and applications. *SIAM J. Comp.*, 44(5), 2015.

21 Subhash Khot. On the Power of Unique 2-prover 1-round Games. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing (STOC '02)*, STOC '02, pages 767–775, New York, NY, USA, 2002. ACM. doi:10.1145/509907.510017.

22 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs? *SIAM J. Comput.*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

23 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.

24 Jean-Louis Krivine. Sur la constante de Grothendieck. *C. R. Acad. Sci. Paris Sér. A-B*, 284(8):A445–A446, 1977.

25 Nati Linial and Adi Shraibman. Lower bounds in communication complexity based on factorization norms. *Random Structures and Algorithms*, 34:368–394, 2009.

26 David Pérez-García, Michael Wolf, Carlos Palazuelos, Ignacio Villanueva, and Marius Junge. Unbounded violation of tripartite Bell inequalities. *Communications in Mathematical Physics*, 279:455, 2008.

27 Ran Raz. Exponential separation of quantum and classical communication complexity. In *31st annual ACM symposium on theory of computing*, pages 358–367, 1999.

28 Yaoyun Shi and Yufan Zhu. Tensor Norms and the Classical Communication Complexity of Nonlocal Quantum Measurement. *SIAM J. Comput.*, 38(3):753–766, 2008.

29 Terence Tao and Tamar Ziegler. The inverse conjecture for the Gowers norm over finite fields in low characteristic. *Annals of Combinatorics*, 16(1):121–188, 2012.

30 Boris S. Tsirelson. Quantum analogues of the Bell Inequalities. The case of two spatially separated domains. *J. Soviet Math.*, 36:557–570, 1987.

31 Adam Bene Watts, Aram W Harrow, Gurtej Kanwar, and Anand Natarajan. Algorithms, Bounds, and Strategies for Entangled XOR Games. *arXiv preprint*, 2018. arXiv:1801.00821.

32 M. Zukowski. Bell theorem involving all settings of measuring apparatus. *Phys. Lett. A*, 177(290), 1993.

Token Sliding on Split Graphs

Rémy Belmonte
University of Electro-Communications, Chofu, Tokyo, 182-8585, Japan
remybelmonte@gmail.com

Eun Jung Kim
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France
eun-jung.kim@dauphine.fr

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France
michail.lampis@lamsade.dauphine.fr

Valia Mitsou
Université Paris-Diderot, IRIF, CNRS, 75205, Paris, France
vmitsou@liris.cnrs.fr

Yota Otachi
Kumamoto University, Kumamoto, 860-8555, Japan
otachi@cs.kumamoto-u.ac.jp

Florian Sikora
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France
florian.sikora@dauphine.fr

Abstract

We consider the complexity of the Independent Set Reconfiguration problem under the Token Sliding rule. In this problem we are given two independent sets of a graph and are asked if we can transform one to the other by repeatedly exchanging a vertex that is currently in the set with one of its neighbors, while maintaining the set independent. Our main result is to show that this problem is PSPACE-complete on split graphs (and hence also on chordal graphs), thus resolving an open problem in this area.

We then go on to consider the c-Colorable Reconfiguration problem under the same rule, where the constraint is now to maintain the set c-colorable at all times. As one may expect, a simple modification of our reduction shows that this more general problem is PSPACE-complete for all fixed $c \geq 1$ on chordal graphs. Somewhat surprisingly, we show that the same cannot be said for split graphs: we give a polynomial time ($n^{O(c)}$) algorithm for all fixed values of c, except $c = 1$, for which the problem is PSPACE-complete. We complement our algorithm with a lower bound showing that c-Colorable Reconfiguration is W[2]-hard on split graphs parameterized by c and the length of the solution, as well as a tight ETH-based lower bound for both parameters.

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

Keywords and phrases reconfiguration, independent set, split graph

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.13

Funding Supported by JSPS and MAEDI under the Japan-France Integrated Action Program (SAKURA) Project GRAPA 38593YJ, by FMJH program PGMO and EDF via project 2016-1760H/C16/1507 "Stability versus Optimality in Dynamic Environment Algorithmics" and projects "ESIGMA" (ANR-17-CE23-0010) and "HOSIGRA" (ANR-17-CE40-0022), and by JSPS KAKENHI Grant Numbers JP18K11157, JP18K11168, JP18K11169, JP18H04091.

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 13; pp. 13:1–13:17

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

1 Introduction

A reconfiguration problem is a problem of the following type: we are given an instance of a decision problem, two feasible solutions S, T, and a local modification rule. The question is whether S can be transformed to T by repeated applications of the modification rule in a way that maintains the solution feasible at all times. Due to their numerous applications, reconfiguration problems have attracted much interest in the literature, and reconfiguration versions of standard problems (such as Satisfiability, Dominating Set, and Independent Set) have been widely studied (see the surveys [10, 19] and the references therein).

Among reconfiguration problems on graphs, Independent Set Reconfiguration is certainly the most well-studied. The complexity of this problem depends heavily on the rule specifying the allowed reconfiguration moves. The main reconfiguration rules that have been studied for Independent Set Reconfiguration are Token Addition & Removal (TAR) [16, 18], Token Jumping (TJ) [2, 3, 12, 13, 14], and Token Sliding (TS) [1, 5, 6, 8, 11, 17]. In all rules, we are required to keep the current set independent at all times. TAR allows us to add or remove any vertex in the current set, as long as the set's size is always higher than a predetermined threshold. TJ allows to exchange any vertex in the set with any vertex outside it (thus keeping the size of the set constant at all times). Finally, under TS, we are allowed to exchange a vertex in the current independent set with one of its neighbors, that is, we are allowed to perform a TJ move only if the two involved vertices are adjacent.

The Independent Set Reconfiguration problem has been intensively studied under all three rules. Because the problem is PSPACE-complete in general for all three rules [16], this has motivated the study of its complexity in restricted classes of graphs, with an emphasis on graphs where Independent Set is polynomial-time solvable, such as chordal graphs and bipartite graphs. By now, many results of this type have been discovered (see Table 1 for a summary).

Our first, and main, focus of this paper is to concentrate on a case of this problem which has so far remained elusive, namely, the complexity of Independent Set Reconfiguration on chordal graphs under the TS rule. This case is of particular interest because it is one of the few cases where the problem is known to be tractable under both TAR and TJ. Indeed, Kamiński, Medvedev, and Milanič [16] showed that under these two rules Independent Set Reconfiguration is polynomial-time solvable on even-hole-free graphs, a class that contains chordal graphs. In the same paper they explicitly asked as an open question if the same problem is tractable on even-hole-free graphs under TS ([16, Question 2]).

This question was then taken up by Bonamy and Bousquet [1] who made some progress by showing that Independent Set Reconfiguration under TS is polynomial-time solvable on *interval graphs*, an important subclass of chordal graphs. They also gave some first evidence that it may be hard to obtain a similarly positive result for chordal graphs by showing that a related problem, the problem of determining if *all* independent sets of the same size can be transformed to each other under TS, is coNP-hard on split graphs, another subclass of chordal graphs. Note, however, that this is a problem that is clearly distinct from the more common reconfiguration problem (which asks if two *specific* sets are reachable from each other), and that the coNP-hardness is not tight, since the best known upper bound for this problem is also PSPACE.

The complexity of Independent Set Reconfiguration under TS on split and chordal graphs has thus remained as an open problem. Our first, and main, contribution in this paper is to settle this problem by showing that the problem is PSPACE-complete already on split graphs (Theorem 9), and therefore also on chordal and even-hole-free graphs.

Table 1 Complexity of Independent Set Reconfiguration on some graph classes.

	Independent Set Reconfiguration	
	TS	TJ/TAR
perfect	PSPACE-complete [16]	
even-hole-free	PSPACE-complete (Theorem 9)	P [16]
chordal	PSPACE-complete (Theorem 9)	P (even-hole-free)
split	PSPACE-complete (Theorem 9)	P (even-hole-free)
interval	P [1]	P (even-hole-free)
bipartite	PSPACE-complete [17]	NP-complete [17]

c-Colorable Reconfiguration. A natural generalization of Independent Set Reconfiguration was recently introduced in [15]: in c-Colorable Reconfiguration we are given a graph $G = (V, E)$ and two sets $S, T \subseteq V$, both of which induce a c-colorable graph. The question is whether S can be transformed to T (under any of the previously mentioned rules) in a way that maintains a c-colorable graph at all times. Clearly, $c = 1$ is the case of Independent Set Reconfiguration. It was shown in [15] that this problem is already PSPACE-complete on split graphs under all three rules, when c is part of the input. It was thus posed as an open question what is the complexity of the same problem when c is fixed. Some first results in this direction were given in the form of an $n^{O(c)}$ (XP) algorithm that works for split graphs under the TAR and TJ rules (but not TS). Motivated by this work, the second area of focus of this paper is to investigate how the hardness of 1-Colorable Reconfiguration for split graphs established in Theorem 9 extends to larger, but fixed c.

Our first contribution in this direction is to show that, for chordal graphs, c-Colorable Reconfiguration under TS is PSPACE-complete for any fixed $c \geq 1$. This is, of course, not surprising, as the problem is PSPACE-complete for $c = 1$; indeed, the reduction we present in Theorem 10 is a tweak of the construction of Theorem 9 that increases c.

What is perhaps more surprising is that we show (under standard assumptions) that, even though Theorem 9 establishes hardness for $c = 1$ on split graphs, a similar tweak cannot establish hardness for higher c on the same class for TS. Indeed, we provide an algorithm which solves TS c-Colorable Reconfiguration in split graphs in time $n^{O(c)}$ for any c *except* $c = 1$. Thus, Independent Set Reconfiguration turns out to be the *only* hard case of c-Colorable Reconfiguration for split graphs under TS. Since the $n^{O(c)}$ algorithm of [15] for TAR/TJ reconfiguration of split graphs works for all fixed c, it thus seems that this anomalous behavior is peculiar to the Token Sliding rule.

Finally, we address the natural question of whether one can improve this $n^{O(c)}$ algorithm, by showing that the problem is W[2]-hard parameterized by c and the length of the solution ℓ for all three rules. This is in a sense doubly tight, since in addition to our algorithm and the algorithm of [15] which run in $n^{O(c)}$, it also matches the trivial $n^{O(\ell)}$ algorithm which tries out all solutions of length ℓ. More strongly, under the ETH our reduction implies that the problem cannot be solved in $n^{o(c+\ell)}$ meaning that these algorithms are in a sense "optimal".

2 Definitions

We use standard graph-theoretic terminology. For a graph $G = (V, E)$ and a set $S \subseteq V$ we use $G[S]$ to denote the graph induced by S. A graph is chordal if it does not contain a k-vertex cycle C_k as an induced subgraph for any $k > 3$. A graph is split if its vertex set can be partitioned into two sets K, I such that K induces a clique and I induces an

independent set. It is a well-known fact that split graphs are chordal, and it is easy to see that both classes are closed under induced subgraphs. We use $\chi(G), \omega(G)$ to denote the chromatic number and maximum clique size of a graph G respectively. It is known that, because chordal graphs are perfect, if G is chordal then $\chi(G) = \omega(G)$ [21]. We also recall that a graph G is chordal if and only if every induced subgraph of G contains a simplicial vertex, where a vertex is simplicial if its neighborhood is a clique.

Let $G = (V, E)$ be a graph and $c \geq 1$ an integer. Given two sets $S, T \subseteq V$ such that $\chi(G[S]), \chi(G[T]) \leq c$, we say that S can be c-transformed into T by one token sliding (TS) move if $|T| = |S|$ and there exist $u, v \in V$ with $(u, v) \in E$ such that $\{u\} = T \setminus S$, $\{v\} = S \setminus T$. One easy way to think of TS moves is by picturing the elements of the current set S as tokens placed on the vertices of the graph, and a single move as "sliding" a token along an edge (hence the name Token Sliding).

We say that S is c-reachable from T, or that S can be c-transformed into T, by a sequence of TS moves if there exists a sequence of sets $I_0, I_1, \ldots, I_\ell$, with $I_0 = S, I_\ell = T$ and for each $i \in \{0, \ldots, \ell - 1\}$, $\chi(G[I_i]) \leq c$ and I_i can be c-transformed into I_{i+1} by one TS move. We will simply say that S can be transformed into T or that S is reachable from T, if S, T are independent sets and S can be 1-transformed into T. We focus on the following problems.

▶ **Definition 1.** *In* c-Colorable Reconfiguration *we are given a graph* $G = (V, E)$ *and two sets* $S, T \subseteq V$ *with* $|S| = |T|$ *and* $\chi(G[S]), \chi(G[T]) \leq c$. *We are asked if* S *can be* c*-transformed into* T. Independent Set Reconfiguration *is the special case of* c-Colorable Reconfiguration *where* $c = 1$.

In addition to TS moves we will consider Token Jumping (TJ) and Token Addition & Removal (TAR) moves. A TJ move is the same as a TS move except that the two vertices u, v are not required to be adjacent. Two c-colorable sets S, T are reachable with one TAR move with threshold k if $|S|, |T| \geq k$ and $|(S \setminus T) \cup (T \setminus S)| = 1$. We note here that, because our main focus in this paper is the TS rule, whenever we refer to a transformation without explicitly specifying under which rule this transformation is performed the reader may assume that we are referring to the TS rule.

We assume that the reader is familiar with basic complexity notions such as the class PSPACE [20], as well as basic notions in parameterized complexity, such as the class W[2] (see e.g. [4]). In Theorem 9 we will perform a reduction from the PSPACE-complete NCL (non-deterministic constraint logic) reconfiguration problem introduced by Demaine and Hearn in [8] (see also [7, 9]). Let us recall this problem. In the NCL reconfiguration problem we are given as input a graph $G = (V, E)$, whose edge set is partitioned into two sets, R (red) and B (blue). We consider blue edges as edges of weight 2 and red edges as edges of weight 1. A valid configuration of G is an orientation of all the edges with the property that all vertices have weighted in-degree at least 2. In the NCL configuration-to-configuration problem we are given two valid orientations of G, D and D', and are asked if there is a sequence of valid orientations $D_0, D_1, \ldots, D_t$ such that $D = D_0, D' = D_t$ and for all $i \in \{0, \ldots, t - 1\}$ we have that D_i, D_{i+1} agree on all edges except one. We recall the following theorem:

▶ **Theorem 2** (Corollary 6 of [8]). *The NCL configuration-to-configuration problem is PSPACE-complete even if all vertices of* G *have degree exactly three and, moreover, even if all vertices belong in one of the following two types: OR vertices, which are vertices incident on exactly three blue edges and no red edges; and AND vertices which are vertices incident on two red edges and one blue edge.*

3 Token Sliding on Split Graphs is PSPACE-complete

The main result of this section is that Independent Set Reconfiguration is PSPACE-complete under the TS rule when restricted to split graphs.

Overview of the proof

Our proof is a reduction from the NCL (non-deterministic constraint logic) reconfiguration problem of Theorem 2. The first step of our proof is a relatively straightforward reduction from the NCL reconfiguration problem to token sliding on split graphs. Its main idea is roughly as follows: for each edge $e = (u, v)$ of the original graph we construct two selection vertices e_u, e_v in the independent set of our split graph. The idea is that at each point exactly one of the two will contain a token (i.e. will belong in the current independent set), hence our independent set will in a natural way represent an orientation of the original graph. In order to allow a single reconfiguration step to take place we add for each pair of selection vertices e_u, e_v one or two "gate" vertices (depending on the color of e), which are common neighbors of e_u, e_v and belong in the clique. The idea is that a single re-orientation step would, for example, take a token from e_u, slide it to a gate vertex connected to the pair e_u, e_v, and then slide it to e_v: this sequence would represent re-orienting e from u to v. In order to simulate the in-degree constraint we add edges between each selection vertex e_u and gate vertices corresponding to edges incident on *the other* endpoint of e, since keeping a token on e_u represents an orientation of e towards u, which makes it harder to re-orient the edges incident on the other endpoint of e.

The above sketch captures the basic idea of our reduction, except for one significant obstacle. The correspondence between orientations and independent sets is only valid if we can guarantee that no intermediate independent set will "cheat" by, for example, placing tokens on both e_u and e_v. Since we have added edges from e_u, e_v to gate vertices that correspond to other edges (in order to simulate the interaction between edges in the NCL instance), nothing prevents a reconfiguration solution from using these edges to slide a token from one selection pair to another. The main problem thus becomes enforcing consistency, or in other words forcing the solution sequence to only use the appropriate gate vertices to slide tokens as intended. This is handled in the second step of our reduction which, given the split graph construction sketched above, makes a large number of copies and connects them appropriately in a way that the only feasible token sliding solutions are indeed those that correspond to valid orientations of the original graph.

In the remainder of this section we use the following notation: $G = (V, E)$, where $E = R \cup B$, is the graph supplied with the initial NCL reconfiguration instance and D, D' are the initial and target orientations; $G_b = (V_b, E_b)$ is the "basic" split graph of our construction in the first step and S, T the independent sets of G_b for which we need to decide reachability; and $G_f = (V_f, E_f)$ is the split graph of our final token sliding instance with S_f, T_f being its corresponding independent sets.

Before we proceed, let us first slightly edit our given NCL reconfiguration instance. We will now allow some vertices to have degree two and call these vertices COPY vertices. Using these we can force the OR vertices to become an independent set.

▶ **Lemma 3.** *NCL reconfiguration remains PSPACE-complete on graphs where (i) all vertices are either AND vertices (two incident red edges, one incident blue edge), OR vertices (three incident blue edges), or COPY vertices (two incident blue edges) (ii) every blue edge is incident on exactly one COPY vertex.*

Proof. For every blue edge $e = (u, v) \in B$ in the original graph we delete this edge from the graph, introduce a new COPY vertex w, and connect w to u, v with blue edges. It is not hard to see that this transformation does not change the type of any original vertex or the answer to the reconfiguration problem. ◀

First Step of the Construction

We assume (Lemma 3) that in the given graph G we have three types of vertices (AND, OR, COPY) and that each blue edge is incident on one COPY vertex. Let us now describe the construction of G_b.

1. For each $e = (u, v) \in R$ we construct two selector vertices e_u, e_v and one gate vertex g_e.
2. For each $e = (u, v) \in B$ we construct two selector vertices e_u, e_v and two gate vertices $g_{e,1}, g_{e,2}$.
3. For each edge $e = (u, v) \in R$ we connect g_e to both e_u, e_v. For each edge $e = (u, v) \in B$ we connect both $g_{e,1}, g_{e,2}$ to both e_u, e_v. We call the edges added in this step gate edges.
4. For each AND vertex u, such that $e = (u, v_1) \in B$ and $f = (u, v_2) \in R$, $h = (u, v_3) \in R$ we add the following edges: $(e_{v_1}, g_f), (e_{v_1}, g_h), (f_{v_2}, g_{e,1}), (f_{v_2}, g_{e,2}), (h_{v_3}, g_{e,1}), (h_{v_3}, g_{e,2})$ (see Figure 1). In other words, for each edge involved in this part we connect the selector which represents its other endpoint (not u) to the gate vertices of edges that should be unmovable if this edge is not oriented towards u.
5. For each OR vertex u such that $e = (u, v_1), f = (u, v_2), h = (u, v_3) \in B$ we add the following edges: $(e_{v_1}, g_{f,1}), (e_{v_1}, g_{h,1}), (e_{v_2}, g_{e,1}), (e_{v_2}, g_{h,2}), (e_{v_3}, g_{e,2}), (e_{v_3}, g_{f,2})$. In other words, we connect the selector vertex for each v_i to a distinct gate of the edges $(u, v_j), (u, v_k)$, for i, j, k distinct. Informally, this makes sure that if two of the edges are oriented away from u the third edge is stuck, but if at most one is oriented away from u the other edges have a free gate.
6. For each COPY vertex u such that $e = (u, v_1), f = (u, v_2) \in B$ we add the following edges: $(e_{v_1}, g_{f,1}), (e_{v_1}, g_{f,2}), (f_{v_2}, g_{e,1}), (f_{v_2}, g_{e_2})$. In other words, we connect the selector vertex for v_1 in a way that blocks the movement of the token from f_u, and similarly for v_2.
7. We connect all gate vertices into a clique to obtain a split graph. Note that the remaining vertices (that is, the selector vertices e_v) form an independent set.

We now construct two independent sets S, T of G_b in the natural way: given an orientation D, for each $e = (u, v)$ we place e_u in S if and only if D orients e towards u; we construct T from D' in the same way. This completes the basic construction.

Before proceeding, let us make some basic observations regarding the neighborhoods of gate vertices of the graph G_b. We have the following:

- If $e = (u, v) \in R$, let u', v' be vertices of G such that $f = (u, u') \in B$, $h = (v, v') \in B$ (that is, u', v' are the second endpoints of the blue edges incident on u, v). We have that $N(g_e) = \{e_u, e_v, f_{u'}, h_{v'}\}$.
- If $e = (u, v) \in B$, u is a COPY vertex and v is an AND vertex, let $f = (u, u') \in B$ be the other edge incident on u, and $h = (v, v'), \ell = (v, v'') \in R$ be the other two edges incident on v. Then $N(g_{e,1}) = N(g_{e,2}) = \{e_u, e_v, f_{u'}, h_{v'}, \ell_{v''}\}$.
- If $e = (u, v) \in B$, u is a COPY vertex and v is an OR vertex, let $f = (u, u') \in B$ be the other edge incident on u, and $h = (v, v'), \ell = (v, v'') \in B$ be the other two edges incident on v. Then one of the vertices $g_{e,1}, g_{e,2}$ has neighbors $\{e_u, e_v, f_{u'}, h_{v'}\}$ and the other has neighbors $\{e_u, e_v, f_{u'}, \ell_{v''}\}$.

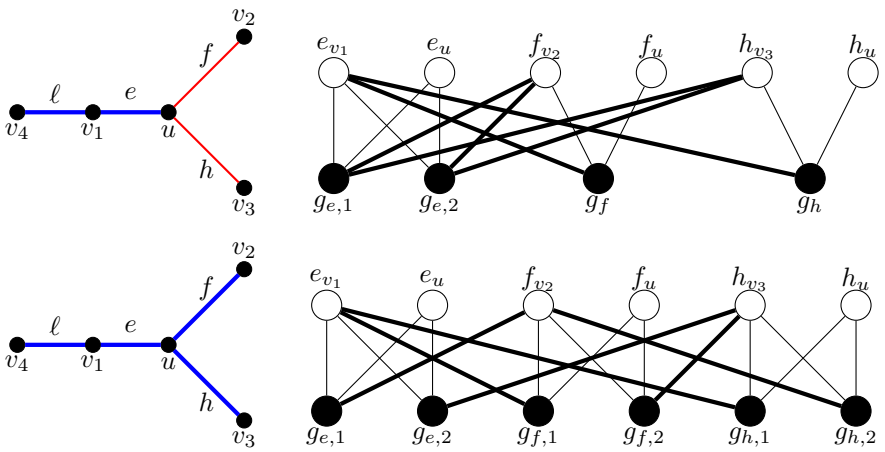


Figure 1 Construction when u is an AND vertex (top) or an OR vertex (bottom). In both cases v_1 is a COPY vertex. The part of the construction corresponding to ℓ is not drawn: ℓ_{v_4} would be a common neighbor of $g_{e,1}, g_{e,2}$ and e_u would be a common neighbor of $\ell_{e,1}, \ell_{e,2}$. Edges connecting selector vertices to their corresponding gates are drawn thinner for readability. On the right, black (gate) vertices are connected in a clique.

We are now ready to show that if we only consider "consistent" configurations in G_b, then the new instance simulates the original NCL reconfiguration problem.

▶ **Lemma 4.** *There is a valid reconfiguration of the NCL instance given by G, D, D' if and only if there exists a valid reconfiguration under the TS rule from S to T in G_b such that no independent set of the reconfiguration sequence contains both e_u, e_v for any $e = (u, v) \in E$.*

Proof. Since G_b is a split graph, any independent set contains at most one vertex from the clique made up of the gate vertices. We will call an independent set that contains no gate vertices a "main" configuration. Furthermore, for main configurations that also obey the restrictions of the lemma (i.e. do not contain both e_u, e_v for any $e \in E$), we observe that there is a natural one-to-one correspondence with the set of orientations of G: an edge $e = (u, v)$ is oriented towards u if and only if e_u is in the independent set. (We implicitly use the fact that the number of tokens is $|E|$, therefore for each pair e_u, e_v exactly one vertex has a token in such a main configuration).

Suppose now that we have two consecutive valid orientations D_i, D_{i+1} in the reconfiguration sequence of G such that D_i, D_{i+1} differ only on the edge $e = (u, v)$, which D_i orients towards u. We want to show that the sets I_i, I_{i+1} obtained using the correspondence above from D_i, D_{i+1} can be obtained from each other with a pair of sliding token moves. Indeed, the sets I_i, I_{i+1} are identical except that $\{e_u\} = I_i \setminus I_{i+1}$ and $\{e_v\} = I_{i+1} \setminus I_i$. We would like to slide the token from e_u to e_v using a gate vertex adjacent to both vertices.

First, assume that $e \in R$, so there exists a single gate vertex g_e. Furthermore, u, v are both AND vertices. Since both D_i, D_{i+1} are valid configurations, in both configurations the blue edges incident on u, v are oriented towards these two vertices. As a result g_e has no neighbor in I_i.

Second, suppose $e = (u, v) \in B$ and one of u, v is a COPY vertex. If e is incident on an AND vertex, because both D_i, D_{i+1} are valid and agree on all edges except e we have that both red edges incident on the AND vertex are oriented towards it in both configurations. Similarly, the second blue edge incident on the COPY endpoint of e is oriented towards it in both configurations. We therefore observe that neither $g_{e,1}$, nor $g_{e,2}$ has a neighbor in I_i except e_u, so we can safely slide $e_u \to g_{e,1} \to e_v$.

Similarly, for the last case, suppose that $e = (u, v) \in B$ and one of the endpoints of e is an OR vertex, while the other is a COPY vertex. Again, because D_i, D_{i+1} are both valid and only disagree on e, at least one of the blue edges incident on the OR vertex (other than e) is oriented towards it in both configurations. As before, the second blue edge incident on the COPY vertex is oriented towards it in both configurations. Therefore, one of $g_{e,1}, g_{e,2}$ has no neighbor in I_i except e_u, so we can safely slide the token from e_u to e_v with two moves.

To complete the proof, we need to show that if we have a valid token sliding reconfiguration sequence, this gives a valid reorientation sequence for G. The main observation now is that in a shortest token sliding solution that obeys the properties of the lemma, a token that slides out of e_u must necessarily in the next move slide into e_v, where $e = (u, v) \in E$. To see this, observe that because of the requirement that the set does not contain both selector vertices of any edge, the tokens found on other selector vertices dominate all gate vertices except those corresponding to e. Since we can neither repeat configurations, nor add a second token to the clique made up of gate vertices, the next move must slide the token to the other selector vertex.

To see that the orientation sequence obtained through the natural translation of main configurations is valid, consider two consecutive main configurations I_i, I_{i+1} in the token sliding solution, such that the corresponding orientations are D_i, D_{i+1}, and D_i is valid. We will show that D_{i+1} is also valid. Suppose that D_{i+1} differs from D_i in the edge $e = (u, v)$ which is oriented towards u in D_i (it is not hard to see that D_i, D_{i+1} cannot differ in more than one edge). Thus, I_i is transformable in two moves to I_{i+1} by sliding e_u to a gate corresponding to e and then to e_v. If e is a red edge, this means that in D_i both blue edges incident on u, v are directed towards u, v, so the reorientation is valid. If e is blue, we first assume that u is a COPY vertex. Since a gate corresponding to u is free, the other blue edge incident on u is oriented towards u in D_i and we have a valid move. Finally, if e is blue and u is an OR vertex, we conclude that, since at least one gate from $g_{e,1}, g_{e,2}$ is available in I_i, at least one of the two other blue edges incident on u is directed towards u in D_i and we have a valid move. ◀

Second Step: Enforcing Consistency

We will now construct a graph G_f that will function in a way similar to the graph we have already constructed but in a way that enforces consistency. Let $G_b = (V_b, E_b)$ be the graph constructed in the first step of our reduction, and let $E_g \subseteq E_b$ be the set of gate edges, that is, the set of edges that connect the selector vertices for an edge e to the corresponding gate(s).

Let $m := |E|$ and $C := m + 4$. We first take C disjoint copies of $G_b = (V_b, E_b)$ and for a vertex $v \in V_b$ we will use the notation v^i, where $1 \leq i \leq C$ to denote the vertex corresponding to v in the i-th copy. Then, for every edge $(u, v) \in E_b \setminus E_g$ (every non-gate edge) and for all $i \neq j \in \{1, \ldots, C\}$ we add the edge (u^i, v^j). This completes the construction of G_f and it is not hard to see that the graph is split, as the C copies of the clique of G_b form a larger clique. To complete our instance let us explain how to translate an independent set of G_b that contains no vertices of the clique to an independent set of G_f: we do this in the natural way by including in the new independent set all C copies of vertices of the original independent set. Since both the initial and final independent sets in our first construction use no vertices in the clique, we have in this way two independent sets of size mC in the new graph, and thus a valid Token Sliding instance. Let S, T be the two independent sets of G_b we are asked to transform and S_f, T_f the corresponding independent sets of G_f.

We first show that if we have a solution for reconfiguration in G_b then we have a solution for reconfiguring the sets in the new graph.

▶ **Lemma 5.** *Let I_1, I_2 be two independent sets of G_b of size m that use no vertices of the clique, respect the conditions of Lemma 4, and can be transformed to one another by two sliding moves. Then the independent sets I'_1, I'_2 which are obtained in G_f by including all copies of vertices of I_1, I_2 respectively can be transformed into one another by a sequence of $2C$ TS moves.*

Proof. Each of I_1, I_2 uses exactly one of the vertices e_u, e_v, for each edge $e = (u, v) \in E$, because of their size, the fact that they contain no vertex of the clique, and the fact that neither contains both e_u, e_v for any edge $e = (u, v) \in E$ (this is the condition of Lemma 4). If I_1 can be transformed into I_2 with two sliding moves, the first move takes a token from an independent set vertex, say e_u and moves it to the clique and the second moves the same token to e_v. Since I_1 contains a token on each pair of selector vertices, the only vertex of the clique on which the token can be moved is a gate vertex corresponding to e, say g_e (if e is red) or $g_{e,1}$ (if e is blue). We now observe that if g_e (or similarly $g_{e,1}$) is available in I_1 (that is, it has no neighbors in I_1 besides e_u), then the same is true for g_e^i for all $i \in \{1, \ldots, C\}$ in I'_1. To see this, note that the neighbors of g_e^i are, e_u^i, e_v^i, and, for each $v \in N(g_e)$ all the vertices v^j for $j \in \{1, \ldots, C\}$. Since none of the neighbors of g_e is in I_1, g_e^i is available. We therefore slide, one by one, a token from e_u^i to g_e^i and then to e_v^i, for all $i \in \{1, \ldots, C\}$. ◀

Now, for the more involved direction of the reduction we first observe that it is impossible for a reconfiguration to arrive at a situation where the solution is highly irregular, in the sense that, for an edge $e = (u, v)$ we have multiple tokens on copies of both e_u and e_v.

▶ **Lemma 6.** *Let S_f be the initial independent set constructed in our instance and S' be an independent set which for some $e = (u, v) \in E$ and for some $i, j \in \{1, \ldots, C\}$ with $i \neq j$ has $e_u^i, e_v^i, e_u^j, e_v^j \in S'$. Then S' is not reachable with TS moves from S_f.*

Proof. Let S' be an independent set that satisfies the conditions of the lemma but is reachable from S_f with the minimum number of token sliding moves. Consider a sequence that transforms S_f to S', and let S'' be the independent set immediately before S' in this sequence. S'' contains exactly three of the vertices $e_u^i, e_v^i, e_u^j, e_v^j$. Without loss of generality say $e_v^j \notin S''$. Therefore, the move that transforms S'' to S' slides a token into e_v^j from one of the neighbors of this vertex. We now observe that $N(e_v^j)$ contains C copies of each neighbor of e_v in G_b, plus the gate vertices corresponding to e in the j-th copy of G_b. However, the C copies of the neighbors of e_v are also neighbors of e_v^i, hence a token cannot slide through these vertices. Furthermore, the gate vertices of e are also neighbors of e_u^j. We therefore have a contradiction. ◀

We now use Lemma 6 to show that for each original edge, the graph G_f contains some non-trivial number of tokens on the selector vertices of that edge.

▶ **Lemma 7.** *Let S_f be the initial independent set constructed in our instance and S' be an independent set which for some $e = (u, v) \in E$ has $|S' \cap (\{e_u^i \mid 1 \leq i \leq C\} \cup \{e_v^i \mid 1 \leq i \leq C\})| < 4$. Then S' is unreachable from S_f.*

Proof. Suppose S' is reachable. Then by Lemma 6, for each edge $e = (u, v) \in E$ we have $|S' \cap (\{e_u^i \mid 1 \leq i \leq C\} \cup \{e_v^i \mid 1 \leq i \leq C\})| \leq C + 1$, because otherwise there would exist (by pigeonhole principle) $e_u^i, e_v^i, e_u^j, e_v^j \in S'$. We now use a simple counting argument. The total number of tokens is mC, while for any edge $f \in E$ we have $\sum_{e \in E \setminus \{f\}} |S' \cap (\{e_u^i \mid 1 \leq i \leq$

$C\}\cup\{e_v^i \mid 1 \le i \le C\})| \le (m-1)(C+1)$. However, $(m-1)(C+1) = mC+m-C-1 = mC-5$, where we use the fact that $C = m+4$. As a result $|S' \cap (\{e_u^i \mid 1 \le i \le C\} \cup \{e_v^i \mid 1 \le i \le C\})| \ge 4$ for any edge $e \in E$, as the independent set S' uses at most one vertex from the clique. ◀

We are now ready to establish the final lemma that gives a mapping from a sliding token reconfiguration in G_f to one in G_b.

▶ **Lemma 8.** *If there exists a reconfiguration from S_f to T_f in G_f under the TS rule then there exists a reconfiguration from S to T in G_b under the TS rule which for each edge $e = (u, v) \in E$ contains at most one of the vertices e_u, e_v in every independent set in the sequence.*

Proof. Take a configuration I of G_f, that is an independent set in the supposed sequence from S_f to T_f. We map this independent set to an independent set I' of G_b as follows: for each edge $e = (u, v) \in E$, we set $e_u \in I'$ if and only if $|I \cap \{e_u^i \mid 1 \le i \le C\}| \ge |I \cap \{e_v^i \mid 1 \le i \le C\}|$. Informally, this means that we take the majority setting from G_f. We note that this always gives an independent set I' that contains exactly one vertex from $\{e_u, e_v\}$ for each $e = (u, v) \in E$.

Our main argument now is to show that if I_1, I_2 are two consecutive independent sets of the solution for G_f, then the sets I_1', I_2' which are obtained in the way described above in G_b are either identical or can be obtained from one another with two sliding moves. If I_1', I_2' are not identical, they may differ in at most two vertices corresponding to an edge $e = (u, v) \in E$, say $\{e_u\} = I_1' \setminus I_2'$ and $\{e_v\} = I_2' \setminus I_1'$. This is not hard to see, since I_2 is obtained from I_1 with one sliding move, and this move can only affect the majority opinion for at most one edge.

Now we would like to argue that it is possible to slide e_u to a gate vertex associated to e and then to e_v in G_b. Consider the transition from I_1 to I_2. This move either slides a token from some e_u^i to the clique, or slides a token from the clique to some e_v^j (because the majority opinion changed from e_u to e_v). Because of Lemma 7, both I_1 and I_2 contain at least four vertices in some copies of e_u, e_v. Hence, since at least half of these vertices are in copies of e_u in I_1, there exists some $e_u^i \in I_1 \cap I_2$. Similarly, there exists some $e_v^j \in I_1 \cap I_2$. Consider now a gate vertex g in the clique of G_b such that g is not associated with e. If g has an edge to $\{e_u, e_v\}$ in G_b, then all copies of g in G_f have an edge to $I_1 \cap I_2$, therefore cannot belong in either set. As a result, the clique vertex that is used in the transition from I_1 to I_2 is a copy of a gate vertex associated with e (either g_e, or one of $g_{e,1}, g_{e,2}$, depending on the color of e). This gate vertex copy therefore has no neighbor in $I_1 \cap I_2$. From this we conclude that the same gate vertex in G_b also has no neighbor in $I_1' \cap I_2'$, as the majority opinion only changed for e. It is therefore legal to slide from e_u to this gate vertex and then to e_v. ◀

▶ **Theorem 9.** *Sliding Token Reconfiguration is PSPACE-complete for split graphs.*

Proof. We begin with an instance of the PSPACE-complete NCL reconfiguration problem, as given in Lemma 3. We construct the instance G_f, S_f, T_f of Sliding Token Reconfiguration on split graphs as described (it's clear that this can be done in polynomial time). If the NCL reconfiguration instance is a YES instance, then by Lemma 4 there exists a sliding token reconfiguration of G_b, and by repeated applications of Lemma 5 to independent sets that do not contain clique vertices in the reconfiguration of G_b there exists a sliding token reconfiguration of G_f. If on the other hand there exists a sliding token reconfiguration on G_f, then by Lemma 8 there exists a reconfiguration that satisfies the condition of Lemma 4 on G_b, hence the original NCL instance is a YES instance. ◀

4 PSPACE-completeness for Chordal Graphs for $c \geq 2$

In this section, we build upon the PSPACE-completeness result from Section 3 to show that c-Colorable Set Reconfiguration is PSPACE-complete, for every $c \geq 2$, when the input graph is restricted to be chordal.

▶ **Theorem 10.** *For every $c \geq 2$, the c-Colorable Set Reconfiguration problem under the TS rule is PSPACE-complete, even when the input graph is restricted to be chordal.*

Proof. We provide a reduction from Independent Set Reconfiguration where the input graph G is restricted to be a split graph, which we proved to be PSPACE-complete in Theorem 9. Let $G = (V, E)$ be an input split graph for Independent Set Reconfiguration. We construct a chordal graph G' as follows, starting from a graph isomorphic to G and two non-empty independents set S, T of the same size. For every edge $uv \in E(G)$, we add $|V(G)|$ sets of $c-1$ new vertices $W^1_{uv}, \ldots, W^{|V(G)|}_{uv}$, such that W^i_{uv} induces a clique for every $1 \leq i \leq |V(G)|$, and every vertex of W^i_{uv} is made adjacent to both u and v, for every $1 \leq i \leq |V(G)|$. In addition, we create a new set $S' = S \cup \bigcup_{uv \in E(G), 1 \leq i \leq |V(G)|} W^i_{uv}$ and a set $T' = T \cup \bigcup_{uv \in E(G), 1 \leq i \leq |V(G)|} W^i_{uv}$. In other words, we append $|V(G)|$ disjoint cliques of size $c-1$ to every edge of G, and add all those newly created vertices to S and to T. The chordality of G' follows from the fact that the new vertices of the sets W^i_{uv} are all simplicial in G', hence G' is chordal if and only if G is chordal as well (and G is split).

We now claim the following: given in independent set T of G, the instance (G, S, T) of Independent Set Reconfiguration is a YES-instance if and only if the instance (G', S', T') of c-Colorable Set Reconfiguration is a YES-instance as well. Observe that, by the construction, S' and T' are c-colorable because the maximum clique in $G'[S']$ contains at most one vertex of S and at most the $c-1$ vertices of a clique W^i_{uv}.

The forward direction of the previous claim follows easily: performing the same moves as those of a reconfiguration sequence from S to T in G', starting from S', yields a reconfiguration sequence where every step preserves c-colorability, and produces the desired set T'.

For the backwards direction, we claim that, for any c-colorable set R' reachable from S', it holds that the vertices of $R' \cap V(G)$ are pairwise non-adjacent. In other words, the tokens placed on original vertices of G form an independent set. Indeed, observe that the number of vertices of G' that do not belong to R' satisfies $|V(G') \setminus R'| = |V(G) \setminus S| < |V(G)|$. This immediately implies that for any set R' and edge $uv \in E(G)$, we have $|R' \cap \bigcup_{1 \leq i \leq |V(G)|} W^i_{uv}| \geq (c-2)|V(G)| + 1$, and therefore $G[R' \cap \bigcup_{1 \leq i \leq |V(G)|} W^i_{uv}]$ contains a clique of size $c-1$ as an induced subgraph, i.e., one of the sets W^i_{uv} is completely contained in R'. This implies that, for every edge uv of G, we have $|R' \cap \{u, v\}| \leq 1$, i.e., the vertices of $R' \cap V(G)$ are pairwise non-adjacent, as desired. ◀

5 XP-time Algorithm on Split Graphs for fixed $c \geq 2$

In this section we present an $n^{O(c)}$ algorithm for c-Colorable Reconfiguration under the TS rule, on split graphs, for $c > 1$. Recall that a split graph $G = (V, E)$ is a graph whose vertex set V is partitioned into a clique K and an independent set I. An input instance consists of a split graph G, and two c-colorable sets $S, T \subseteq V$.

Before proceeding, let us give some high-level ideas as well as some intuition why this problem, which is PSPACE-complete for $c = 1$ (Theorem 9), admits such an algorithm for larger c. Our algorithm consists of two parts: a rigid and a non-rigid reconfiguration part. In the rigid reconfiguration part the algorithm decides if two sets are reachable by using

moves that never slide tokens into or out of I. Because of this restriction and the fact that the sets are c-colorable, the total number of possible configurations is $n^{O(c)}$, so this part can be solved with exhaustive search (this is similar to the algorithm of [15] for TJ/TAR). In the non-rigid part we assume we are given two sets S, T which, in addition to being c-colorable, have $|S \cap K|, |T \cap K| \leq c - 1$. The main insight is now that *any* two such sets are reachable via TS moves (Lemma 11 below). Informally, the algorithm guesses a partition of the optimal reconfiguration into a rigid prefix, a rigid suffix, and a non-rigid middle, and uses the two parts to calculate each independently.

The intuitive reason that our algorithm cannot work for $c = 1$ is the non-rigid part. The crucial Lemma 11 on which this part is based fails for $c = 1$: for instance, if G is a star with three leaves and S, T are two distinct sets each containing two leaves, then S, T satisfy all the conditions for $c = 1$, but are not reachable from each other with TS moves. Such counterexamples do not, however, exist for higher c, because for sets that satisfy the conditions of Lemma 11 we know we can always freely move tokens around inside the clique (and without loss of generality, such tokens exist). Note also, that this difficulty is specific to the TS rule: the algorithm of [15] implicitly uses the fact that any two sets with $c - 1$ tokens in the clique are always reachable, as this is an almost trivial fact if one is allowed to use TJ moves. Thus, Lemma 11 is the main new ingredient that makes our algorithm work.

Let us now proceed with a detailed description of the algorithm. First, let us fix some notation. For a vertex set $R \subseteq V$, we write the subsets $R \cap K$ and $R \cap I$ as R_K and R_I respectively.

Throughout this section, we assume that input graph $G = (K \cup I, E)$ is connected (and thus each vertex in I has a neighbor in K); otherwise we can consider instances induced by each component separately.

▶ **Lemma 11.** *Let G be a split graph, $c \geq 2$, and $S, T \subseteq V$ be two c-colorable sets such that $|S_K|, |T_K| \leq c - 1$. Then T is c-reachable from S. Furthermore, a reconfiguration sequence from S to T can be produced in polynomial time.*

Proof. We first observe that if $S_I = T_I$, then there is an easy optimal c-transformation. By making one TS move from $u \in S_K \setminus T_K$ to $v \in T_K \setminus S_K$, one can c-transform S to T with $|S \setminus T|$ sliding moves (thus yielding an optimal reconfiguration sequence). It is clear that all the sets resulting from these TS moves are c-colorable because each of them has at most $c - 1$ vertices in K.

Therefore, it suffices to show that there is always a c-transformation of T which decrease $|S_I \setminus T_I|$ as long as $S \neq T$. Note that we can assume that there exists $v \in S_I \setminus T_I$ (otherwise we exchange the roles of S and T). In the case when $T_K = \emptyset$, one can transform T to T' with TS moves from a vertex of $T_I \setminus S_I$ to v. Trivially this is a c-transformation, and it holds that $|T'_K| = \emptyset$. (Note that this argument would not be valid if $c = 1$). If $T_K \neq \emptyset$, then one can make at most two TS moves from a vertex of T_K to v. Because T has at most $c - 1$ vertices and these TS moves maintain at most $c - 1$ vertices in K, c-colorability of T is preserved. Moreover, the new set has at most $c - 1$ vertices in K while its intersection with S in I is strictly larger. This completes the proof of the first statement. The proof is constructive and easily translates to a polynomial-time algorithm. ◀

Let us now introduce a notion that will be useful in our algorithm. For two c-colorable sets S, T with $S_I = T_I$ we say that S has a *rigid* c-transformation to T if there exists a valid c-transformation from S to T with TS moves which also has the property that every c-colorable set R of the transformation has $R_I = S_I$.

▶ **Lemma 12.** *Given a split graph $G = (V, E)$, with $V = K \cup I$, and two c-colorable sets $S, T \subseteq V$ with $S_I = T_I$, there is an algorithm that decides if there exists a rigid c-transformation of S to T in time $n^{O(c)}$.*

Proof. The main observation is that since all intermediate sets must have $R_I = S_I$, we are only allowed to slide tokens inside K. However, S_K contains at most c vertices (as it is c-colorable), therefore, there are at most n^c potentially reachable sets: one for each collection of $|S_K|$ vertices of the clique.

We now construct a secondary graph with a node for each subset of V that contains $|S_K|$ vertices of K and the vertices of S_I, and connect two such nodes if their corresponding sets are reachable with a single TS move in G. In this graph we check if there is a path from the node that represents S to the one that represents T and if yes output the sets corresponding to the nodes of the path as our rigid reconfiguration sequence. ◀

▶ **Theorem 13.** *There is an algorithm that decides c-Colorable Reconfiguration on split graphs under the TS rule in time $n^{O(c)}$, for $c \geq 2$.*

Proof. We distinguish the following cases: (i) $|S_K|, |T_K| \leq c-1$, (ii) $|S_K| = c$ and $|T_K| = c-1$, (iii) $|S_K| = |T_K| = c$. This covers all cases since S, T are c-colorable and we can assume without loss of generality that $|S_K| \geq |T_K|$.

For case (i) we invoke Lemma 11. The answer is always Yes, and the algorithm of the lemma produces a feasible reconfiguration sequence.

For case (ii), suppose there exists a reconfiguration sequence from S to T, call it $T_0 = S, T_1, \ldots, T_\ell = T$. Let i be the smallest index such that $|T_i \cap K| \leq c - 1$. Clearly such an index exists, since $|T_K| \leq c - 1$. We now guess the configuration T_{i-1} and the configuration T_i (that is, we branch into all possibilities). Observe that there are at most n^c choices for T_{i-1} as we have $T_{i-1} \cap I = S_I$ and $|T_{i-1} \cap K| = c$. Furthermore, once we have selected a T_{i-1}, there are $n^{O(1)}$ possibilities for T_i, as T_i is reachable from T_{i-1} with one TS move.

We observe that if we guessed correctly, then there exists a rigid c-transformation from S to T_{i-1} (by the minimality of i and the fact that $|S_K| = c$); we use the algorithm of Lemma 12 to check this. Furthermore, the configuration T_i is always transformable to T by Lemma 11. Therefore, if the algorithm of Lemma 12 returns a solution, then we have a c-transformation from S to T. Conversely, if a c-transformation from S to T exists, since we tried all possibilities for T_{i-1}, one of the branches will find it.

Finally, for case (iii), if $S_I = T_I$ we first use Lemma 12 to check if there is a rigid c-transformation from S to T. If one is found, we are done. If not, or if $S_I \neq T_I$ we observe that, similarly to case (ii), in any feasible transformation $T_0 = S, T_1, \ldots, T_\ell = T$, there exists an i such that $|T_i \cap K| \leq c - 1$ (otherwise the transformation would be rigid). Pick the minimum such i. We now guess the configurations T_{i-1}, T_i (as before, there are $n^{c+O(1)}$ possibilities) and use Lemma 12 to verify that T_{i-1} is reachable from S. If T_{i-1} is reachable from S, we need to verify that T is reachable from T_i. However, we observe that this reduces to case (ii), because $|T_i \cap K| \leq c - 1$, so we proceed as above. If the algorithm returns a valid sequence we accept, while we know that if a valid sequence exists, then there exists a correct guess for T_{i-1}, T_i that we consider. ◀

6 W-hardness for Split Graphs

In this section we show that c-Colorable Reconfiguration on split graphs is W[2]-hard parameterized by c and the length ℓ of the reconfiguration sequence under all three reconfiguration rules (TAR, TJ, and TS). In this sense, this section complements Section 5 by

showing that the $n^{O(c)}$ algorithm that we presented for c-Colorable Reconfiguration on split graphs cannot be significantly improved under standard assumptions.

We will rely on known results on the hardness of Dominating Set Reconfiguration. We recall that in this problem we are given a graph $G = (V, E)$, two dominating sets $S, T \subseteq V$ of size at most k and are asked if we can transform S into T by a series of TAR operations while keeping the size of the current set at most k at all times. More formally, we are asked if there exists a sequence $T_0 = S, T_1, \ldots, T_\ell = T$ such that for each $i \in \{0, \ldots, \ell - 1\}$, $|T_i| \leq k$, T_i is a dominating set of G, and $|(T_i \setminus T_{i+1}) \cup (T_{i+1} \setminus T_i)| = 1$.

▶ **Theorem 14** ([18]). Dominating Set Reconfiguration *is W[2]-hard parameterized by the maximum size of the allowed dominating sets k and the length ℓ of the reconfiguration sequence under the TAR rule.*

Before proceeding, let us make two remarks on Theorem 14: first, because the reduction of [18] is linear in the parameters, it is not hard to see that it also implies a tight ETH-based lower bound based on known results for Dominating Set; second, using an argument similar to that of Theorem 1 of [16], the same hardness can be obtained for the TJ rule.

▶ **Corollary 15.** Dominating Set Reconfiguration *is W[2]-hard parameterized by the maximum size of the allowed dominating sets k and the length ℓ of the reconfiguration sequence under the TAR, or TJ rule. Furthermore, the problem does not admit an algorithm running in $n^{o(c+\ell)}$ under the ETH for any of the two rules.*

Proof. To obtain hardness under the TJ rule we use an argument similar to that of Theorem 1 of [16]. Suppose we are given an instance of k-Dominating Set Reconfiguration $G = (V, E)$ and $S, T \subseteq V$ where k is the maximum size of any dominating set allowed and we use the TAR rule, that is, an instance produced by the reduction establishing Theorem 14. We recall that in the instances produced for this reduction we have $k = \Theta(\ell)$ and that S can be transformed into T with ℓ TAR moves if and only if S can be transformed into T with some number of TAR moves (in other words, if ℓ moves are not sufficient, then S and T are in fact unreachable). This observation will be useful because it means that in the reduction that follows we do not have to preserve ℓ exactly but only guarantee that it increases by at most a constant factor.

We can assume without loss of generality that $|S| = |T| = k - 1$: if $|S| < k - 1$ we can add to S arbitrary vertices to make its size $k - 1$, while if $|S| = k$ then S cannot be a minimal dominating set (otherwise it would be impossible to transform it to any other set and we would have an obvious NO instance) so there is a vertex that we can remove from S without affecting the answer. In both cases we appropriately increase ℓ by the number of modifications we made to S, T to preserve reachability. We want to show that the instance is now equivalent under the TJ rule. In particular, there exists a TAR reconfiguration with 2ℓ moves if there exists a TJ reconfiguration with ℓ moves.

First, if there exists a TJ reconfiguration from S to T then there exists a TAR reconfiguration from S to T: for each move that exchanges $u \in S$ with $v \notin S$ we first add v to S and then remove u.

For the converse direction, suppose that there is a TAR reconfiguration of S to T. If moves alternate in this reconfiguration, that is, if all intermediate sets have size between $k-2$ and k, then it is not hard to see how to perform the same reconfiguration with TJ moves. Suppose then that the reconfiguration performs two consecutive vertex removal moves, so we have the dominating sets T_i, T_{i+1}, T_{i+2} appearing consecutively in the reconfiguration sequence, with $|T_i| = |T_{i+1}| + 1 = |T_{i+2}| + 2$. Let j be the smallest index with $j > i + 2$ such

that $|T_j| > |T_{j-1}|$ (i.e. j signifies the first time we added a vertex after the i-th move). Let $T_i \setminus T_{i+1} = \{u\}$ and $T_j \setminus T_{j-1} = \{v\}$. Then, if $u = v$ we can add u to all sets $T_{i+1}, \ldots, T_{j-1}$ and obtain a shorter reconfiguration sequence (since now $T_i = T_{i+1}$ and $T_j = T_{j-1}$). Similarly, if $u \neq v$ and $v \in T_{i+1}$ we add v to all sets $T_{i+2}, \ldots, T_{j-1}$ to which it doesn't appear and we have a shorter reconfiguration sequence. Finally, if $u \neq v$ and $v \notin T_{i+1}$, we insert after T_{i+1} the set $T_{i+1} \cup \{v\}$ and then add v to all sets $T_{i+2}, \ldots, T_{j-1}$. We now have $T_{j-1} = T_j$, so we have a valid TAR reconfiguration of the same length but with one less pair of consecutive vertex removals. Repeating this argument produces a TAR reconfiguration which can be performed with TJ moves.

For the ETH-based lower bound it suffices to recall that, under the ETH t-Dominating Set does not admit an $n^{o(t)}$ algorithm [4], and that the reduction establishing Theorem 14 in [18] is a reduction from t-Dominating Set that sets $k, \ell = O(t)$. ◀

▶ **Theorem 16.** *The c-Colorable Reconfiguration problem is W[2]-hard parameterized by c and the reconfiguration length ℓ when restricted to split graphs under any of the three reconfiguration rules (TAR, TJ, TS). Furthermore, under the ETH, the same problem does not admit an $n^{o(c+\ell)}$ algorithm.*

Proof. We use a reduction from Dominating Set Reconfiguration similar to the one used in [15] to prove that our problem is PSPACE-complete if c is part of the input. Let $G = (V, E)$ be an input graph for Dominating Set Reconfiguration. We construct a split graph G' as follows: we take two copies of V, call them V_1, V_2; we turn V_1 into a clique; for each $u \in V_1$ and $v \in V_2$ we add the edge (u, v) if and only if $u \notin N[v]$ in G. In other words, we connect each vertex from V_1 with all the vertices of V_2 which it *does not* dominate in G.

We assume now that we have started with k-Dominating Set Reconfiguration instance under the TJ rule, which is W[2]-hard according to Corollary 15 parameterized by $k + \ell$. We will first show hardness of c-Colorable Reconfiguration for TJ and TS parameterized by $c + \ell$.

We construct a one-to-one correspondence between size k dominating sets of G and k-colorable sets of vertices of G' of size $n + k$, where $n = |V|$: for each such set $S \subseteq V$ we define its image $\phi(S)$ in G' as $\{u \in V_1 \mid u \in S\} \cup V_2$. In other words, we select all the vertices of S from V_1 and all of V_2. It is not hard to see that $\phi(S)$ is indeed k-colorable: if not, there exists a clique of size $k + 1$ in $G'[S']$ (since split graphs are perfect), which must consist of the k vertices of S from V_1, plus a vertex v from V_2. But v must be dominated by a vertex $u \in S$ in G, which means that v and the copy of u in V_1 are not connected.

Let us also observe that for every k-colorable set S' of size $n + k$ in G' we have that $S' = \phi(S)$ for some dominating set S of size k in G. To see this, observe that S' must contain exactly k vertices of V_1 (since it is k-colorable, V_1 is a clique, and $|V_2| = n$). These vertices must be a dominating set of G as otherwise there would exist a vertex v that is not in any of their closed neighborhoods, and the copy of v in V_2 together with $S' \cap V_1$ would form a clique of size $k + 1$, contradicting the k-colorability of S'.

Given the above correspondence it is not hard to complete the reduction: if we are given two dominating sets $S, T \subseteq V$ with the initial instance we set $\phi(S), \phi(T)$ as the two k-colorable graphs of the new instance. We observe that any valid TJ move that transforms a dominating set T_i to a dominating set T_{i+1} in G, corresponds to a TJ move that transforms $\phi(T_i)$ to $\phi(T_{i+1})$ in G'. Crucially, such a move is also a TS move, as the symmetric difference of T_i and T_{i+1} is contained in the clique. Hence, there is also a one-to-one correspondence

between TJ k-dominating set reconfigurations in G and TS k-colorable subgraph (of size $n + k$) reconfiguration in G'. We therefore set the length of the desired reconfiguration sequence in G' to ℓ.

Finally, to obtain hardness of the new instance under the TAR rule we set the lower bound on the size of any intermediate set to $n + k - 1$. Since $|\phi(S)| = |\phi(T)| = n + k$ this means that any TJ c-colorable reconfiguration can also be performed with at most 2ℓ TAR moves. For the converse direction we observe that in any TAR reconfiguration we never have a set of size $n + k + 1$ or more, since such a set would necessarily induce a graph that needs $k + 1$ colors. Hence, such a reconfiguration must consist of alternating vertex removal and addition moves, which can be performed with ℓ TJ moves.

The ETH-based lower bounds follow from Corollary 15 and the fact that the reduction we performed is at most linear in all parameters. ◀

References

1 Marthe Bonamy and Nicolas Bousquet. Token Sliding on Chordal Graphs. In *WG 2017*, volume 10520 of *Lecture Notes in Computer Science*, pages 127–139, 2017. doi:10.1007/978-3-319-68705-6_10.

2 Paul S. Bonsma, Marcin Kaminski, and Marcin Wrochna. Reconfiguring Independent Sets in Claw-Free Graphs. In *SWAT*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2014.

3 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token Jumping in Minor-Closed Classes. In *FCT*, volume 10472 of *Lecture Notes in Computer Science*, pages 136–149. Springer, 2017.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

5 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. *Theor. Comput. Sci.*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.

6 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding Token on Bipartite Permutation Graphs. In *ISAAC*, volume 9472 of *Lecture Notes in Computer Science*, pages 237–247. Springer, 2015.

7 Robert A. Hearn. *Games, puzzles, and computation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006. URL: http://hdl.handle.net/1721.1/37913.

8 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

9 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009.

10 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *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.

11 Duc A. Hoang and Ryuhei Uehara. Sliding Tokens on a Cactus. In *ISAAC*, volume 64 of *LIPIcs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

12 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. *Theor. Comput. Sci.*, 412(12–14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

13 Takehiro Ito, Marcin Kaminski, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the Parameterized Complexity for Token Jumping on Graphs. In *TAMC*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer, 2014.

14 Takehiro Ito, Marcin Jakub Kaminski, and Hirotaka Ono. Fixed-Parameter Tractability of Token Jumping on Planar Graphs. In *ISAAC*, volume 8889 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014.

15 Takehiro Ito and Yota Otachi. Reconfiguration of Colorable Sets in Classes of Perfect Graphs. In *SWAT*, volume 101 of *LIPIcs*, pages 27:1–27:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

16 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

17 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. In *SODA 2018*, pages 185–195, 2018. doi:10.1137/1.9781611975031.13.

18 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the Parameterized Complexity of Reconfiguration Problems. *Algorithmica*, 78(1):274–297, 2017.

19 Naomi Nishimura. Introduction to Reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

20 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

21 Douglas B. West. *Introduction to graph theory*. Prentice Hall, Upper Saddle River, 2nd edition, 2001.

Building Strategies *into* QBF Proofs

Olaf Beyersdorff
Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany
olaf.beyersdorff@uni-jena.de

Joshua Blinkhorn
Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany
joshua.blinkhorn@uni-jena.de

Meena Mahajan
The Institute of Mathematical Sciences, HBNI, Chennai, India
meena@imsc.res.in

Abstract

Strategy extraction is of paramount importance for quantified Boolean formulas (QBF), both in solving and proof complexity. It extracts (counter)models for a QBF from a run of the solver resp. the proof of the QBF, thereby allowing to certify the solver's answer resp. establish soundness of the system. So far in the QBF literature, strategy extraction has been algorithmically performed *from* proofs. Here we devise the first QBF system where (partial) strategies are built *into* the proof and are piecewise constructed by simple operations along with the derivation.

This has several advantages: (1) lines of our calculus have a clear semantic meaning as they are accompanied by semantic objects; (2) partial strategies are represented succinctly (in contrast to some previous approaches); (3) our calculus has strategy extraction by design; and (4) the partial strategies allow new sound inference steps which are disallowed in previous central QBF calculi such as Q-Resolution and long-distance Q-Resolution.

The last item (4) allows us to show an exponential separation between our new system and the previously studied reductionless long-distance resolution calculus, introduced to model QCDCL solving.

Our approach also naturally lifts to dependency QBFs (DQBF), where it yields the first sound and complete CDCL-type calculus for DQBF, thus opening future avenues into DQBF CDCL solving.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases QBF, DQBF, resolution, proof complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.14

Funding Supported by the EU Marie Curie IRSES grant CORCON, and grant no. 60842 from the John Templeton Foundation.

1 Introduction

Proof complexity investigates the resources for proving logical theorems, focussing foremost on the minimal size of proofs needed in a particular calculus. Since its inception the field has enjoyed strong connections to computational complexity (cf. [14, 17]) and to first-order logic [16, 25]).

During the past decade, proof complexity has emerged as a key tool to model and analyse advances in the algorithmic handling of hard problems such as SAT and beyond. While traditionally perceived as a computationally hard problem, SAT solvers have been enormously successful in tackling huge industrial instances [28, 38] and hard combinatorial problems [21]. As each run of a solver on an unsatisfiable formula can be understood as a proof of unsatisfiability, each solver implicitly defines a proof system. This connection turns

proof complexity into the main theoretical approach towards understanding the power and limitations of solving, with bounds on proof size directly corresponding to bounds on solver running time [14, 29].

The algorithmic success story of solving has not stopped at SAT, but is currently extending to even more computationally complex problems such as *quantified Boolean formulas* (QBF), which is PSPACE complete, and *dependency QBFs* (DQBF), which is even NEXP complete. While quantification does not increase expressivity, (D)QBFs can encode many problems far more succinctly, including application domains such as automated planning [15, 18], verification [5, 27], synthesis [20, 26] and ontologies [24].

The past 15 years have seen *huge advances in QBF solving*, which currently reaches the point of industrial applicability. While some of the main innovations in SAT solving, including the development of conflict-driven clause learning (CDCL), revolutionised SAT in the late 1990s [36], this development in QBF is happening *now*. Consequently, QBF proof complexity has received considerable attention in recent years. Compared with QBF, solving in DQBF is at its very beginnings, both in implementations (2018 was the first year that saw a DQBF track in the QBF competition [1]) as well as in its accompanying theory [35].

Strategy extraction is one of the distinctive features of QBF and DQBF, manifest in both solving and proof complexity. For solving it guarantees that together with the true/false answer the (D)QBF solver can produce a model (resp. countermodel) of the (D)QBF, thus *certifying* the correctness of the answer. On the proof complexity side, this implies that proof calculi modelling QBF solving should allow strategy extraction in the sense that from a refutation of false QBF, a countermodel of the QBF can be efficiently constructed. This feature – without analogue in the propositional domain – enables strong lower bound techniques in QBF proof complexity [8, 9, 11], exploiting the fact that formulas requiring hard strategies cannot have short proofs in calculi with efficient strategy extraction.

As in SAT versus propositional proof complexity, one of the prime challenges in QBF and DQBF is to create compelling proof-theoretic models that capture central features of (D)QBF solving and at the same time remain amenable to a proof-theoretic analysis. While there exist several orthogonal approaches in QBF solving with quite different associated proof calculi, we will focus here on the paradigm of conflict-driven clause learning in QBF (QCDCL) [39]. Proof-theoretically its most basic model is Q-Resolution [22], which as in propositional resolution operates on clauses (of prenex QBFs).

Q-Resolution (Q-Res) uses the resolution rule of propositional resolution and augments this with a universal reduction rule that allows to eliminate universal variables from clauses. Combining these two rules requires some technical care: without any side-conditions the two rules result in an unsound system. Typically this is circumvented by prohibiting the derivation of universal tautologies. It was noted early on that in solving this is needlessly prohibitive [39], and universal tautologies can be permitted under certain side-conditions. Later formalised as the proof system *long-distance Q-Resolution* (LD-Q-Res) [3], it was even shown that LD-Q-Res exponentially shortens proofs in comparison to Q-Res [19], thus demonstrating the appeal of the approach for solving. In fact, when enabling long-distance steps in QBF solving, universal reduction is not strictly needed and this reductionless approach was adopted in the QBF solver GhostQ [23]. To model this solving paradigm, Bjørner, Janota, and Klieber [13] introduced the calculus of *reductionless* LD-Q-Res.

The interplay between long-distance resolution and universal reduction steps becomes even more intriguing in DQBF. In [2] it was shown that lifting Q-Res (using the rules of resolution and universal reduction) to DQBF results in an incomplete proof system, whereas lifting LD-Q-Res (using long-distance resolution steps together with universal reduction) becomes unsound [12].

Naturally, the intriguing question of why and how deriving "universal tautologies" in long-distance steps might help solving has attracted attention among theoreticians and practitioners alike. Instead of a universal tautology $u \vee \bar{u}$, most formalisations of long-distance resolution actually use the concept of a "merged" literal u^*. While it is clear (and implicit in the literature) that merged literals u^* correspond to partial strategies for u rather than universal tautologies, a formal semantic account of long-distance steps (and stronger calculi using merging [10]) was only recently given by Suda and Gleiss [37], where partial strategies are constructed for each individual proof inference. However, as already noted in [37], the models considered in [37] fail to have efficient strategy extraction in the sense that the constructed (partial) strategies may need exponential-size representations.

Our contributions

A. The new calculus of Merge Resolution. Starting from the reductionless LD-Q-Res system of [13] and its role of modelling QCDCL solving, we develop a new calculus that we call Merge Resolution (M-Res). Like reductionless LD-Q-Res, the system M-Res only uses a resolution rule and does not permit universal reduction steps. Reductionless LD-Q-Res and M-Res are therefore both refutational calculi that finish as soon as they derive a purely universal clause.

As the prime novel feature of M-Res we build partial strategies *into* proofs. We achieve this by computing explicit representations of strategies in a variant of binary decision diagrams (called *merge maps* here), which are updated and refined at each proof step by simple operations. These merge maps are part of the proof. As a consequence, M-Res has efficient strategy extraction by design.

This is in contrast to all previous existing QBF calculi in the literature, where strategies are algorithmically constructed *from* proofs. In particular, this also applies to the approaches taken in [19, 37] for LD-Q-Res and in [13] for reductionless LD-Q-Res. But also the choice of our representation as merge maps matters: as [13, 37] both represent (partial) strategies as trees, the constructed strategies may grow exponentially in the proof size, thus losing the desirable property of efficient strategy extraction. In contrast, in our model merge maps are always linear in the size of the clause derivations.

B. Exponential separation of M-Res from reductionless LD-Q-Res. Including merge maps explicitly into proofs also has another far-reaching advantage: it allows resolution steps not only forbidden in Q-Res, but even disallowed in LD-Q-Res. In a nutshell, LD-Q-Res allows resolution steps only when universal variables quantified left of the pivot have *constant and equal* strategies in both parent clauses. In M-Res we have explicit representations of strategies and thus can allow resolution steps as long as the strategies in both parent clauses are *isomorphic* to each other, a property that we can check efficiently for merge maps.

This manifests in shorter proofs. We show this by explicitly giving an example of a family of QBFs that admit linear-size proofs in M-Res (Theorem 21), but require exponential size in reductionless LD-Q-Res (Theorem 20). The separating formulas are a variant of the equality formulas introduced in [8]. While the original formulas from [8] are hard for Q-Res, but easy in LD-Q-Res, we here consider a "squared" version, for which we naturally use resolution steps for clauses with associated non-constant winning strategies, allowed in M-Res, but forbidden in LD-Q-Res.

This shows that M-Res is exponentially stronger than reductionless LD-Q-Res, thus also pointing towards potential improvements in QCDCL solving. While the simulation of reductionless LD-Q-Res by M-Res is almost immediate and also the upper bound in M-Res is

comparatively straightforward, the lower bound is a technically involved argument specifically tailored towards the squared equality formulas.

C. A sound and complete CDCL calculus for DQBF. As our final contribution we show that the new QBF system of M-Res naturally lifts to a sound and complete calculus for DQBF. As shown in [2], the lifting of Q-Res to DQBF is incomplete, whereas the combination of universal reduction and long-distance steps presents soundness issues, both in DQBF [12] as well as in the related framework of dependency schemes [6, 7].

Here we show that M-Res overcomes both these soundness and completeness issues and therefore has exactly the right strength for a natural DQBF resolution calculus. In fact, it is the first DQBF CDCL-type system in the literature[1] and as such paves the way towards CDCL solving in DQBF. Again, by design our DQBF system has efficient strategy extraction.

2 Preliminaries

Propositional logic. Let $\mathcal{Z}$ be a countable set of Boolean variables. A *literal* is a Boolean variable $z \in \mathcal{Z}$ or its negation $\bar{z}$, a *clause* is a set of literals, and a *CNF* is a set of clauses. For a literal l, we define $\mathrm{var}(l) := z$ if $l = z$ or $l = \bar{z}$; for a clause C, we define $\mathrm{vars}(C) := \{\mathrm{var}(l) : l \in C\}$; for a CNF ϕ we define $\mathrm{vars}(\phi) := \cup_{C\in\phi}\mathrm{vars}(C)$. An assignment to a set $Z \subseteq \mathcal{Z}$ of Boolean variables is a function $\rho : Z \to \{0, 1\}$, conventionally represented as a set of literals in which z (resp. $\bar{z}$) represents the assignment $z \mapsto 1$ (resp. $z \mapsto 0$). The set of all assignments to Z is denoted $\langle Z\rangle$. Given a subset $Z' \subseteq Z$, $\rho{\restriction_{Z'}}$ is the restriction of ρ to Z'. The CNF $\phi[\rho]$ is obtained from ϕ by removing any clause containing a literal in ρ, and removing the negated literals $\{\bar{l} : l \in \rho\}$ from the remaining clauses. We say that ρ *falsifies* ϕ if $\phi[\rho]$ contains the empty clause, and that ϕ is *unsatisfiable* if it is falsified by each $\rho \in \langle Z\rangle$.

Given two clauses R_1 and R_2 and a literal l such that $l \in R_1$ and $\bar{l} \in R_2$, we define the resolvent $\mathrm{res}(R_1, R_2, l) := (R_1 \setminus \{l\}) \cup (R_2 \setminus \{\bar{l}\})$. (Note that $\mathrm{res}(R_1, R_2, l) = \mathrm{res}(R_2, R_1, \bar{l})$.) A *resolution refutation* of a CNF ϕ is a sequence $C_1, \ldots, C_k$ of clauses in which C_k is the empty clause and, for each $i \in [k]$, either (a) $C_i \in \phi$ or (b) $C_i = \mathrm{res}(C_a, C_b, z)$ for some $a, b < i$ and $z \in \mathrm{vars}(\phi)$.

Quantified Boolean formulas. A *quantified Boolean formula* (QBF) in *prenex conjunctive normal form* (PCNF) is denoted $\Phi := \mathcal{Q} \cdot \phi$, where (a) $\mathcal{Q} := \mathcal{Q}_1 Z_1 \cdots \mathcal{Q}_n Z_n$ is the *quantifier prefix*, in which the $Z_i \subset \mathcal{Z}$ are pairwise disjoint finite sets of Boolean variables, $\mathcal{Q}_i \in \{\exists, \forall\}$ for each $i \in [n]$, and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$ for each $i \in [n-1]$, and (b) the *matrix* ϕ is a CNF over $\mathrm{vars}(\Phi) := \bigcup_{i=1}^{n} Z_i$.

The existential (resp. universal) variables of Φ, typically denoted X (resp. U), is the set obtained as the union of the Z_i for which $\mathcal{Q}_i = \exists$ (resp. $\mathcal{Q}_i = \forall$). The prefix $\mathcal{Q}$ defines a binary relation $<_{\mathcal{Q}}$ on $\mathrm{vars}(\Phi)$, such that $z <_{\mathcal{Q}} z'$ holds iff $z \in Z_i$, $z' \in Z_j$, and $i < j$, in which case we say that z' *is right of* z and z *is left of* z'. For each $u \in U$, we define $L_{\mathcal{Q}}(u) := \{x \in X : x <_{\mathcal{Q}} u\}$, i.e. the existential variables left of u.

A *strategy* h for a QBF Φ is a set $\{h_u : u \in U\}$ of functions $h_u : \langle L_{\mathcal{Q}}(u)\rangle \to \{u, \bar{u}\}$. Additionally h is *winning* if, for each $\alpha \in \langle X\rangle$, the restriction of ϕ by $\alpha \cup \{h_u(\alpha{\restriction_{L_{\mathcal{Q}}(u)}}) : u \in U\}$ contains the empty clause. We use the terms "winning strategy" and "countermodel" interchangeably. A QBF is called *false* if it has a countermodel, and *true* if it does not.

[1] Previous DQBF resolution systems either use expansion [12] or extension variables [33].

QBF proof systems. We deal with line-based refutational QBF systems that typically employ axioms and inference rules to prove the falsity of QBFs. We say that P is *complete* if there exists a P refutation of every false QBF, *sound* if there exists no P refutation of any true QBF. We call P a *proof system* if it is sound, complete, and polynomial-time checkable. Given two QBF proof systems P_1 and P_2, P_1 *p-simulates* P_2 if there exists a polynomial-time procedure that takes a P_2-refutation and outputs a P_1-refutation of the same QBF [17].

3 Reductionless long-distance Q-Resolution

In this section we recall the definition of reductionless LD-Q-Res, prove that it is refutationally complete, and demonstrate that it does not have polynomial-time strategy extraction in either of the computational models of [13, 37]. The system appeared first in [13, Fig. 1], where it was referred to as Q^w-resolution.

▶ **Definition 1** (reductionless LD-Q-Res [13]). *A* reductionless LD-Q-Res derivation *from a QBF $\Phi := \mathcal{Q} \cdot \phi$ is a sequence $\pi := C_1, \ldots, C_k$ of clauses in which at least one of* (a) *or* (b) *holds for each $i \in [k]$:*

(a) Axiom. *C_i is a clause from the matrix ϕ;*

(b) Long-distance resolution. *There exist integers $a, b < i$ and an existential pivot $x \in X$ such that $C_i = \mathrm{res}(C_a, C_b, x)$ and, for each $u \in \mathrm{vars}_\forall(C_a) \cap \mathrm{vars}_\forall(C_b)$, if $u <_\mathcal{Q} x$, then $\{u, \bar{u}\} \not\subseteq C_i$.*

The final clause C_k is the conclusion *of π, and π is a* refutation *of Φ iff C_k contains no existential variables.*

A pair of complementary universal literals $\{u, \bar{u}\}$ appearing in a clause is referred to singly as a *merged literal*. It is clear from a wealth of literature[2] that merged literals are "placeholders" for partial strategies, the exact representation left implicit in the structure of the derivation.

We illustrate the rules of the calculus by showing that the equality formulas [8] have linear-size refutations.

▶ **Definition 2** (equality formulas [8]). *The* equality family *is the QBF family whose n^{th} instance has prefix $\exists\{x_1, \ldots, x_n\}\forall\{u_1, \ldots, u_n\}\exists\{t_1, \ldots, t_n\}$ and matrix consisting of the clauses $\{x_i, u_i, t_i\}, \{\bar{x}_i, \bar{u}_i, t_i\}$ for $i \in [n]$, and $\{\bar{t}_1, \ldots, \bar{t}_n\}$.*

▶ **Example 3.** We construct linear-size reductionless LD-Q-Res refutations in two stages. First, resolve each pair $\{x_i, u_i, t_i\}$, $\{\bar{x}_i, \bar{u}_i, t_i\}$ of clauses over pivot x_i to obtain $C_i := \{u_i, \bar{u}_i, t_i\}$. Note that it is allowed to introduce the merged literal $\{u_i, \bar{u}_i\}$ since variable u_i is right of the pivot x_i. Second, resolve the C_i successively against the long clause $\{\bar{t}_1, \ldots, \bar{t}_n\}$ over pivot t_i, to obtain a full set of merged literals $C := \{u_i, \bar{u}_i : i \in [n]\}$. Here, even though u_i is left of the pivot t_i, the appearance of the merged literal $\{u_i, \bar{u}_i\}$ in the resolvent is allowed, since variable u_i is absent from one of the antecedents. The derivation is a refutation since the conclusion C contains no existential literals.

Given a false QBF Φ with a countermodel h, we construct a canonical reductionless LD-Q-Res refutation based on the "full binary tree" representation of a countermodel [34].

[2] The notion is evident to a greater or lesser degree in all of the papers [4, 7, 19, 30, 32, 37].

For each $\alpha \in \langle X \rangle$, there exists some C_α in the matrix falsified by $\alpha \cup h(\alpha)$. The set of all such C_α may be successively resolved over existential pivots in reverse prefix order, finally producing a clause containing no existentials. Merged literals never block resolution steps in this construction, as they only ever appear to the right of the pivot variable.

▶ **Lemma 4.** *Every false QBF has a reductionless* LD-Q-Res *refutation.*

Soundness and polynomial-time checkability of reductionless LD-Q-Res are immediate, as the system uses a subset of the rules of the classical long-distance Q-resolution proof system [3].

The computational model of Bjørner et al. [13]. In tandem with reductionless LD-Q-Res, the authors of [13] introduced a computational model based on tree-like branching programs. The model is used to explicitly construct the partial strategies represented implicitly by merged literals. It can be demonstrated that tree-like branching programs constructed in this way cannot represent strategies efficiently; that is, the system does not have polynomial-time strategy extraction in the associated model.

The computational model of Suda and Gleiss [37]. The authors of [37] proposed a model of partial strategies based on so-called *policies*. They noted that the equality formulas have linear-size refutations in the strong QBF system IRM-calc [10], whereas policies witnessing their falsity must be exponentially large, therefore IRM-calc does not admit polynomial-time strategy in policies. The same is true for reductionless LD-Q-Res, since Example 3 shows that the equality formulas also have linear-size refutations there.

That neither model is suitable for efficient strategy extraction shows that using either *inside* the derivation would result in an artificial, exponential size blow-up. The root of the issue is tree-like models versus DAG-like proofs. The DAG-like computational model that we introduce in the following section is tightly knitted to the refutation, yielding linear-time strategy extraction for free.

4 Merge Resolution

In this section we introduce Merge Resolution (M-Res, Subsection 4.2), and prove that it is sound and complete for QBF (Subsection 4.3). The salient feature of M-Res is the built-in partial strategies, represented as *merge maps*. Given the problems with the computational models of [13, 37], the principal technical challenge is to find a suitable way to define and combine partial strategies devoid of an artifical proof-size inflation.

4.1 Merge maps

Our computational model. A merge map is a branching program that queries a set of existential variables and outputs an assignment to some universal variable, i.e. a literal in $\{u, \bar{u}, *\}$, where $*$ stands for "no assignment". As we intend to tie the DAG structure of the merge maps to the DAG structure of the proof, we will label query nodes with natural numbers based on the proof line indexing (we elaborate on this later). Hence, from a technical standpoint it makes sense to define a merge map as a function from the index set of its nodes.

▶ **Definition 5** (merge map)**.** *A* merge map M *for a Boolean variable* u *over a finite set* X *of Boolean variables is a function from a finite set* N *of natural numbers satisfying, for each* $i \in N$*, either* $M(i) \in \{u, \bar{u}, *\}$ *or* $M(i) \in X \times N_{<i} \times N_{<i}$*, where* $N_{<i} := \{i' \in N : i' < i\}$*.*

A triple of the form $(x, a, b) \in X \times N_{<i} \times N_{<i}$ represents the instruction "if $x = 0$ then goto a else goto b", whereas the literals $\{u, \bar{u}, *\}$ represent output values.

We depict merge maps pictorially as DAGs. The nodes are the domain elements, and the leaf nodes as well as the directed edges are labelled by literals. In a merge map M, if $M(i)$ is a literal l, then node i is labeled l. If $M(i) = (x, a, b)$, then the DAG has the edge $i \to a$ labeled $\bar{x}$ and the edge $i \to b$ labeled x. As shown in Figure 1, the DAG naturally describes a deterministic branching program computing a Boolean function.

$$
\begin{array}{rcccl}
M & : & 1 & \mapsto & u \\
 & & 2 & \mapsto & \bar{u} \\
 & & 3 & \mapsto & (w, 1, 2) \\
 & & 4 & \mapsto & * \\
 & & 5 & \mapsto & (w, 4, 2) \\
 & & 6 & \mapsto & (v, 5, 3)
\end{array}
$$

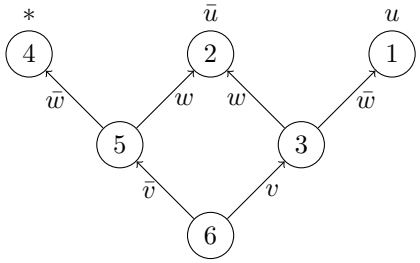


Figure 1 Function and branching program representations of a merge map M.

Relations. Merge Resolution uses two relations to determine preconditions for the binary operations. Firstly, we give M-Res the power to identify merge maps with equivalent representations, up to indexing. We term equivalent representations "isomorphic".

▶ **Definition 6** (isomorphism). *Two merge maps M_1 and M_2 for u over X with domains N_1 and N_2 are* isomorphic *(written $M_1 \simeq M_2$) iff there exists a bijection $f : N_1 \to N_2$ such that the following hold for each $i \in N_1$:*

(a) *if $M_1(i)$ is a literal in $\{u, \bar{u}, *\}$ then $M_2(f(i)) = M_1(i)$;*

(b) *if $M_1(i)$ is the triple (x, a, b) then $M_2(f(i)) = (x, f(a), f(b))$.*

Our second relation, *consistency*, simply identifies whether or not two merge maps agree on the intersection of their domains.

▶ **Definition 7** (consistency). *Two merge maps M_1 and M_2 for u over X with domains N_1 and N_2 are* consistent *(written $M_1 \bowtie M_2$) iff $M_1(i) = M_2(i)$ for each $i \in N_1 \cap N_2$.*

Operations. M-Res uses two binary operations to build merge maps for the resolvent based on those of the antecedents. The *select* operation identifies equivalent merge maps by means of the isomorphism relation. It also allows a *trivial* merge map to be discarded; we call a merge map trivial iff it is isomorphic to $1 \mapsto *$. (The operation is undefined if the merge maps are neither isomorphic nor do they contain a trivial map.)

▶ **Definition 8** (select). *Let M_1 and M_2 be merge maps for which $M_1 \simeq M_2$ or one of M_1, M_2 is trivial. Then* $\text{select}(M_1, M_2) := M_2$ *if M_1 is trivial, and* $\text{select}(M_1, M_2) := M_1$ *otherwise.*

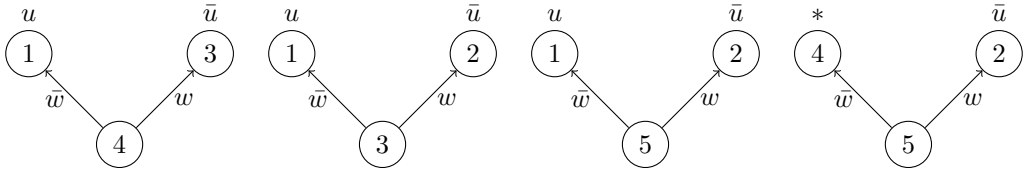
The *merge* operation allows two consistent merge maps to be combined as the children of a fresh query node. Antecedent maps are only ever merged for universal variables right of the pivot x. The inclusion of a natural number n allows the new query node to be identified with the resolvent, via its index in the proof sequence. In this way, query nodes are shared between later merge maps, rather than being duplicated; the result is a DAG-like structure which faithfully follows that of the derivation.

▶ **Definition 9** (merge). *Let M_1 and M_2 be consistent merge maps for u over X with domains N_1 and N_2, let $n > \max(N_1 \cup N_2)$ be a natural number, and let $x \in X$. Then* $\mathrm{merge}(M_1, M_2, n, x)$ *is the function from $N_1 \cup N_2 \cup \{n\}$ defined by*

$$\mathrm{merge}(M_1, M_2, n, x)(i) := \begin{cases} (x, \max(N_1), \max(N_2)) & \text{if } i = n, \\ M_1(i) & \text{if } i \in N_1, \\ M_2(i) & \text{if } i \in N_2 \setminus N_1. \end{cases}$$

▶ **Example 10.** For the merge maps depicted in Figure 2, isomorphism and consistency (or lack thereof) are as given in the table below. Furthermore, note that $\mathrm{select}(A, B) = \mathrm{select}(A, C) = A$ and $\mathrm{merge}(D, B, 6, v)$ gives the merge map from Figure 1.

relation	isomorphic	not isomorphic
consistent	$A \bowtie C$; $A \simeq C$	$B \bowtie D$; $B \not\simeq D$
not consistent	$A \not\bowtie B$; $A \simeq B$	$C \not\bowtie D$; $C \not\simeq D$



(a) Merge map A **(b)** Merge map B **(c)** Merge map C **(d)** Merge map D

Figure 2 Relations and operations on merge maps.

4.2 Definition of M-Res

We are now ready to put down the rules of Merge Resolution. Given a non-tautological clause C and a Boolean variable u, the *falsifying u-literal for C* is $\bar{l}$ if there is a literal $l \in C$ with $\mathrm{var}(l) = u$, and $*$ otherwise.

▶ **Definition 11** (merge resolution). *Let $\Phi := \mathcal{Q} \cdot \phi$ be a QBF with existential variables X and universal variables U. A* merge resolution *(*M-Res*)* derivation *of L_k from Φ is a sequence $\pi := L_1, \ldots, L_k$ of lines $L_i := (C_i, \{M_i^u : u \in U\})$ in which at least one of the following holds for each $i \in [k]$:*

(a) Axiom. *There exists a clause in $C \in \phi$ such that C_i is the existential subclause of C, and, for each $u \in U$, M_i^u is the merge map for u over $L_{\mathcal{Q}}(u)$ with domain $\{i\}$ mapping i to the falsifying u-literal for C;*

(b) Resolution. *There exist integers $a, b < i$ and an existential pivot $x \in X$ such that $C_i = \mathrm{res}(C_a, C_b, x)$ and, for each $u \in U$, either* (i) *$M_i^u = \mathrm{select}(M_a^u, M_b^u)$, or* (ii) *$x <_{\mathcal{Q}} u$ and $M_i^u = \mathrm{merge}(M_a^u, M_b^u, i, x)$.*

The final line L_k is the conclusion *of π, and π is a* refutation *of Φ iff $C_k = \emptyset$. The* size *of π is $|\pi| = k$.*

▶ **Example 12.** Consider the following M-Res refutation of the QBF with prefix $\exists x \forall u \exists t$ and matrix consisting of the clauses $\{x, u, t\}$, $\{\bar{x}, \bar{u}, t\}$, $\{x, u, \bar{t}\}$ and $\{\bar{x}, \bar{u}, \bar{t}\}$.

Line	Rule	C_i	M_i	Query
L_1	axiom	$\{x, t\}$	$1 \mapsto \bar{u}$	
L_2	axiom	$\{\bar{x}, t\}$	$2 \mapsto u$	
L_3	$\text{res}(L_1, L_2, x)$	$\{t\}$	$\text{merge}(M_1, M_2, 3, x)$	$3 \mapsto (x, 1, 2)$
L_4	axiom	$\{x, \bar{t}\}$	$4 \mapsto \bar{u}$	
L_5	axiom	$\{\bar{x}, \bar{t}\}$	$5 \mapsto u$	
L_6	$\text{res}(L_4, L_5, x)$	$\{t\}$	$\text{merge}(M_4, M_5, 6, x)$	$6 \mapsto (x, 4, 5)$
L_7	$\text{res}(L_3, L_6, t)$	$\{\}$	$\text{select}(M_3, M_6) = M_3$	

As shown in Figure 3, M_3 and M_6 are isomorphic, so $\text{select}(M_3, M_6)$ is defined and equal to M_3. For this reason, the resolution of antecedents L_3 and L_6 into L_7 is allowed, and the final merge map M_7 is simply a copy of M_3. The analogous resolution would be disallowed in reductionless LD-Q-Res because the pivot t is right of u, and the non-constant merge maps M_3 and M_6 would appear as merged literals $\{u, \bar{u}\}$ in the antecedent clauses.

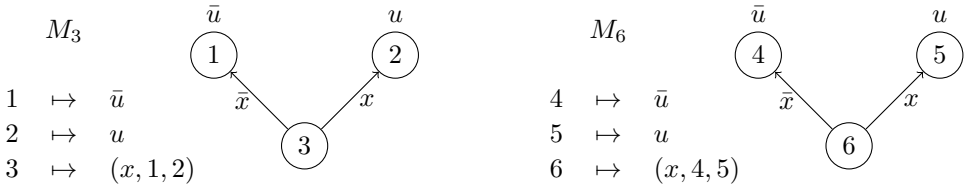


Figure 3 Functions and branching programs for merge maps M_3 and M_6 from Example 12.

Regarding M-Res proof size, observe that the domain of the merge map at line i is a subset of $[i]$. This means that merge maps grow linearly in the size of the derivation, and the size blow-up associated with the previous models [13, 37] is sidestepped. Moreover, number of lines is justifiably the correct size measure for M-Res.

4.3 Soundness and completeness of M-Res

The soundness of M-Res comes down to the fact that the merge maps at a given line form a partial strategy for the input QBF, in the technical sense of [37]. This means that any total existential assignment that falsifies the clause C_i will falsify the matrix when extended by the output of the merge maps M_i^u. Soundness is proved by induction on the proof structure with exactly this invariant. At the conclusion, all existential assignments falsify the empty clause C_k, and hence the M_k^u compute a countermodel.

▶ **Lemma 13.** *Let $(\emptyset, \{M^u : u \in U\})$ be the conclusion of an* M-Res *refutation of a QBF Φ. Then the functions computed by $\{M^u : u \in U\}$ are a countermodel for Φ.*

Completeness of M-Res is shown via the p-simulation of reductionless LD-Q-Res. The simulation copies precisely the structure of the reductionless LD-Q-Res refutation, while replacing merged literals by merge maps in the natural way.

▶ **Theorem 14.** M-Res *p-simulates reductionless* LD-Q-Res.

It is easy to see that M-Res refutations can be checked in polynomial time, since the isomorphism and consistency relations are computable in linear time.

▶ **Theorem 15.** M-Res *is a QBF proof system.*

5 Proof complexity: Merge Resolution vs Reductionless LD-Q-Res

In this section we exponentially separate M-Res from reductionless LD-Q-Res. The separating formulas are a kind of "squaring" of the equality formulas from Definition 2.

▶ **Definition 16** (squared equality formulas). *The* squared equality family *is the QBF family whose n^{th} instance* $\mathrm{EQ}^2(n) := \mathcal{Q}(n) \cdot \mathrm{eq}^2(n)$ *has prefix*

$$\mathcal{Q}(n) := \exists\{x_1, y_1, \ldots, x_n, y_n\}\forall\{u_1, v_1, \ldots, u_n, v_n\}\exists\{t_{i,j} : i, j \in [n]\},$$

and CNF matrix $\mathrm{eq}^2(n)$ *consisting of the clauses*

$$\begin{aligned}
&\{x_i, y_j, u_i, v_j, t_{i,j}\}, \quad \{x_i, \bar{y}_j, u_i, \bar{v}_j, t_{i,j}\}, \quad \text{for } i, j \in [n],\\
&\{\bar{x}_i, y_j, \bar{u}_i, v_j, t_{i,j}\}, \quad \{\bar{x}_i, \bar{y}_j, \bar{u}_i, \bar{v}_j, t_{i,j}\}, \quad \text{for } i, j \in [n],\\
&\{\bar{t}_{i,j} : i, j \in [n]\}.
\end{aligned}$$

The only winning strategy for the universal player is to set $u_i = x_i$ and $v_j = y_j$ for each $i, j \in [n]$. At the final block, the existential player is faced with the full set of $\{t_{i,j}\}$ unit clauses, and to satisfy all of them is to falsify the square clause $\{\bar{t}_{i,j} : i, j \in [n]\}$. No other strategy can be winning, as it would fail to produce all n^2 unit clauses.

5.1 $\mathrm{EQ}^2(n)$ lower bound for reductionless LD-Q-Res

We first give a formal definition of a refutation *path*; that is, a sequence of consecutive resolvents beginning with an axiom and ending at the conclusion.

▶ **Definition 17** (path). *Let π be a reductionless* LD-Q-Res *refutation. A* path *from a clause C in π is a subsequence $C_1, \ldots, C_k$ of π in which:*

- *$C = C_1$ is an axiom of π;*
- *C_k is the conclusion of π;*
- *for each $i \in [k-1]$, there exists a literal p_i and a clause R_i occurring before C_{i+1} in π such that $C_{i+1} = \mathrm{res}(C_i, R_i, p_i)$.*

The lower-bound proof is based upon two facts: (1) every total existential assignment corresponds to a path, all of whose clauses are consistent with the assignment (Lemma 18); (2) every path from the square clause contains a "wide" clause containing either all the x_i or all the y_j variables (Lemma 19). It is then possible to deduce the existence of exponentially many wide clauses, i.e. by considering the set of assignments for which each $x_i = y_i$ and each $t_{i,j} = 0$, all of whose corresponding paths begin at the square clause (proof of Theorem 20).

▶ **Lemma 18.** *Let π be a reductionless* LD-Q-Res *refutation of a QBF Φ, and let A be a clause with* $\mathrm{vars}(A) = \mathrm{vars}_\exists(\Phi)$. *Then there exists a path in π in which no existential literal outside of A occurs.*

Proof. We describe a procedure that constructs a sequence $P := C_k, \ldots, C_1$ of clauses in reverse order as follows: To begin with, let the "current clause" C_1 be the conclusion of π. As soon as the current clause C_i is in an axiom, the procedure terminates. Whenever necessary, obtain C_{i+1} as follows: find clauses R_1 and R_2 occurring before C_i in π and a literal $p \in A$ such that C_i is $\mathrm{res}(R_1, R_2, p)$, and set $C_{i+1} := R_1$ as the current clause. P is clearly a path in π by construction. By induction one shows that the existential subclause of C_i is a subset of A, for each $i \in [n]$: The base case $i = 1$ holds trivially since there are no existential literals in the conclusion C_1 of π. For the inductive step, observe that $C_{i+1} = C' \cup \{p\}$, for some subset $C' \subseteq C_i$ and literal $p \in A$. ◀

The second lemma is more technical, and its proof more involved. The proof works directly on the definition of path, the rules of reductionless LD-Q-Res, and the syntax of the squared equality formulas, to show the existence of the wide clause.

▶ **Lemma 19.** *Let $n \geq 2$, and let π be a reductionless* LD-Q-Res *refutation of* $\mathrm{EQ}^2(n)$*. On each path from $\{\bar{t}_{i,j} : i,j \in [n]\}$ in π, there occurs a clause C for which either $\{x_1,\ldots,x_n\} \subseteq \mathrm{vars}(C)$ or $\{y_1,\ldots,y_n\} \subseteq \mathrm{vars}(C)$.*

Proof. Put $X := \{x_1,\ldots,x_n\}$ and $Y := \{y_1,\ldots,y_n\}$. Call a clause R in π a *p-resolvent* if there exist earlier clauses R_1 and R_2 such that $R = \mathrm{res}(R_1, R_2, p)$.

Let $P := C_1,\ldots,C_k$ be a path from $\{\bar{t}_{i,j} : i,j \in [n]\}$ in π. With each C_l we associate an $n \times n$ matrix M_l in which $M_l[i,j] := 1$ if $\bar{t}_{i,j} \in C_i$ and $M_l[i,j] := 0$ otherwise. Let l be the least integer such that M_l has either a 0 in each row or a 0 in each column. Note that $l \geq 2$ since M_1 has no zeros.

We prove the lemma by showing that either $X \subseteq \mathrm{vars}(C_l)$ or $Y \subseteq \mathrm{vars}(C_l)$ must hold. We make use of the following claims, which hold for all $i,j \in [n]$:

(1) for each clause C on P, if $\bar{t}_{i,j} \in C$ then $\{u_i, \bar{u}_i\} \nsubseteq C$;
(2) each x_i-resolvent in π contains $\{u_i, \bar{u}_i\}$ as a subset;
(3) for each $t_{i,j}$-resolvent R in π, if $x_i \notin \mathrm{vars}(R)$ then $\{u_i, \bar{u}_i\} \subseteq R$.

Now, suppose that M_l has a 0 in each row. We proceed to show that every row in M_l also has at least one 1. To see this, suppose on the contrary that M_l contains a full 0 row r (this implies that $l \geq 2$, and hence that M_{l-1} exists). Note that by definition of resolution there can be at most one element that changes from 1 in M_{l-1} to 0 in M_l. Since M_{l-1} does not have a 0 in every column, it does not contain a full zero row. Hence it must be the case that the unique element that went from 1 in M_{l-1} to 0 in M_l is in row r. Since $n \geq 2$, we deduce that M_{l-1} has a 0 in each row, contradicting the minimality of l.

Let $i \in [n]$. Since the i^{th} row in M_l contains a 1, there is some $j \in [n]$ for which $\bar{t}_{i,j} \in C_l$. From claim (1) it follows that $\{u_i, \bar{u}_i\} \nsubseteq C_l$. Moreover, as universal literals accumulate along the path, this means that $\{u_i, \bar{u}_i\} \nsubseteq C_m$ for each $m \leq l$. Since the i^{th} row in M_l contains a 0, there exists $j' \in [n]$ such that $\bar{t}_{i,j'} \notin C_l$. As $\bar{t}_{i,j'} \in C_1$, there must be a $t_{i,j'}$-resolvent $C_{l'}$ on P with $l' \leq l$. Then we have $x_i \in \mathrm{vars}(C_{l'})$ by claim (3). Also, for each $m \leq l$, C_m is not an x_i-resolvent by claim (2). It follows that $x_i \in \mathrm{vars}(C_l)$. Since $i \in [n]$ was chosen arbitrarily, we have $X \subseteq \mathrm{vars}(C_l)$.

Suppose on the other hand that M_l does not contain a 0 in each row. Then M_l contains a 0 in each column. A symmetrical argument then shows that $Y \subseteq \mathrm{vars}(C_l)$.

It remains to prove the three claims.

(1) Observe that each clause in π containing the positive literal $t_{i,j}$ also contains the variable u_i (this holds for every axiom and universal literals are never removed). Let C be a clause on the path P for which $\bar{t}_{i,j} \in C$, and, for the sake of contradiction, suppose that $\{u_i, \bar{u}_i\} \subseteq C$. Since $u_i <_{\mathcal{Q}(n)} t_{i,j}$, there cannot be $t_{i,j}$-resolvent on P following C, as such a resolution step is explicitly forbidden in the rules of reductionless LD-Q-Res. This means that $\bar{t}_{i,j}$ occurs in C_k, the final clause of P. This is a contradiction, since C_k is the conclusion of π, which contains no existential literals. Therefore $\{u_i, \bar{u}_i\} \nsubseteq C$.

(2) Observe that each clause in π containing x_i (resp. $\bar{x}_i$) also contains u_i (resp. $\bar{u}_i$) (again, this holds for every axiom and universal literals are never removed). Let R be an x_i-resolvent of R_1 and R_2 in π. Since $x_i \in R_1$ and $\bar{x}_i \in R_2$, we must have $u_i \in R_1$ and $\bar{u}_i \in R_2$. It follows immediately that $\{u_i, \bar{u}_i\} \subseteq R$.

(3) Observe that each axiom in π containing the positive literal $t_{i,j}$ contains variable x_i. Hence, any clause in π that contains literal $t_{i,j}$ but not variable x_i must appear after an x_i-resolvent on some path, and therefore contains $\{u_i, \bar{u}_i\}$ by Claim (2). Now, let R be a $t_{i,j}$-resolvent of R_1 and R_2 in π. Suppose that $x_i \notin \text{vars}(R)$, which implies that $x_i \notin \text{vars}(R_1)$. Since $t_{i,j} \in R_1$, we have $\{u_i, \bar{u}_i\} \subseteq R_1$, and it follows that $\{u_i, \bar{u}_i\} \subseteq R$. ◀

It remains to prove the lower bound formally from the preceding lemmata.

▶ **Theorem 20.** *The squared equality family requires exponential-size reductionless* LD-Q-Res *refutations.*

Proof. Let $n \in \mathbb{N}$, and let π be a reductionless LD-Q-Res refutation of $\text{EQ}^2(n)$. We show that $|\pi| \geq 2^{n-1}$. The size bound is trivially true for $n = 1$, so we assume $n \geq 2$. Put $X := \{x_1, \ldots, x_n\}$ and $Y := \{y_1, \ldots, y_n\}$, and let $L := \{\bar{t}_{i,j} : i, j \in [n]\}$ be the long clause from $\text{eq}^2(n)$. We call a non-tautological clause S *symmetrical* iff $\text{vars}(S) = X \cup Y$ and $x_i \in S \Leftrightarrow y_i \in S$ for each $i \in [n]$. (A symmetrical clause represents a total assignment to $X \cup Y$). Note that there are 2^n distinct symmetrical clauses.

By Lemma 18, for each symmetrical clause S, there exists a path P_S in π in which all existential literals are contained in $S \cup L$. Moreover, each P_S begins at clause L, since every other clause in $\text{eq}^2(n)$ contains some positive $t_{i,j}$ literal that does not occur in $S \cup L$. By Lemma 19, on each path P from L in π there exists a clause C for which either $X \subseteq \text{vars}(C)$ or $Y \subseteq \text{vars}(C)$. It follows that we can define a function f that maps each symmetrical assignment S to a clause $f(S)$ in π for which either $\text{proj}(S, X) \subseteq f(S)$ or $\text{proj}(S, Y) \subseteq f(S)$. Moreover, since distinct symmetrical clauses S_1 and S_2 satisfy $\text{proj}(S_1, X) \neq \text{proj}(S_2, X)$ and $\text{proj}(S_1, Y) \neq \text{proj}(S_2, Y)$, each $f(S)$ is the image of at most two distinct symmetrical clauses. Hence, π contains at least 2^{n-1} clauses. ◀

Close inspection of the lower-bound proof reveals that particular resolution steps are blocked due to the appearance of merged literals in the antecedents (see the proof of claim (1) of Lemma 19). As we noted in Example 12, such steps remain blocked even if both merged literals implicitly represent the same (non-constant) function, in which case the resolution step is actually perfectly sound. As we will see, the M-Res upper-bound construction makes crucial use of the isomorphism of non-constant merge maps.

5.2 Short M-Res refutations of $\text{EQ}^2(n)$

Here we construct short M-Res refutations of the squared equality formulas. The approach is as follows. First, for each $i, j \in [n]$, obtain a line $(\{t_{i,j}\}, M_{i,j})$ by resolving the axioms for the four clauses in $\text{eq}(n)^2$ that contain $\{t_{i,j}\}$. By the natural application of the merge and select operations, one obtains merge maps $M_{i,j}$ in which the merge map for u_i outputs x_i with a single query, the merge map for v_j outputs y_j with a single query, and all other maps are trivial. Notice that all the non-trivial merge maps for a given universal variable are isomorphic, so these n^2 unit clauses can all be resolved against the square clause, utilising the select operation. It is precisely this final step which is blocked in reductionless LD-Q-Res.

▶ **Theorem 21.** *The squared equality family has $O(n^2)$-size* M-Res *refutations.*

The separation follows immediately from Theorems 20 and 21.

▶ **Theorem 22.** LD-Q-Res *does not p-simulate* M-Res *on QBF.*

6 Extending Merge Resolution to DQBF

In this section, we show that M-Res extends naturally to a DQBF proof system with the addition of a single weakening rule.

An *H-form dependency quantified Boolean formula* (DQBF) is denoted $\Phi := \mathcal{Q}{\cdot}\phi$. Similarly to QBF, the matrix ϕ is a CNF, but the quantifier prefix $\mathcal{Q}$ has a more general specification that allows variable dependencies to be written explicitly. Formally, $\mathcal{Q} := (X, U, L_{\mathcal{Q}})$, in which $X \subset \mathcal{Z}$ and $U \subset \mathcal{Z}$ are finite sets called the existential and universal variables of Φ, and $L_{\mathcal{Q}} : U \to \wp(X)$ is the *support set function*.

This is not the conventional notation for DQBF (cf. [2]), but it coincides conveniently with our QBF notation. In particular, our definition of "countermodel" need not change, and we call a DQBF false if it has a countermodel, and true if it does not. We redefine $<_{\mathcal{Q}}$ as a binary relation on $X \times U$ such that $x <_{\mathcal{Q}} u$ holds iff $x \in X$, $u \in U$ and $x \in L_{\mathcal{Q}}(u)$.

To lift M-Res to DQBF, we take Φ to be a DQBF in Definition 11 and add an extra case:

(c) Weakening. *There exists an integer $a < i$ such that C_i is an existential superclause of C_a and, for each $u \in U$, either* (i) $M_i^u = M_a^u$, *or* (ii) M_a^u *is trivial and* $M_i^u := i \mapsto l$ *for some literal* $l \in \{u, \bar{u}\}$.

By "existential superclause" it is meant that $\mathrm{vars}(C_i) \subseteq X$ and $C_a \subseteq C_i$.

Weakening is, in a clear sense, the simplest rule with which one extends M-Res to DQBF. Its function is merely to represent exactly the paths of the countermodel on which the canonical completeness construction is based. In general, the countermodel needs to be represented in full since merge maps must be isomorphic in order to apply the select operation.

Soundness and Completeness

Soundness of M-Res for DQBF is proved in the same way as for QBF, i.e. by showing that the concluding merge maps compute a countermodel. Lemma 13 lifts straightforwardly to DQBF, so we need only show that weakening preserves the induction invariant (see the paragraph preceding Lemma 13). This turns out to be rather straightforward, since a weakened clause is falsified by fewer existential assignments, and the weakening of a merge map always instantiates an undetermined assignment.

▶ **Lemma 23.** *Let $(\emptyset, \{M^u : u \in U\})$ be the conclusion of an* M-Res *refutation of a DQBF Φ. Then the functions computed by $\{M^u : u \in U\}$ form a countermodel for Φ.*

Completeness, on the other hand, cannot be established with an analogue of Theorem 14; DQBF is strictly larger than QBF, and hence simulation of reductionless LD-Q-Res does not guarantee completeness. Our proof rather extends the method by which completeness of reductionless LD-Q-Res was proved in Lemma 4; namely, the construction of a "full binary tree" of resolution steps based on the countermodel, following the prefix order of existential variables.

We give an overview of the construction. Let $\Phi := (X, U, L_{\mathcal{Q}}) \cdot \phi$ be a false DQBF with a countermodel h. For each $\alpha \in \langle X \rangle$, the assignment $\alpha \cup h(\alpha)$ falsifies some clause $C_\alpha \in \phi$ by definition of countermodel. Now, consider the M-Res line whose clause is the largest existential clause falsified by α and whose merge maps are constant functions computing $h(\alpha)$. Each such line can be derived in two M-Res steps, by weakening the axiom corresponding to C_α. Moreover, the clauses $\{C_\alpha : \alpha \in \langle X \rangle\}$ form the leaves of a full binary tree resolution refutation which can be completed using an arbitrary order of the existential pivots X. The merge maps are constructed by merging over the pivot x iff $x \in L_{\mathcal{Q}}(u)$; otherwise the select operation takes the merge map from either antecedent, since the full binary tree structure *guarantees* that they are isomorphic.

As merge maps essentially represent the structure of resolution steps in the subderivation, it is no surprise that the merge maps in our construction also have a full binary tree structure. This structure is captured by the following definition.

▶ **Definition 24** (binary tree merge map)**.** *A* binary tree merge map *for a variable u over a sequence of variables $x_1, \ldots, x_n$ is a function M with domain $[2^{n+1} - 1]$ and rule*

$$M(i) := \begin{cases} (x_{\lfloor \log i \rfloor + 1}, 2i, 2i+1) & \text{if } 1 \leq i < 2^n\,, \\ l_i & \text{if } 2^n \leq i < 2^{n+1}\,, \end{cases}$$

where each $l_i \in \{u, \bar{u}\}$.

At the technical level, we must define existential restrictions for DQBFs and DQBF countermodels. Let $\Phi := (X, U, L_\mathcal{Q}) \cdot \phi$ be a DQBF with a countermodel h and let l be a literal with $\mathrm{var}(l) = x \in X$. The *restriction* of Φ by l is $\Phi[l] := (X \setminus \{x\}, U, L'_\mathcal{Q}) \cdot \phi[l]$, where $L'_\mathcal{Q}$ maps each $u \in U$ to $L_\mathcal{Q}(u) \setminus \{x\}$. The restriction of h by l is $h[l] := \{h_u[l] : u \in U\}$, where the functions $h_u[l] : \langle L'_\mathcal{Q}(u) \rangle \to \{u, \bar{u}\}$ are defined by $h_u[l](\alpha) := h_u((\alpha \cup \{l\})\restriction_{L_\mathcal{Q}(u)})$.

The construction itself is defined recursively in the completeness proof, combining full binary tree refutations for $\Phi[x]$ and $\Phi[\bar{x}]$ for some $x \in X$ with a single resolution step. We use the fact that restrictions preserve countermodels in the following sense.

▶ **Proposition 25.** *Let h be a countermodel for a DQBF $\Phi := (X, U, L_\mathcal{Q}) \cdot \phi$ and let l be a literal with $\mathrm{var}(l) \in X$. Then $h[l]$ is a countermodel for $\Phi[l]$.*

As the final precursor to the completeness proof, we show that a derivation of the negated literal $\bar{l}$ and the restricted countermodel $h[l]$ can be obtained easily from a refutation of the restricted DQBF $\Phi[l]$.

▶ **Proposition 26.** *Let $\Phi := (X, U, L_\mathcal{Q}) \cdot \phi$ be a false DQBF, let l be a literal with $\mathrm{var}(l) \in X$, and let $(\emptyset, \{M_u : u \in U\})$ be the conclusion of be an* M-Res *refutation of $\Phi[l]$. Then there exists an* M-Res *derivation of $(\{\bar{l}\}, \{M_u : u \in U\})$ from Φ.*

Proof. Let π be the refutation with the given conclusion. The desired derivation may be obtained from π simply by adding the literal $\{\bar{l}\}$ to each clause, applying weakening where necessary, and adjusting the indexing of the merge maps to account for the extra weakening steps. ◀

▶ **Lemma 27.** *Every false H-form DQBF has an* M-Res *refutation.*

Proof. Let $\Phi := (X, U, L_\mathcal{Q}) \cdot \phi$ be a false DQBF, and let $X := \{x_1, \ldots, x_n\}$ where the x_i are pairwise distinct. For any M-Res refutation π with conclusion $(C_k, \{M_k^u : u \in U\})$, let $\{h_u : u \in U\}$ be the *concluding countermodel* for π, where the h_u are the functions computed by the *concluding merge maps* M_k^u. A merge map for $u \in U$ over $L_\mathcal{Q}(u)$ is said to be *complete* if it is isomorphic to a binary tree merge map for u over the sequence

$$x_{\sigma(1)}, \ldots x_{\sigma(|L_\mathcal{Q}(u)|)}\,,$$

which enumerates $L_\mathcal{Q}(u)$ in increasing index order; that is, $\sigma : [|L_\mathcal{Q}(u)|] \to [n]$ is the unique function satisfying $\{x_{\sigma(i)} : i \in [|L_\mathcal{Q}(u)|]\} = L_\mathcal{Q}(u)$ and $i < j \Leftrightarrow \sigma(i) < \sigma(j)$ for each $i, j \in [|L_\mathcal{Q}(u)|]$. By induction on the number n of existential variables, we show that, for each countermodel h for Φ, there exists an M-Res refutation whose concluding countermodel is h and whose concluding merge maps are complete. To that end, let $h := \{h_u : u \in U\}$ be an arbitrary countermodel for Φ.

For the base case $|X| = 0$, observe that each h_u is a constant function with some singleton codomain $\{l_u\}$. By definition of countermodel, there exists a clause $C \in \phi$ such that $C = \{\bar{l}_u : u \in \mathrm{vars}(C)\}$. Applying the axiom rule to C, one obtains a derivation of the line $(\emptyset, \{M^u : u \in U\})$ in which M^u computes the constant function h_u if $u \in \mathrm{vars}(C)$, and is trivial otherwise. With a single weakening step, each trivial M^u can be swapped for a merge map isomorphic to $1 \mapsto l_u$. Then each M^u is trivially complete and computes the constant function h_u.

For the inductive step, let $n \in \mathbb{N}$. Combining Propositions 25 and 26 with the inductive hypothesis, we deduce that there exist M-Res derivations π and π' of the lines $(\{\bar{x}_n\}, \{M_u : u \in U\})$ and $(\{x_n\}, \{M'_u : u \in U\})$ from Φ in which the M_u and M'_u are complete merge maps computing $h_u[x_n]$ and $h_u[\bar{x}_n]$. Assume that the lines of π are indexed from 1 to $|\pi|$ and that those of π' are indexed from $|\pi| + 1$ to $|\pi| + |\pi'|$. For each $u \in U$, the domains of M_u and M'_u are disjoint, so $M_u \bowtie M'_u$. If $x_n \notin L_Q(u)$, then $h_u[x_n] = h_u[\bar{x}_n]$, and we must have $M_u \simeq M'_u$ since complete merge maps computing the same function must be isomorphic. It follows that the line $(\emptyset, \{M''_u : u \in U\})$ can be derived from Φ, where

$$M''_u := \begin{cases} \mathrm{merge}(M_u, M'_u, |\pi| + |\pi'| + 1, x_i) & \text{if } x_i \in L_Q(u), \\ M_u & \text{if } x_i \notin L_Q(u). \end{cases}$$

It is easy to see that the M''_u are complete merge maps computing the h_u. ◀

The weakening rule is clearly polynomial-time checkable. Thus the following is immediate from Lemmata 23 and 27.

▶ **Theorem 28.** M-Res *is a proof system for H-form DQBF.*

It is natural to consider whether the weakening rule is necessary for completeness. This is indeed the case; there exist false DQBFs that cannot be refuted by M-Res without weakening.

For example, consider the DQBF $\Phi := (X, U, L_Q) \cdot \phi$ in which $X := \{x_1, x_2\}$, $U := \{u_1, u_2\}$, the support set function is given by $L_Q(u_1) = \{x_1\}, L_Q(u_2) = \{x_2\}$, and the matrix ϕ consists of the clauses

$$\{\bar{x}_1, \bar{x}_2, \bar{u}_1, \bar{u}_2\}, \{\bar{x}_1, x_2, \bar{u}_1, u_2\}, \{x_1, \bar{x}_2, u_1, \bar{u}_2\}, \{x_1, x_2, u_1\}.$$

It is readily verified that the only countermodel for this DQBF sets $u_1 = x_1$ and $u_2 = x_2$. However, the absence of variable u_2 in the clause $\{x_1, x_2, u_1\}$ means that the corresponding M-Res axiom has a merge map for u_2 isomorphic to $1 \mapsto *$. Since an M-Res refutation of Φ needs a full binary tree of resolution steps, this particular merge map must be instantiated at some point with a concrete literal $\bar{u}_2$ or u_2. To see this, observe that a resolution over x_1 must take place in which, among the antecedents, at least one merge map for u_2 (descended from axioms containing the negative literal $\bar{x}_1$) does not contain $*$ in its range; and since x_1 is not in $L_Q(u_2)$, the antecedents' merge maps for u_2 must be isomorphic.

7 Conclusions

We argue that building strategies *into* proofs is the natural way to deal with incompleteness for DQBF CDCL-systems [2]. The other approach, known as Fork Resolution [33], uses extension variables, and is not known to correspond to an existing implementation [35].

We also suggest that H-form (rather than S-form) DQBFs may be more suitable for CDCL-style solving, since associated proof systems "prove the existence of Herbrand functions". In the QBF realm, this is of course equivalent to proving the non-existence of Skolem

functions, but that does not carry over to DQBF (in a precise technical sense [2]). From this standpoint, it is natural to refute H-form DQBFs by finding the Herbrand functions that certify falsity. Moreover, it is unnatural to refute S-form DQBFs – which amounts to proving the non-existence of Skolem functions – by looking for Herbrand functions that may exist *even if the formula is true*. We suggest that this notion is the source of the soundness issues [12] associated with CDCL systems for DQBF.

Explicit representations may also be relevant for QBF solving. In dependency learning [31], variable dependencies are ignored until clause learning is blocked by an illegal merge. Our work demonstrates that many "illegal" merges are perfectly sound inferences; moreover, Merge Resolution provides a mechanism for identifying such cases based on isomorphism.

Particular implementations may want to fine-tune the details. Isomorphism is an easy way to determine the equivalence of two Boolean functions, but in general it seems unlikely that two equivalent functions will have identical representations. This points towards efficient (approximate) equivalence testing as the key to a successful implementation of M-Res.

References

1 QBFEVAL homepage. `http://www.qbflib.org/index_eval.php`. Accessed: 2018-09-26.

2 Valeriy Balabanov, Hui-Ju Katherine Chiang, and Jie-Hong R. Jiang. Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. *Theoretical Computer Science*, 523:86–100, 2014.

3 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF Certification and its Applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

4 Valeriy Balabanov, Jie-Hong Roland Jiang, Mikoláš Janota, and Magdalena Widl. Efficient Extraction of QBF (Counter)models from Long-Distance Resolution Proofs. In Blai Bonet and Sven Koenig, editors, *Conference on Artificial Intelligence (AAAI)*, pages 3694–3701. AAAI Press, 2015.

5 Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 5(1-4):133–191, 2008.

6 Olaf Beyerdorff, Joshua Blinkhorn, Leroy Chew, Renate Schmidt, and Martin Suda. Reinterpreting Dependency Schemes: Soundness Meets Incompleteness in DQBF. *Journal of Automated Reasoning (in press)*, 2018.

7 Olaf Beyersdorff and Joshua Blinkhorn. Dependency Schemes in QBF Calculi: Semantics and Soundness. In Michel Rueher, editor, *Principles and Practice of Constraint Programming (CP)*, volume 9892 of *Lecture Notes in Computer Science*, pages 96–112. Springer, 2016.

8 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, Cost and Capacity: A Semantic Technique for Hard Random QBFs. In Anna R. Karlin, editor, *ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

9 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower Bounds: From Circuits to QBF Proof Systems. In Madhu Sudan, editor, *ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 249–260. ACM, 2016.

10 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On Unification of QBF Resolution-Based Calculi. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 81–93. Springer, 2014.

11 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof Complexity of Resolution-based QBF Calculi. In Ernst W. Mayr and Nicolas Ollinger, editors, *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30 of *Leibniz International*

Proceedings in Informatics (LIPIcs), pages 76–89. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

12 Olaf Beyersdorff, Leroy Chew, Renate A. Schmidt, and Martin Suda. Lifting QBF Resolution Calculi to DQBF. In Nadia Creignou and Daniel Le Berre, editors, *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *Lecture Notes in Computer Science*, pages 490–499. Springer, 2016.

13 Nikolaj Bjørner, Mikolás Janota, and William Klieber. On Conflicts and Strategies in QBF. In Ansgar Fehnker, Annabelle McIver, Geoff Sutcliffe, and Andrei Voronkov, editors, *International Conference on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations (LPAR)*, volume 35 of *EPiC Series in Computing*, pages 28–41. EasyChair, 2015.

14 Samuel R. Buss. Towards NP-P via proof complexity and search. *Annals of Pure and Applied Logic*, 163(7):906–917, 2012.

15 Michael Cashmore, Maria Fox, and Enrico Giunchiglia. Partially Grounded Planning as Quantified Boolean Formula. In Daniel Borrajo, Subbarao Kambhampati, Angelo Oddi, and Simone Fratini, editors, *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2013.

16 Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, Cambridge, 2010.

17 Stephen A. Cook and Robert A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

18 Uwe Egly, Martin Kronegger, Florian Lonsing, and Andreas Pfandler. Conformant planning as a case study of incremental QBF solving. *Annals of Mathematics and Artificial Intelligence*, 80(1):21–45, 2017.

19 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-Distance Resolution: Proof Generation and Strategy Extraction in Search-Based QBF Solving. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 8312 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2013.

20 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of Bounded Synthesis. In Axel Legay and Tiziana Margaria, editors, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017.

21 Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, 2017.

22 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for Quantified Boolean Formulas. *Information and Computation*, 117(1):12–18, 1995.

23 William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A Non-prenex, Non-clausal QBF Solver with Game-State Learning. In Ofer Strichman and Stefan Szeider, editors, *Interenational Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 6175 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2010.

24 Roman Kontchakov, Luca Pulina, Ulrike Sattler, Thomas Schneider, Petra Selmer, Frank Wolter, and Michael Zakharyaschev. Minimal Module Extraction from DL-Lite Ontologies Using QBF Solvers. In Craig Boutilier, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 836–841. AAAI Press, 2009.

25 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1995.

26 Andrew C. Ling, Deshanand P. Singh, and Stephen Dean Brown. FPGA logic synthesis using quantified boolean satisfiability. In Fahiem Bacchus and Toby Walsh, editors, *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *Lecture Notes in Computer Science*, pages 444–450. Springer, 2005.

27 Hratch Mangassarian, Andreas G. Veneris, and Marco Benedetti. Robust QBF Encodings for Sequential Circuits with Applications to Verification, Debug, and Test. *IEEE Transactions on Computers*, 59(7):981–994, 2010.

28 Joao Marques-Silva and Sharad Malik. Propositional SAT Solving. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 247–275. Springer, 2018.

29 Jakob Nordström. On the interplay between proof complexity and SAT solving. *SIGLOG News*, 2(3):19–44, 2015.

30 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Long Distance Q-Resolution with Dependency Schemes. In Nadia Creignou and Daniel Le Berre, editors, *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *Lecture Notes in Computer Science*, pages 500–518. Springer, 2016.

31 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency Learning for QBF. In Serge Gaspers and Toby Walsh, editors, *International Conference on Theory and Practice of Satisfiability Testing (SAT)*, volume 10491 of *Lecture Notes in Computer Science*, pages 298–313. Springer, 2017.

32 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Polynomial-Time Validation of QCDCL Certificates. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *International Conference on Theory and Practice of Satisfiability Testing (SAT)*, volume 10929 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2018.

33 Markus N. Rabe. A Resolution-Style Proof System for DQBF. In Serge Gaspers and Toby Walsh, editors, *International Conference on Theory and Practice of Satisfiability Testing (SAT)*, volume 10491 of *Lecture Notes in Computer Science*, pages 314–325. Springer, 2017.

34 Horst Samulowitz, Jessica Davies, and Fahiem Bacchus. Preprocessing QBF. In Frédéric Benhamou, editor, *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4204 of *Lecture Notes in Computer Science*, pages 514–529. Springer, 2006.

35 Christoph Scholl and Ralf Wimmer. Dependency Quantified Boolean Formulas: An Overview of Solution Methods and Applications - Extended Abstract. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *International Conference on Theory and Practice of Satisfiability Testing (SAT)*, volume 10929 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2018.

36 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, pages 131–153. IOS Press, 2009.

37 Martin Suda and Bernhard Gleiss. Local Soundness for QBF Calculi. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *International Conference on Theory and Practice of Satisfiability Testing (SAT)*, volume 10929 of *Lecture Notes in Computer Science*, pages 217–234. Springer, 2018.

38 Moshe Y. Vardi. Boolean satisfiability: Theory and engineering. *Communications of the ACM*, 57(3):5, 2014.

39 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean Satisfiability solver. In *International Conference on Computer-aided Design (ICCAD)*, pages 442–449, 2002.

Tight Analysis of the Smartstart Algorithm for Online Dial-a-Ride on the Line

Alexander Birx
Institute of Mathematics and Graduate School CE, TU Darmstadt, Germany
birx@gsc.tu-darmstadt.de

Yann Disser
Institute of Mathematics and Graduate School CE, TU Darmstadt, Germany
disser@mathematik.tu-darmstadt.de

Abstract

The online Dial-a-Ride problem is a fundamental online problem in a metric space, where transportation requests appear over time and may be served in any order by a single server with unit speed. Restricted to the real line, online Dial-a-Ride captures natural problems like controlling a personal elevator. Tight results in terms of competitive ratios are known for the general setting and for online TSP on the line (where source and target of each request coincide). In contrast, online Dial-a-Ride on the line has resisted tight analysis so far, even though it is a very natural online problem.

We conduct a tight competitive analysis of the Smartstart algorithm that gave the best known results for the general, metric case. In particular, our analysis yields a new upper bound of 2.94 for open, non-preemptive online Dial-a-Ride on the line, which improves the previous bound of 3.41 [Krumke'00]. The best known lower bound remains 2.04 [SODA'17]. We also show that the known upper bound of 2 [STACS'00] regarding Smartstart's competitive ratio for closed, non-preemptive online Dial-a-Ride is tight on the line.

2012 ACM Subject Classification Theory of computation → Online algorithms; Mathematics of computing → Combinatorial optimization

Keywords and phrases dial-a-ride on the line, elevator problem, online algorithms, competitive analysis, smartstart, competitive ratio

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.15

Related Version `http://arxiv.org/abs/1901.04272`

Funding This work was supported by the "Excellence Initiative" of the German Federal and State Governments and the Graduate School CE at TU Darmstadt.

1 Introduction

Online optimization deals with settings where algorithmic decisions have to be made over time without knowledge of the future. A typical introductory example is the problem of controlling an elevator/conveyor system, where requests to transport passengers/goods arrive over time and the elevator needs to decide online how to adapt its trajectory along the real line. In terms of competitive analysis, the central question in this context is how much longer our trajectory will be in the worst-case, relative to an optimum offline solution that knows all requests ahead of time, i.e., we ask for solutions with good *competitive ratio*.

While the elevator problem is a natural online problem, even simplified versions of it have long resisted tight analysis. *Online* TSP *on the line* is such a simplification, where a single server on the real line needs to serve requests that appear over time at arbitrary positions by visiting their location, i.e., requests do not need to be transported. We distinguish the *closed* and *open* variants of this problem, depending on whether the server needs to eventually return to the origin or not. Determining the exact competitive ratios for either variant

had been an open problem for more than two decades [3, 5, 12, 13, 15, 16], when Bjelde et al. [4] were finally able to conduct a tight analysis that established competitive ratios of roughly 1.64 for the closed case and 2.04 for the open case.

The next step towards formally capturing the intuitive elevator problem is to allow transportation requests that appear over time; and to fix a capacity $c \in \mathbb{N} \cup \{\infty\}$ of the server that limits the number of transportation requests that can be served simultaneously. The resulting *online* Dial-a-Ride *problem on the line* has received considerable attention in the past [1, 4, 8, 13, 14, 16], but still resists tight analysis. The best known (non-preemptive) bounds put the competitive ratio in the range $[1.75, 2]$ for the closed variant (see [4, 1]). For the open variant the best known (non-preemptive) bounds put the competitiv ratio in the range $[2.04, 3]$ for $c = 1$ and in the range $[2.04, 3.41]$ for $c > 1$ (see [4, 13]). In this paper, we show an improved upper bound of (roughly) 2.94 for open online Dial-a-Ride on the line for arbitrary capacity $c \in \mathbb{N} \cup \{\infty\}$.

A straight-forward algorithm for online Dial-a-Ride on the line is the algorithm Ignore [1]: Whenever the server is idle and unserved requests R_t are present at the current time t, compute an optimum schedule to serve these requests from the current location, and follow this schedule while *ignoring* newly incoming requests. Ignore has a competitive ratio of exactly 4.[†] This competitive ratio can be improved by potentially waiting before starting the optimum schedule, in order to protect against requests that come in right after we decide to start. Ascheuer et al. [1] proposed the algorithm Smartstart (see Algorithm 1) that delays the execution of the optimum schedule until a certain time t relative to the length $L(t, p, R_t)$ of this schedule (formal definitions below).

Smartstart is parameterized by a factor $\Theta > 1$ that scales this waiting period. In this paper, we conduct a tight analysis of the best competitive ratio of Smartstart for open/closed online Dial-a-Ride on the line, over all parameter values $\Theta > 1$.

Results and techniques. The Smartstart algorithm is of particular importance for online Dial-a-Ride, since, on arbirary metric spaces, it achieves the best possible competitive ratio of 2 for the closed variant [1, 3], and the best known competitive ratio of $2 + \sqrt{2} \approx 3.41$ for the open variant [13]. We provide a conclusive treatment of this algorithm for online Dial-a-Ride on the line in terms of competitive analysis, both for the open and the closed variant.

Regarding the open case, we show that Smartstart attains a competitive ratio of $\rho^* \approx 2.94$ for parameter value $\Theta^* \approx 2.05$ (Section 3). To show this, we derive two separate upper bounds depending on Θ (cf. Figure 1): an upper bound $f_1(\Theta)$ for the case that Smartstart has a waiting period before starting its last schedule (Proposition 3.3), and an upper bound $f_2(\Theta)$ for the case that Smartstart begins its final schedule immediately (Proposition 3.4). The resulting general upper bound of $\max\{f_1(\Theta), f_2(\Theta)\}$ has its minimum precisely at the intersection point (Θ^*, ρ^*) of f_1 and f_2.

On the other hand, we show that for $\Theta \in (2, 3)$ there are instances where Smartstart waits before starting its final schedule and has competitive ratio at least $f_1(\Theta)$ (Proposition 4.2). Similarly, we show that for $\Theta \in [2, 2.303]$ there are instances where Smartstart does not wait before starting its final schedule and has competitive ratio at least $f_2(\Theta)$ (Proposition 4.3). Together, this implies that the general upper bound of $\max\{f_1(\Theta), f_2(\Theta)\}$ is tight for $\Theta \in (2, 2.303]$, and thus for $\Theta = \Theta^*$ (cf. Figure 1).

[†] The full proof can be found at `http://arxiv.org/abs/1901.04272`.

To complete our analysis of SMARTSTART, we give lower bound constructions for different domains of Θ (g_1 through g_4 in Figure 1) that establish that Θ^* is indeed the best parameter choice for SMARTSTART in the worst-case (Lemma 4.4). The key ingredient to all our lower bounds is a way to *lure* SMARTSTART away from the origin (Lemma 4.1).

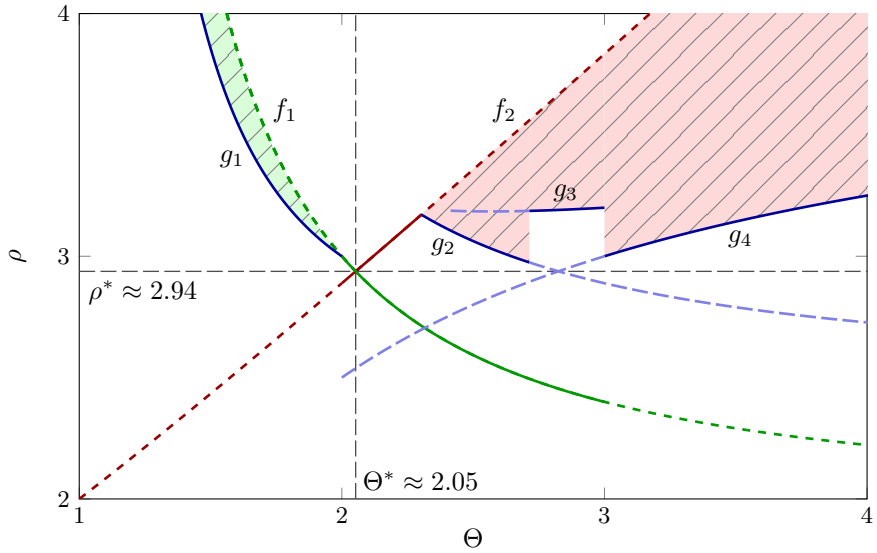


Figure 1 Overview over our bounds for SMARTSTART. The functions f_1 (green) / f_2 (red) are upper bounds for the cases where SMARTSTART waits / does not wait before starting the final schedule, respectively. The upper bounds are drawn solid in the domains where they are tight for their corresponding case. The functions g_1 through g_4 (blue) are general lower bounds; dashed continuations indicate how far these bounds could be extended.

Finally, for the closed variant of the problem, we provide a lower bound of 2 on the best-possible competitive ratio of SMARTSTART over all possible choices of the parameter $\Theta > 1$ (Section 5). This tightly matches the known upper bound for general metric spaces [1].

Significance. The main contribution of this paper is a conclusive treatment of the algorithm SMARTSTART for online DIAL-A-RIDE on the line in terms of competitive analysis. Additionally, our analysis yields an improved upper bound of (roughly) 2.94 for non-preemptive, open online DIAL-A-RIDE on the line. This is the first bound below 3 and narrows the gap for the competitive ratio to $[2.04, 2.94]$. Our work is likely to serve as a starting point towards devising better algorithms (preemptive or non-preemptive) that narrow the gaps for both the open and closed setting by avoiding critical "mistakes" of SMARTSTART, as evidenced by our lower bound constructions

Further related work. In this paper, we focus on the non-preemptive variant of online DIAL-A-RIDE on the line, where requests cannot be unloaded on the way in reaction to the arrival of new requests. For the case where preemption is allowed, the best known bounds for the closed version are $[1.64, 2]$ (see [3, 1]), which is slightly worse than the gap of $[1.75, 2]$ in the non-preemptive case. On the other hand, the best bounds for the open, preemptive variant are $[2.04, 2.41]$ (see [4]), which is better than the gap of $[2.04, 2.94]$ in the non-preemptive case. In particular, the preemptive and non-preemptive cases can currently not be separated in terms of competitive ratios.

A variant of the online Dial-a-Ride problem where the objective is to minimize the maximal flow time, instead of the makespan, has been studied by Krumke et al. [14, 15]. They established that in many metric spaces no online algorithm can be competitive with respect to this objective. Hauptmeier et al. [11] showed that a competitive algorithm is possible if we restrict ourselves to instances with "reasonable" load, which roughly means that requests that appear over a sufficiently large time period T can always be served in time at most T.

Lipmann et al. [17] studied a natural variant of closed, online Dial-a-Ride where the destinations of requests are only revealed upon collection by the server. For general metric spaces and server capacity c, they showed a tight competitive ratio of 3 in the preemptive setting, and lower/upper bounds of $\max\{3.12, c\}$ and $2c + 2$, respectively, in the non-preemptive setting.

Yi and Tian [18] considered the online Dial-a-Ride problem with deadlines, with the objective of serving the maximum number of requests. They provided bounds on the competitive ratio depending on the diameter of the metric space. In [19] they further studied this setting when the destination of requests are only revealed upon collection by the server.

The offline version of Dial-a-Ride on the line has been studied in various settings, for an overview see [7]. For the closed, non-preemptive case without release times, Gilmore and Gomory [9] and Atallah and Kosaraju [2] gave a polynomial time algorithm for a server with unit capacity $c = 1$, and Guan [10] showed that the problem is hard for $c = 2$. Bjelde et al. [4] extended this result to any finite $c \geq 2$ and both the open and closed case. They further showed that with release times the problem is already hard for finite $c \geq 1$. On the other hand, the complexity of the case $c = \infty$ has not yet been established. The closed, preemptive case without release times was shown to be polynomial time solvable for $c = 1$ by Atallah and Kosaraju [2], and for $c \geq 2$ by Guan [10].

For the closed, non-preemptive case with finite capacity, Krumke [13] provided a 3-approximation algorithm. Finally, Charikar and Raghavachari [6] gave approximation algorithms for the closed case without release times, both preemptive and non-preemptive, on general metric spaces. They also claimed to have a 2-approximation for the line, but this result appears to be incorrect (personal communication).

2 Preliminaries

Formally, an instance of Dial-a-Ride on the line is given by a set of requests denoted by $\sigma = \{(a_1, b_1; r_1), (a_2, b_2; r_2), \ldots, (a_n, b_n; r_n)\}$ that need to be served by a single server with capacity $c \in \mathbb{N} \cup \{\infty\}$, travelling with unit speed and starting at the origin on the real line. Request σ_i appears at time $r_i > 0$ at position $a_i \in \mathbb{R}$ of the real line and needs to be transported to position $b_i \in \mathbb{R}$. The objective of the Dial-a-Ride problem on the line is to find a shortest schedule for the server to transport all requests without carrying more than c requests at once, where the length of a schedule is the length of the resulting trajectory. In the *closed* version of the problem, the server eventually needs to return to the origin, in the *open* version it does not. In the *online* Dial-a-Ride problem on the line, each request σ_i is revealed only at time r_i, and n is only revealed implicitly by the fact that no more requests appear. In contrast, in the *offline* problem, all requests are known ahead of time (but release times still need to be respected).

We define $L(t, p, R)$ to be the length of a shortest schedule that starts at position p at time t and serves all requests in $R \subseteq \sigma$ after they appeared (i.e., the schedule must respect

release times). Observe that, for all $0 \leq t \leq t'$, $p, p' \in \mathbb{R}$, and $R \subseteq \sigma$, we have

$$L(t,p,R) \geq L(t',p,R), \tag{1}$$
$$L(t,p,R) \leq |p-p'| + L(t,p',R). \tag{2}$$

By $x_- := \min\{0, \min_{i=1,\dots,n}\{a_i\}, \min_{i=1,\dots,n}\{b_i\}\}$ we denote the leftmost and by $x_+ := \max\{0, \max_{i=1,\dots,n}\{a_i\}, \max_{i=1,\dots,n}\{b_i\}\}$ the rightmost position that needs to be visited by the server. Here and throughout, we orient the real line from left to right. Obviously, there is an optimum trajectory that only visits points in $[x_-, x_+]$, and we let Opt be such a trajectory and $\textsc{Opt}(\sigma) := L(0,0,\sigma)$ be its length.

Algorithm 1: Smartstart.

```
p_1 ← 0
for j = 1, 2, ... do
    while t ≤ L(t, p_j, R_t)/(Θ − 1) do
        wait
    t_j ← t
    S_j ← optimal offline schedule serving R_t starting from p_j
    execute S_j
    p_{j+1} ← current position
```

For the description of online algorithms, we denote by t the current time and by R_t the set of requests that have appeared until time t but have not been served yet. The algorithm Smartstart is given in Algorithm 1. Essentially, Smartstart waits before starting an optimal schedule to serve all available requests at time

$$\min_{t' \geq t} \left\{ t' \geq \frac{L(t', p, R_{t'})}{\Theta - 1} \right\}, \tag{3}$$

where p is the current position of the server and $\Theta > 1$ is a parameter of the algorithm that scales the waiting time. Importantly, Smartstart ignores incoming requests while executing a schedule. Whenever we need to distinguish the behavior of Smartstart for different values of $\Theta > 1$, we write $\textsc{Smartstart}_\Theta$ to make the choice of Θ explicit. The length of Smartstart's trajectory is denoted by $\textsc{Smartstart}(\sigma)$. Note that the schedules used by Smartstart are NP-hard to compute for $1 < c < \infty$, see [4].

We let $N \in \mathbb{N}$ be the number of schedules needed by Smartstart to serve σ. The j-th schedule is denoted by S_j, its starting time by t_j, its starting point by p_j, its ending point by p_{j+1} (cf. Algorithm 1), and the set of requests served in S_j by σ_{S_j}. For convenience, we set $t_0 = p_0 = 0$. Finally, we denote by $y_-^{S_j}$ the leftmost and by $y_+^{S_j}$ the rightmost position that occurs in the requests σ_{S_j}. Note that $y_-^{S_j}$ and $y_+^{S_j}$ need not lie on different sides of the origin, in contrast to $x_{-/+}$.

3 Upper Bound for the Open Version

In this section, we give an upper bound on the completion time

$$\textsc{Smartstart}(\sigma) = t_N + L(t_N, p_N, \sigma_{S_N}) \tag{4}$$

of Smartstart, relative to $\textsc{Opt}(\sigma)$. To do this, we consider two cases, depending on whether or not Smartstart waits after finishing schedule S_{N-1} and before starting the final

schedule S_N. If SMARTSTART waits, the starting time of schedule S_N is given by

$$t_N = \frac{1}{\Theta - 1} L(t_N, p_N, \sigma_{S_N}), \tag{5}$$

otherwise, we have

$$t_N = t_{N-1} + L(t_{N-1}, p_{N-1}, \sigma_{S_{N-1}}). \tag{6}$$

We start by giving a lower bound on the starting time of a schedule.[†]

▶ **Lemma 3.1.** *Algorithm* SMARTSTART *does not start schedule S_j earlier than time $\frac{|p_{j+1}|}{\Theta}$, i.e., we have $t_j \geq \frac{|p_{j+1}|}{\Theta}$.*

Proof sketch. Since SMARTSTART at least has to move from p_j to p_{j+1}, we have

$$L(t_j, p_j, \sigma_{S_j}) \geq |p_j - p_{j+1}|.$$

Note however that SMARTSTART needs at least time $|p_j|$ to reach p_j. Therefore, we have

$$\begin{aligned} t_j &\geq \min\{t \geq |p_j| \colon t + |p_j - p_{j+1}| \leq \Theta t\} \\ &= \min\left\{t \geq |p_j| \colon \frac{|p_j - p_{j+1}|}{\Theta - 1} \leq t\right\} \\ &= \max\left\{|p_j|, \frac{|p_j - p_{j+1}|}{\Theta - 1}\right\}. \end{aligned}$$

The claims now follows by showing $\max\left\{|p_j|, \frac{|p_j - p_{j+1}|}{\Theta - 1}\right\} \geq \frac{|p_{j+1}|}{\Theta}$. ◀

The following bound on the length of SMARTSTART's schedules is an essential ingredient in our upper bounds.

▶ **Lemma 3.2.** *For every schedule S_j of* SMARTSTART*, we have*

$$L(t_j, p_j, \sigma_{S_j}) \leq \left(1 + \frac{\Theta}{\Theta + 2}\right) \textsc{Opt}(\sigma).$$

Proof. First, we notice that by the triangle inequality we have

$$L(t_j, p_j, \sigma_{S_j}) \leq |p_j| + L(t_j, 0, \sigma_{S_j}) \leq \textsc{Opt}(\sigma) + |p_j|. \tag{7}$$

Now, let $\sigma_{S_j}^{\textsc{Opt}}$ be the first request of σ_{S_j} that is picked up by OPT and let $a_j^{\textsc{Opt}}$ be its starting point and $r_j^{\textsc{Opt}}$ be its release time. We have

$$L(t_j, p_j, \sigma_{S_j}) \leq |a_j^{\textsc{Opt}} - p_j| + L(t_j, a_j^{\textsc{Opt}}, \sigma_{S_j}), \tag{8}$$

again by the triangle inequality. Since OPT serves all requests of σ_{S_j} starting at position $a_j^{\textsc{Opt}}$ no earlier than time $r_j^{\textsc{Opt}}$, we have

$$L(t_j, a_j^{\textsc{Opt}}, \sigma_{S_j}) \overset{r_j^{\textsc{Opt}} \leq t_j}{\leq} L(r_j^{\textsc{Opt}}, a_j^{\textsc{Opt}}, \sigma_{S_j}) \leq \textsc{Opt}(\sigma) - r_j^{\textsc{Opt}}, \tag{9}$$

which yields

$$\begin{aligned} L(t_j, p_j, \sigma_{S_j}) &\overset{(8)}{\leq} |a_j^{\textsc{Opt}} - p_j| + L(t_j, a_j^{\textsc{Opt}}, \sigma_{S_j}) \\ &\overset{(9)}{\leq} \textsc{Opt}(\sigma) + |a_j^{\textsc{Opt}} - p_j| - r_j^{\textsc{Opt}} \\ &\overset{t_{j-1} < r_j^{\textsc{Opt}}}{<} \textsc{Opt}(\sigma) + |a_j^{\textsc{Opt}} - p_j| - t_{j-1}. \end{aligned} \tag{10}$$

Since p_j is the destination of a request, Opt needs to visit it. In the case that Opt visits p_j before collecting $\sigma_{S_j}^{\text{Opt}}$, Opt still has to collect and serve every request of σ_{S_j} after it has visited position p_j the first time, which directly implies

$$\left(1+\frac{\Theta}{\Theta+2}\right)\text{Opt}(\sigma) > \text{Opt}(\sigma) \geq L(|p_j|,p_j,\sigma_{S_j}) \overset{|p_j|\leq t_j}{\geq} L(t_j,p_j,\sigma_{S_j}).$$

On the other hand, if Opt collects $\sigma_{S_j}^{\text{Opt}}$ before visiting the position p_j, we have

$$t_{j-1}+|a_j^{\text{Opt}}-p_j| \overset{t_{j-1}<r_j^{\text{Opt}}}{<} r_j^{\text{Opt}}+|a_j^{\text{Opt}}-p_j| \leq \text{Opt}(\sigma), \tag{11}$$

since Opt cannot collect $\sigma_{S_j}^{\text{Opt}}$ before time r_j^{Opt} and then still has to visit position p_j. Thus, we have

$$\begin{aligned} L(t_j,p_j,\sigma_{S_j}) &\overset{(10)}{<} \text{Opt}(\sigma)+|a_j^{\text{Opt}}-p_j|-t_{j-1} \\ &\overset{(11)}{\leq} 2\text{Opt}(\sigma)-2t_{j-1} \\ &\overset{\text{Lem 3.1}}{\leq} 2\text{Opt}(\sigma)-2\frac{|p_j|}{\Theta}. \end{aligned} \tag{12}$$

This implies

$$L(t_j,p_j,\sigma_{S_j}) \overset{(7),(12)}{\leq} \min\left\{\text{Opt}(\sigma)+|p_j|, 2\text{Opt}(\sigma)-\frac{2}{\Theta}|p_j|\right\} \leq \left(1+\frac{\Theta}{\Theta+2}\right)\text{Opt}(\sigma),$$

since the minimum above is largest for $|p_j|=\frac{\Theta}{\Theta+2}\text{Opt}(\sigma)$. ◀

The following proposition uses Lemma 3.2 to provide an upper bound for the competitive ratio of Smartstart, in the case, where Smartstart does have a waiting period before starting the final schedule.

▶ **Proposition 3.3.** *In the case that* Smartstart *waits before executing S_N, we have*

$$\frac{\text{Smartstart}(\sigma)}{\text{Opt}(\sigma)} \leq f_1(\Theta) := \frac{2\Theta^2+2\Theta}{\Theta^2+\Theta-2}.$$

Proof. Assume Smartstart waits before starting the final schedule. Then we have

$$t_N+L(t_N,p_N,\sigma_{S_N})=\Theta t_N \tag{13}$$

by definition of Smartstart. This implies

$$\text{Smartstart}(\sigma) \overset{(4)}{=} t_N+L(t_N,p_N,\sigma_{S_N}) \overset{(13)}{=} \Theta t_N \overset{(5)}{=} \frac{\Theta}{\Theta-1}L(t_N,p_N,\sigma_{S_N}).$$

Lemma 3.2 thus yields the claimed bound:

$$\begin{aligned} \text{Smartstart}(\sigma) &= \frac{\Theta}{\Theta-1}L(t_N,p_N,\sigma_{S_N}) \\ &\overset{\text{Lem 3.2}}{\leq} \frac{\Theta}{\Theta-1}\left(1+\frac{\Theta}{\Theta+2}\right)\text{Opt}(\sigma) \\ &= \frac{2\Theta^2+2\Theta}{\Theta^2+\Theta-2}\text{Opt}(\sigma). \end{aligned}$$ ◀

It remains to examine the case, where the algorithm Smartstart has no waiting period before starting the final schedule.†

▶ **Proposition 3.4.** *If* Smartstart *does not wait before executing S_N, we have*

$$\frac{\textsc{Smartstart}(\sigma)}{\textsc{Opt}(\sigma)} \leq f_2(\Theta) := \left(\Theta + 1 - \frac{\Theta - 1}{3\Theta + 3}\right).$$

Proof sketch. If Smartstart starts S_N without waiting, its completion time is given by

$$\textsc{Smartstart}(\sigma) \overset{(6)}{=} t_{N-1} + L(t_{N-1}, p_{N-1}, \sigma_{S_{N-1}}) + L(t_N, p_N, \sigma_{S_N}). \tag{14}$$

Let $\sigma_{S_N}^{\textsc{Opt}}$ be the first request of σ_{S_N} that is picked up by Opt and let $a_N^{\textsc{Opt}}$ be its starting point and $r_N^{\textsc{Opt}}$ be its release time. Then we have

$$\textsc{Opt}(\sigma) \geq r_N^{\textsc{Opt}} + L(r_N^{\textsc{Opt}}, a_N^{\textsc{Opt}}, \sigma_{S_N}). \tag{15}$$

Using the triangle inequality as well as the definition of Smartstart, we obtain

$$\begin{aligned}
\textsc{Smartstart}(\sigma) &\overset{(14)}{=} t_{N-1} + L(t_{N-1}, p_{N-1}, \sigma_{S_{N-1}}) + L(t_N^{\textsc{Opt}}, p_N, \sigma_{S_N}) \\
&\overset{(3)}{\leq} \Theta t_{N-1} + L(t_N^{\textsc{Opt}}, p_N, \sigma_{S_N}) \\
&\overset{(1)}{\leq} \Theta t_{N-1} + |p_N - a_N^{\textsc{Opt}}| + L(t_N^{\textsc{Opt}}, a_N^{\textsc{Opt}}, \sigma_{S_N}) \\
&\overset{(15)}{\leq} \Theta t_{N-1} + |p_N - a_N^{\textsc{Opt}}| + \textsc{Opt}(\sigma) - r_N^{\textsc{Opt}} \\
&\overset{r_N^{\textsc{Opt}} > t_{N-1}}{<} (\Theta - 1) r_N^{\textsc{Opt}} + |p_N - a_N^{\textsc{Opt}}| + \textsc{Opt}(\sigma).
\end{aligned}$$

Clearly, $\textsc{Opt}(\sigma) \geq r_N^{\textsc{Opt}}$ since $\sigma_{S_N}^{\textsc{Opt}}$ cannot be served before this time, and $\textsc{Opt}(\sigma) \geq |p_N - a_N^{\textsc{Opt}}|$ since p_N must be the source or destination of a request (or the origin if $N = 1$) and must thus be visited by Opt. It follows from the above that $\textsc{Smartstart}(\sigma) \leq (\Theta + 1)\textsc{Opt}(\sigma)$. To get a better bound, we use that not both inequalities for $\textsc{Opt}(\sigma)$ can be tight simultaneously: From $\textsc{Opt}(\sigma) = r_N^{\textsc{Opt}}$ it follows that Opt finishes at position $a_N^{\textsc{Opt}}$. Assume that $\textsc{Opt}(\sigma) = |p_N - a_N^{\textsc{Opt}}|$ holds as well. Since Opt finishes at position $a_N^{\textsc{Opt}}$, this is only possible if $p_N = 0$ and $\textsc{Opt}(\sigma) = |a_N^{\textsc{Opt}}|$. Without loss of generality, there is no request $(0, 0; 0)$, hence Smartstart always waits before starting its first schedule, and thus a schedule S_{N-1} must exist. Because of $p_N = 0$, this schedule must end in the origin, which implies that there is some request that needs to be delivered to the origin after time 0. But this contradicts $\textsc{Opt}(\sigma) = |a_N^{\textsc{Opt}}|$, since Opt needs to deliver this request, too. The bound of the proposition is now obtained by carefully balancing $r_N^{\textsc{Opt}}$ and $|p_N - a_N^{\textsc{Opt}}|$. ◀

We combine the results of Proposition 3.3 and Proposition 3.4 to obtain the main result of this section.

▶ **Theorem 3.5.** *Let Θ^* be the only positive, real solution of $f_1(\Theta) = f_2(\Theta)$, i.e.,*

$$\Theta^* + 1 - \frac{\Theta^* - 1}{3\Theta^* + 3} = \frac{2\Theta^{*2} + 2\Theta^*}{\Theta^{*2} + \Theta^* - 2}.$$

Then, $\textsc{Smartstart}_{\Theta^*}$ *is ρ^*-competitive with $\rho^* := f_1(\Theta^*) = f_2(\Theta^*) \approx 2.93768$.*

Proof. For the case, where SMARTSTART does wait before starting the final schedule, we have established the upper bound

$$\frac{\text{SMARTSTART}(\sigma)}{\text{OPT}(\sigma)} \leq \frac{2\Theta^2+2\Theta}{\Theta^2+\Theta-2} = f_1(\Theta)$$

in Proposition 3.3 and for the case, where SMARTSTART starts the final schedule immediately after the second to final one, we have established the upper bound

$$\frac{\text{SMARTSTART}(\sigma)}{\text{OPT}(\sigma)} \leq \Theta + 1 - \frac{\Theta-1}{3\Theta+3} = f_2(\Theta)$$

in Proposition 3.4. Therefore the parameter for SMARTSTART with the smallest upper bound is

$$\Theta^* = \operatorname*{argmin}_{\Theta>1} \{\max\{f_1(\Theta), f_2(\Theta)\}\}.$$

We note that f_1 is strictly decreasing for $\Theta > 1$ and that f_2 is strictly increasing for $\Theta > 1$. Therefore the minimum above lies at the intersection point of f_1 and f_2 that is larger than 1, i.e., Θ^* is the only positive, real solution of

$$\Theta + 1 - \frac{\Theta-1}{3\Theta+3} = \frac{2\Theta^2+2\Theta}{\Theta^2+\Theta-2}.$$

The resulting upper bound for the competitive ratio is

$$\rho^* = f_1(\Theta^*) = f_2(\Theta^*) \approx 2.93768.$$ ◀

4 Lower Bound for the Open Version

In this section, we explicitly construct instances that demonstrate that the upper bounds given in the previous section are tight for certain ranges of $\Theta > 1$, in particular for $\Theta = \Theta^*$ (as in Theorem 3.5). Further, we show that choices of $\Theta > 1$ different from Θ^* yield competitive ratios worse than $\rho^* \approx 2.94$. Together, this implies that ρ^* is exactly the best possible competitive ratio for SMARTSTART.

All our lower bounds rely on the following lemma that gives a way to lure SMARTSTART away from the origin, with almost no time overhead. More specifically, the lemma provides a way to make SMARTSTART move to any position $p > 0$ within time $p + \mu$, where $\mu > 0$ is arbitrarily small.

▶ **Lemma 4.1.** *Let the capacity $c \in \mathbb{N} \cup \{\infty\}$ of the server be arbitrary but fixed, $p > 0$ be any position on the real line and $\mu > 0$ be any positive number. Furthermore, let $\delta > 0$ be such that $\frac{p}{\delta\Theta} = n \in \mathbb{N}$ and $\delta < (\Theta-1)\mu$. Algorithm* SMARTSTART *finishes serving the set of requests $\sigma = \{\sigma_1, \ldots, \sigma_{n+1}\}$ with*

$$\begin{aligned}\sigma_1 &= (\delta, \delta; 0),\\ \sigma_i &= \left(i\delta, i\delta; \frac{1}{\Theta-1}\delta + (i-1)\delta\right) \text{ for } i \in \{2, \ldots, n\}\\ \sigma_{n+1} &= (p, p; \mu + n\delta)\end{aligned}$$

and reaches the position p at time $p + \mu$, provided that no additional requests appear until time $\frac{p}{\Theta} + \mu$.

Proof. We show via induction that every request σ_i with $i \in \{1, \dots, n\}$ is served in a separate schedule S_i with starting position $p_i = (i-1)\delta$ and starting time

$$t_i = \frac{1}{\Theta - 1}\delta + (i-1)\delta.$$

This is clear for $i = 1$: By definition, SMARTSTART starts from $p_1 = 0$. The schedule S_1 to serve σ_1 is started at time

$$t_1 = \min\left\{ t \geq 0 \;\middle|\; \frac{L(t, 0, \{\sigma_1\})}{\Theta - 1} \leq t \right\} = \frac{1}{\Theta - 1}\delta,$$

and reaches position δ at time $\frac{1}{\Theta-1}\delta + \delta = \frac{\Theta}{\Theta-1}\delta$. Note that the release time of every request σ_i is larger than t_1, ensuring that S_1 indeed only serves σ_1.

We assume the claim is true for some $k \in \{1, \dots, n-1\}$. Consider $i = k+1$. By reduction, the server finishes schedule S_k at position $p_{k+1} = k\delta$ at time $\frac{1}{\Theta-1}\delta + k\delta$. Therefore, we have

$$t_{k+1} \geq \frac{1}{\Theta - 1}\delta + k\delta.$$

On the other hand, we have

$$\frac{L\left(\frac{\delta}{\Theta-1} + k\delta, k\delta, \{\sigma_{k+1}\}\right)}{\Theta - 1} = \frac{\delta}{\Theta - 1} < \frac{1}{\Theta - 1}\delta + k\delta.$$

Since there are no other unserved requests at time $\frac{1}{\Theta-1}\delta + k\delta$, the schedule S_{k+1} is started at time $t_{k+1} = \frac{1}{\Theta-1}\delta + k\delta$ and only serves σ_{k+1} as claimed. It remains to examine the final request σ_{n+1}. The above shows that in the schedule S_n is finished at time

$$t_n + L(t_n, p_n, \{\sigma_n\}) = \frac{1}{\Theta - 1}\delta + (n-1)\delta + \delta = \frac{1}{\Theta - 1}\delta + n\delta < \mu + n\delta$$

at position $n\delta = \frac{p}{\Theta}$, i.e., before the request σ_{n+1} is released at time $\mu + n\delta$. On the other hand, we have

$$\frac{L\left(\mu + n\delta, \frac{p}{\Theta}, \{\sigma_{n+1}\}\right)}{\Theta - 1} = \frac{\frac{\Theta-1}{\Theta}p}{\Theta - 1} = \frac{p}{\Theta} = n\delta < \mu + n\delta.$$

Therefore the final schedule S_{n+1} is started at time $t_{n+1} = \mu + n\delta = \mu + \frac{p}{\Theta}$, and we get

$$\begin{aligned} \textsc{Smartstart}((\sigma_i)_{i\in\{1,\dots,n+1\}}) &= t_{n+1} + L(t_{n+1}, p_{n+1}, \{\sigma_{n+1}\}) \\ &= \mu + \frac{p}{\Theta} + \frac{\Theta - 1}{\Theta}p \\ &= \mu + p. \end{aligned}$$

Note that for every request the starting point is identical to the ending point. Thus, our construction remains valid for every capacity $c \in \mathbb{N} \cup \{\infty\}$. Furthermore, there is no interference with requests that are released after time $t_{n+1} = \mu + \frac{p}{\Theta}$. ◀

Equipped with this strategy to lure SMARTSTART away from the origin, we now move on to establish lower bounds matching Propositions 3.3 and 3.4.†

▶ **Proposition 4.2.** *Let the capacity $c \in \mathbb{N} \cup \{\infty\}$ of the server be arbitrary but fixed and let $2 < \Theta < 3$. For every sufficiently small $\varepsilon > 0$, there is a set of requests σ such that* SMARTSTART *waits before starting the final schedule and such that the inequality*

$$\frac{\textsc{Smartstart}(\sigma)}{\textsc{Opt}(\sigma)} \geq \frac{2\Theta^2 + 2\Theta}{\Theta^2 + \Theta - 2} - \varepsilon$$

holds, i.e., the upper bound established in Proposition 3.3 is tight for $\Theta \in (2, 3)$.

Proof sketch. We start by luring SMARTSTART to position 1 via Lemma 4.1. This can be done such that the schedule ending in 1 starts at time $\mu + \frac{1}{\Theta}$ for some sufficiently small $\mu > 0$ Immediately after the start of this schedule, we add a series of non-overlapping requests that require the server to move to position $-\frac{1}{\Theta}$ and afterwards to position 1. We can show that OPT serves the resulting set of requests simply by moving to $-\frac{1}{\Theta}$ and then straight to 1. On the other hand, independent of the capacity, SMARTSTART needs to cross the space between the origin and point 1 two more times. A quantitative analysis of this setting yields the claimed bound. ◀

▶ **Proposition 4.3.** *Let the capacity $c \in \mathbb{N} \cup \{\infty\}$ of the server be arbitrary but fixed and let $2 \leq \Theta \leq \frac{1}{2}(1+\sqrt{13})$. For every sufficiently small $\varepsilon > 0$ there is a set of requests σ such that* SMARTSTART *immediately starts S_N after S_{N-1} and such that*

$$\frac{\text{SMARTSTART}(\sigma)}{\text{OPT}(\sigma)} \geq \Theta + 1 - \frac{\Theta - 1}{3\Theta + 3} - \varepsilon,$$

i.e., the upper bound established in Proposition 3.4 is tight for $\Theta \in [2, \frac{1}{2}(1+\sqrt{13})] \approx [2, 2.303]$.

Proof. Let $\varepsilon > 0$ with $\varepsilon < \frac{1}{5\Theta}\frac{3\Theta^2-\Theta}{3\Theta+3}$ and $\varepsilon' = \frac{3\Theta+3}{3\Theta^2-\Theta}\varepsilon$. We apply Lemma 4.1 with $p = 1$ and $\mu = \frac{\varepsilon'}{2}$. For convenience, we start the enumeration of the schedules with the first schedule after the application of Lemma 4.1. Algorithm SMARTSTART reaches position $p_1 = 1$ at time $1 + \frac{\varepsilon'}{2}$. Now let the requests

$$\sigma_1^{(1)} = \left(2 + \frac{1}{\Theta} - \varepsilon', 2 + \frac{1}{\Theta} - \varepsilon'; \frac{1}{\Theta} + \varepsilon'\right),$$
$$\sigma_1^{(2)} = \left(-\frac{1}{\Theta}, -\frac{1}{\Theta}; \frac{1}{\Theta} + \varepsilon'\right)$$

appear. Note that both requests are released after time $\frac{1}{\Theta} + \frac{\varepsilon'}{2}$ and, therefore, do not interfere with the application of Lemma 4.1. If SMARTSTART serves $\sigma_1^{(2)}$ before serving $\sigma_1^{(1)}$ the time it needs is at least

$$\left|1 - \left(-\frac{1}{\Theta}\right)\right| + \left|\left(-\frac{1}{\Theta}\right) - \left(2 + \frac{1}{\Theta} - \varepsilon'\right)\right| = 1 + \frac{1}{\Theta} + 2 + \frac{2}{\Theta} - \varepsilon' = 3 + \frac{3}{\Theta} - \varepsilon'.$$

The best schedule that serves $\sigma_1^{(2)}$ after serving $\sigma_1^{(1)}$ needs time

$$\left|1 - \left(2 + \frac{1}{\Theta} - \varepsilon'\right)\right| + \left|\left(2 + \frac{1}{\Theta} - \varepsilon'\right) - \left(-\frac{1}{\Theta}\right)\right| = 1 + \frac{1}{\Theta} - \varepsilon' + 2 + \frac{2}{\Theta} - \varepsilon' = 3 + \frac{3}{\Theta} - 2\varepsilon'.$$

Thus, SMARTSTART serves $\sigma_1^{(2)}$ after serving $\sigma_1^{(1)}$, and, for all $t \geq 1 + \frac{\varepsilon'}{2}$, we obtain

$$L\left(t, p_1, \{\sigma_1^{(1)}, \sigma_1^{(2)}\}\right) = L\left(t, 1, \{\sigma_1^{(1)}, \sigma_1^{(2)}\}\right) = 3 + \frac{3}{\Theta} - 2\varepsilon'.$$

By assumption, we have $\Theta \leq \frac{1}{2}(1+\sqrt{13})$ and $\varepsilon < \frac{1}{5\Theta}\frac{3\Theta^2-\Theta}{3\Theta+3}$, i.e., $\varepsilon' < \frac{1}{5\Theta} < 1$, which implies

that for the time $1+\frac{\varepsilon'}{2}$, when SMARTSTART reaches position $p_1 = 1$, the inequality

$$\begin{aligned}
\frac{L\left(1+\frac{\varepsilon'}{2}, p_1, \{\sigma_1^{(1)}, \sigma_1^{(2)}\}\right)}{\Theta - 1} &= \frac{3+\frac{3}{\Theta}-2\varepsilon'}{\Theta-1} \\
&= \frac{3-2\varepsilon'}{\Theta-1} + \frac{3}{\Theta(\Theta-1)} \\
&\overset{1<\Theta\le\frac{1}{2}(1+\sqrt{13})}{\geq} \frac{3-2\varepsilon'}{\frac{1}{2}(\sqrt{13}-1)} + \frac{3}{\frac{1}{4}(\sqrt{13}-1)(1+\sqrt{13})} \\
&= \frac{3-2\varepsilon'}{\frac{1}{2}(\sqrt{13}-1)} + 1 \\
&\overset{\frac{1}{2}(\sqrt{13}-1)<2}{>} 1+\frac{\varepsilon'}{2}
\end{aligned}$$

holds. Thus, SMARTSTART has a waiting period and starts schedule S_1 at time

$$\begin{aligned}
t_1 &= \min\left\{ t \ge 1+\frac{\varepsilon'}{2} \;\middle|\; \frac{L(t, p_1, \{\sigma_1^{(1)}, \sigma_1^{(2)}\})}{\Theta-1} \le t \right\} \\
&= \min\left\{ t \ge 1+\frac{\varepsilon'}{2} \;\middle|\; \frac{3+\frac{3}{\Theta}-2\varepsilon'}{\Theta-1} \le t \right\} \\
&= \frac{3+\frac{3}{\Theta}-2\varepsilon'}{\Theta-1} \\
&= \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2\varepsilon'}{\Theta-1}.
\end{aligned}$$

Next, we let the final request

$$\sigma_2 = \left(\frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2}{\Theta} - \varepsilon', \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2}{\Theta} - \varepsilon'; \frac{3\Theta+3}{\Theta(\Theta-1)}\right)$$

appear. SMARTSTART finishes schedule S_1 at time

$$t_1 + L(t_1, p_1, \{\sigma_1^{(1)}, \sigma_1^{(2)}\}) = \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2\varepsilon'}{\Theta-1} + 3 + \frac{3}{\Theta} - 2\varepsilon' = \frac{3\Theta+3}{\Theta-1} - \frac{2\Theta\varepsilon'}{\Theta-1}$$

at position $p_2 = -\frac{1}{\Theta}$. For all $t \ge \frac{3\Theta+3}{\Theta-1} - \frac{2\Theta}{\Theta-1}\varepsilon'$, we obtain

$$L\left(t, -\frac{1}{\Theta}, \{\sigma_2\}\right) = \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{1}{\Theta} - \varepsilon'.$$

By assumption, we have $2 \le \Theta \le \frac{1}{2}(1+\sqrt{13}) < 3$ and $\varepsilon < \frac{1}{5\Theta}\frac{3\Theta^2-\Theta}{3\Theta+3}$, i.e., $\varepsilon' < \frac{1}{5\Theta}$, which implies that, for the finishing time $\frac{3\Theta+3}{\Theta-1} - \frac{2\Theta\varepsilon'}{\Theta-1}$ of schedule S_1, the inequality

$$\begin{aligned}
\frac{L\left(\frac{3\Theta+3}{\Theta-1} - \frac{2\Theta\varepsilon'}{\Theta-1}, -\frac{1}{\Theta}, \{\sigma_2\}\right)}{\Theta-1} &= \frac{3\Theta+3}{\Theta(\Theta-1)^2} - \frac{1+\Theta\varepsilon'}{\Theta(\Theta-1)} \\
&\overset{\Theta\ge 2}{<} \frac{3\Theta+3}{\Theta-1} - \frac{1+\Theta\varepsilon'}{\Theta(\Theta-1)} \\
&\overset{1>5\Theta\varepsilon'}{<} \frac{3\Theta+3}{\Theta-1} - \frac{6\varepsilon'}{\Theta-1} \\
&\overset{\Theta\le 3}{<} \frac{3\Theta+3}{\Theta-1} - \frac{2\Theta\varepsilon'}{\Theta-1}
\end{aligned} \tag{16}$$

holds. (Note that inequality (16) still holds for slightly smaller Θ if we let $\varepsilon \to 0$.) Because of inequality (16), the final schedule S_2 is started at time

$$t_2 = \frac{3\Theta+3}{\Theta-1} - \frac{2\Theta\varepsilon'}{\Theta-1}$$

without waiting. To sum it up, we have

$$\begin{aligned}\textsc{SmartStart}(\sigma) &= t_2 + L(t_2, p_2, \{\sigma_2\}) \\ &= \frac{3\Theta+3}{\Theta-1} - \frac{2\Theta\varepsilon'}{\Theta-1} + \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{1}{\Theta} - \varepsilon' \\ &= \frac{3\Theta+3}{\Theta-1} + \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{1}{\Theta} - \frac{3\Theta-1}{\Theta-1}\varepsilon'.\end{aligned}$$

On the other hand, Opt goes from the origin straight to position $-\frac{1}{\Theta}$ serving request $\sigma_1^{(2)}$ at time $\frac{1}{\Theta}+\varepsilon'$ (i.e., it has to wait for ε' units of time after it reaches position $-\frac{1}{\Theta}$) and returns to the origin at time $\frac{2}{\Theta}+\varepsilon'$. Let $q > 0$ be the position of a request that has occurred by the application of Lemma 4.1 at the beginning of this proof. Then this request is released earlier than time $q + \frac{\varepsilon'}{2}$. Since Opt reaches position q not earlier than time $\frac{2}{\Theta}+\varepsilon'+q > q+\frac{\varepsilon'}{2}$, Opt can go straight from the origin to the right and can serve all remaining requests without waiting. Note that the position $\frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2}{\Theta} - \varepsilon'$ of σ_2 is equal to or to right of the position $2+\frac{1}{\Theta}-\varepsilon'$ of $\sigma_1^{(2)}$ because of $\Theta \le \frac{1}{2}(1+\sqrt{13})$. Thus, Opt finishes at position $\frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2}{\Theta} - \varepsilon'$ and we have

$$\begin{aligned}\textsc{Opt}(\sigma) &= \left|0 - \left(-\frac{1}{\Theta}\right)\right| + \varepsilon' + \left|-\frac{1}{\Theta} - \left(\frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2}{\Theta} - \varepsilon'\right)\right| \\ &= \frac{1}{\Theta} + \varepsilon' + \frac{1}{\Theta} + \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{2}{\Theta} - \varepsilon' \\ &= \frac{3\Theta+3}{\Theta(\Theta-1)}.\end{aligned}$$

Note that Opt can do this even if $c = 1$ since for all requests the starting point is equal to the destination. Since we have $\varepsilon' = \frac{3\Theta+3}{3\Theta^2-\Theta}\varepsilon$, we finally obtain

$$\begin{aligned}\frac{\textsc{SmartStart}(\sigma)}{\textsc{Opt}(\sigma)} &= \frac{\frac{3\Theta+3}{\Theta-1} + \frac{3\Theta+3}{\Theta(\Theta-1)} - \frac{1}{\Theta} - \frac{3\Theta-1}{\Theta-1}\varepsilon'}{\frac{3\Theta+3}{\Theta(\Theta-1)}} \\ &= \Theta + 1 - \frac{\Theta-1}{3\Theta+3} - \frac{3\Theta^2-\Theta}{3\Theta+3}\varepsilon' \\ &= \Theta + 1 - \frac{\Theta-1}{3\Theta+3} - \varepsilon,\end{aligned}$$

as claimed. ◀

Recall that the optimal parameter Θ^* established in Theorem 3.5 is the only positive, real solution of the equation

$$\Theta + 1 - \frac{\Theta-1}{3\Theta+3} = \frac{2\Theta^2+2\Theta}{\Theta^2+\Theta-2},$$

which is $\Theta^* \approx 2.0526$. Therefore, according to Proposition 4.2 and Proposition 4.3 the parameter Θ^* lies in the ranges where the upper bounds of Propositions 3.3 and 3.4 are both tight. It remains to make sure that for all Θ that lie outside of this range the competitive ratio of $\textsc{SmartStart}_\Theta$ is larger than $\rho^* \approx 2.93768$.[†]

▶ **Lemma 4.4.** *Let*

$$I_1 = (1, 2], \quad I_2 = (\tfrac{1}{2}(1+\sqrt{13}), 1+\sqrt{2}], \quad I_3 = (1+\sqrt{2}, 3), \quad I_4 = [3, \infty)$$

be intervals. For every $i \in \{1, 2, 3, 4\}$ there is a set of requests σ, such that, for all $\Theta \in I_i$,

$$\frac{\textsc{Smartstart}(\sigma)}{\textsc{Opt}(\sigma)} > \rho^* \approx 2.93768.$$

Our main theorem now follows from Theorem 3.5 combined with Propositions 4.2 and 4.3, as well as Lemma 4.4.

▶ **Theorem 4.5.** *The competitive ratio of* $\textsc{Smartstart}_{\Theta^*}$ *is exactly*

$$\rho^* = f_1(\Theta^*) = f_2(\Theta^*) \approx 2.93768.$$

For every other $\Theta > 1$ with $\Theta \neq \Theta^*$ the competitive ratio of $\textsc{Smartstart}_{\Theta}$ *is larger than ρ^*.*

Proof. We have shown in Proposition 4.2 that the upper bound

$$\frac{\textsc{Smartstart}(\sigma)}{\textsc{Opt}(\sigma)} \leq f_1(\Theta) = \frac{2\Theta^2 + 2\Theta}{\Theta^2 + \Theta - 2}$$

established in Proposition 3.3 for the case, where Smartstart waits before starting the final schedule, is tight for all $\Theta \in (2, 3)$. Furthermore, we have shown in Proposition 4.3 that the upper bound

$$\frac{\textsc{Smartstart}(\sigma)}{\textsc{Opt}(\sigma)} \leq f_2(\Theta) = \left(\Theta + 1 - \frac{\Theta - 1}{3\Theta + 3}\right)$$

established in Proposition 3.4 for the case, where Smartstart does not wait before starting the final schedule, is tight for all $\Theta \in (2, \frac{1}{2}(1+\sqrt{13})]$. Since $\Theta^* \approx 2.0526$ lies in those ranges, the competitive ratio of $\textsc{Smartstart}_{\Theta^*}$ is indeed exactly ρ^*.

It remains to show that for every $\Theta > 1$ with $\Theta \neq \Theta^*$ the competitive ratio is larger. First, according to Lemma 4.4, the competitive ratio of Smartstart with parameter $\Theta \in (1, 2]$ or $\Theta \in (\frac{1}{2}(1+\sqrt{13}), \infty)$ is larger than ρ^*. By monotonicity of f_1, every function value in $(2, \Theta^*)$ is larger than $f_1(\Theta^*) = \rho^*$. Thus, the competitive ratio of Smartstart with parameter $\Theta \in (2, \Theta^*)$ is larger than ρ^*, since f_1 is tight on $(2, \Theta^*)$ by Proposition 4.2. Similarly, by monotonicity of f_2, every function value in $(\Theta^*, \frac{1}{2}(1+\sqrt{13})]$ is larger than $f_2(\Theta^*) = \rho^*$. Thus, the competitive ratio of Smartstart with parameter $\Theta \in (\Theta^*, \frac{1}{2}(1+\sqrt{13})]$ is larger than ρ^*, since f_1 is tight on $(\Theta^*, \frac{1}{2}(1+\sqrt{13})]$ by Proposition 4.3. ◀

5 Lower Bound for the Closed Version

We provide a lower bound for Smartstart for closed online Dial-a-Ride on the line that matches the upper bound given in [1] for arbitrary metric spaces. Note that in this setting, by definition, every schedule of Smartstart is a closed walk that returns to the origin.

▶ **Theorem 5.1.** *The competitive ratio of* Smartstart *for closed online* Dial-a-Ride *on the line with $\Theta = 2$ is exactly 2. For every other $\Theta > 1$ with $\Theta \neq 2$ the competitive ratio of* $\textsc{Smartstart}_{\Theta}$ *is larger than 2.*

Proof. We show that the competitive ratio of $\textsc{Smartstart}_2$ is at least 2 and that the competitive ratio of $\textsc{Smartstart}_\Theta$ is larger than 2 for all $\Theta \neq 2$. From the fact that Smartstart is 2-competitive even for general metric spaces [1, Thm. 6], it follows that $\textsc{Smartstart}_2$ has competitive ratio exactly 2 on the line.

Let $\Theta \leq 2$ and consider the set of requests $\{\sigma_1\}$ with $\sigma_1 = (0.5, 0.5; 0)$. Obviously, Opt can serve this request and return to the origin in time $\textsc{Opt}(\{\sigma_1\}) = 1$. Thus, for all $t \geq 0$, we have $L(t, 0, \{\sigma_1\}) = 1$. On the other hand, Smartstart waits until time

$$t_1 = \frac{L(t_1, 0, \{\sigma_1\})}{\Theta - 1} = \frac{1}{\Theta - 1}$$

to start its only schedule and finishes at time $\frac{\Theta}{\Theta-1}$. To sum it up, we have

$$\frac{\textsc{Smartstart}(\{\sigma_1\})}{\textsc{Opt}(\{\sigma_1\})} = \frac{\Theta}{\Theta - 1}$$

with $\frac{\Theta}{\Theta-1} > 2$ for all $\Theta < 2$ and $\frac{\Theta}{\Theta-1} = 2$ for $\Theta = 2$. Now let $2 < \Theta \leq 3$ and $\varepsilon \in (0, \min\{1 - \frac{1}{\Theta-1}, \frac{\Theta-2}{2(\Theta-1)}\})$, and consider the set of requests $\{\sigma_1, \sigma_2\}$ with

$$\sigma_1 = (0.5, 0.5; 0) \qquad \text{and} \qquad \sigma_2 = \left(1 - \frac{1}{\Theta - 1} - \varepsilon, 1 - \frac{1}{\Theta - 1} - \varepsilon; \frac{1}{\Theta - 1} + \varepsilon\right).$$

By assumption, we have $\Theta > 2$ and $\varepsilon < 1 - \frac{1}{\Theta-1}$, which implies

$$0 \overset{\varepsilon<1-\frac{1}{\Theta-1}}{<} 1 - \frac{1}{\Theta - 1} - \varepsilon \overset{\Theta\leq 3}{\leq} 0.5,$$

i.e., the position of request σ_2 lies between 0 and 0.5. If Opt moves to position 0.5 and then returns to the origin, it is at position

$$a_2 = 0.5 - \Bigg|\underbrace{\left(\frac{1}{\Theta - 1} + \varepsilon\right)}_{>0.5} - 0.5\Bigg| = 1 - \frac{1}{\Theta - 1} - \varepsilon$$

at time $r_2 = \frac{1}{\Theta-1} + \varepsilon$. Thus, Opt can serve σ_2 on the way and we have $\textsc{Opt}(\{\sigma_1, \sigma_2\}) = 1$. For all $t \geq 0$, we have $L(t, 0, \{\sigma_1\}) = 1$. Therefore, Smartstart waits until time

$$t_1 = \frac{L(t_1, 0, \{\sigma_1\})}{\Theta - 1} = \frac{1}{\Theta - 1}.$$

before starting its first schedule. Since we have $\frac{1}{\Theta-1} < \frac{1}{\Theta-1} + \varepsilon$, Smartstart starts to serve σ_1 at time t_1 and returns to the origin at time $\frac{\Theta}{\Theta-1}$. For all $t \geq 0$, we have

$$L(t, 0, \{\sigma_2\}) = 2 - \frac{2}{\Theta - 1} - 2\varepsilon,$$

thus Smartstart does not start the second and final schedule before time $\frac{2-\frac{2}{\Theta-1}-2\varepsilon}{\Theta-1}$. By assumption, we have $\Theta > 2$, which implies $\frac{\Theta}{\Theta-1} > \frac{2-\frac{2}{\Theta-1}-2\varepsilon}{\Theta-1}$. Thus, the second schedule is started at time $t_2 = \frac{\Theta}{\Theta-1}$ and finished at time

$$\textsc{Smartstart}(\{\sigma_1, \sigma_2\}) = \frac{\Theta}{\Theta - 1} + 2 - \frac{2}{\Theta - 1} - 2\varepsilon.$$

To sum it up, we have

$$\begin{aligned}\frac{\textsc{Smartstart}(\{\sigma_1,\sigma_2\})}{\textsc{Opt}(\{\sigma_1,\sigma_2\})} &= \frac{\Theta}{\Theta-1}+2-\frac{2}{\Theta-1}-2\varepsilon \\ &\overset{\varepsilon<\frac{\Theta-2}{2(\Theta-1)}}{>} \frac{3\Theta-4}{\Theta-1}-2\frac{\Theta-2}{2(\Theta-1)} \\ &= 2.\end{aligned}$$

Now let $\Theta > 3$ and $\varepsilon \in (0, 0.5 - \frac{1}{\Theta-1})$, and consider the set of requests $\{\sigma_1, \sigma_2\}$ with

$$\sigma_1 = (0.5, 0.5; 0) \qquad \text{and} \qquad \sigma_2 = \left(0.5, 0.5; \frac{1}{\Theta-1}+\varepsilon\right).$$

By assumption, we have $\varepsilon < 0.5 - \frac{1}{\Theta-1}$, which implies $\frac{1}{\Theta-1} + \varepsilon < 0.5$, i.e., σ_2 is released before position 0.5 is reachable. If Opt moves to position 0.5 and then returns to the origin, it can serve both requests without additional waiting time and we have $\textsc{Opt}(\{\sigma_1, \sigma_2\}) = 1$. For all $t \geq 0$, we have $L(t, 0, \{\sigma_1\}) = 1$. Therefore, Smartstart waits until time

$$t_1 = \frac{L(t_1, 0, \{\sigma_1\})}{\Theta-1} = \frac{1}{\Theta-1}.$$

before starting its first schedule. Since we have $\frac{1}{\Theta-1} < \frac{1}{\Theta-1} + \varepsilon$, Smartstart starts to serve σ_1 at time t_1 and returns to the origin at time $\frac{\Theta}{\Theta-1}$. For all $t \geq 0$, we have

$$L(t, 0, \{\sigma_2\}) = 1,$$

thus Smartstart does not start the second and final schedule before time $\frac{1}{\Theta-1}$. By assumption, we have $\Theta > 3$, which implies $\frac{\Theta}{\Theta-1} > \frac{1}{\Theta-1}$. Thus, the second schedule is started at time $t_2 = \frac{\Theta}{\Theta-1}$ and finished at time

$$\textsc{Smartstart}(\{\sigma_1, \sigma_2\}) = \frac{\Theta}{\Theta-1} + 1.$$

To sum it up, we have

$$\frac{\textsc{Smartstart}(\{\sigma_1,\sigma_2\})}{\textsc{Opt}(\{\sigma_1,\sigma_2\})} = \frac{\Theta}{\Theta-1}+1 > 2. \qquad \blacktriangleleft$$

References

1. Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 639–650, 2000.
2. Mikhail J. Atallah and S. Rao Kosaraju. Efficient Solutions to Some Transportation Problems with Applications to Minimizing Robot Arm Travel. *SIAM Journal on Computing*, 17(5), 1988.
3. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the On-Line Travelling Salesman. *Algorithmica*, 29(4):560–581, 2001.
4. Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight Bounds for Online TSP on the Line. In *Proceedings of the 28th Annual Symposium on Discrete Algorithms (SODA)*, pages 994–1005, 2017.
5. Michiel Blom, Sven O. Krumke, Willem E. de Paepe, and Leen Stougie. The Online TSP Against Fair Adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001.

6 Moses Charikar and Balaji Raghavachari. The Finite Capacity Dial-A-Ride Problem. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 458–467, 1998.

7 Willem E. de Paepe, Jan Karel Lenstra, Jiri Sgall, René A. Sitters, and Leen Stougie. Computer-Aided Complexity Classification of Dial-a-Ride Problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004.

8 Esteban Feuerstein and Leen Stougie. On-line Single-server Dial-a-ride Problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

9 Paul C. Gilmore and Ralph E. Gomory. Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, 12(5):655–679, 1964.

10 D. J. Guan. Routing a Vehicle of Capacity Greater Than One. *Discrete Applied Mathematics*, 81(1-3):41–57, 1998.

11 Dietrich Hauptmeier, Sven Oliver Krumke, and Jörg Rambau. The Online Dial-a-Ride Problem Under Reasonable Load. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC)*, pages 125–136, 2000.

12 Patrick Jaillet and Michael R. Wagner. Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses. *Operations Research*, 56(3):745–757, 2008.

13 Sven O. Krumke. Online Optimization Competitive Analysis and Beyond, 2001. Habilitation thesis.

14 Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On Minimizing the Maximum Flow Time in the Online Dial-a-ride Problem. In *Proceedings of the Third International Conference on Approximation and Online Algorithms (WAOA)*, pages 258–269, 2006.

15 Sven O. Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness Helps: An O(1)-Competitive Algorithm for Minimizing the Maximum Flow Time in the Online Traveling Salesman Problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 200–214, 2002.

16 Maarten Lipmann. *On-Line Routing*. PhD thesis, Technical University Eindhoven, 2003.

17 Maarten Lipmann, Xiwen Lu, Willem E. de Paepe, Rene A. Sitters, and Leen Stougie. On-Line Dial-a-Ride Problems Under a Restricted Information Model. *Algorithmica*, 40(4):319–329, 2004.

18 Fanglei Yi and Lei Tian. On the Online Dial-a-ride Problem with Time-windows. In *Proceedings of the 1st International Conference on Algorithmic Applications in Management (AAIM)*, pages 85–94, 2005.

19 Fanglei Yi, Yinfeng Xu, and Chunlin Xin. Online Dial-a-ride Problem with Time-windows Under a Restricted Information Model. In *Proceedings of the 2nd International Conference on Algorithmic Aspects in Information and Management (AAIM)*, pages 22–31, 2006.

Enumerating Minimal Dominating Sets in Triangle-Free Graphs

Marthe Bonamy
CNRS, Université de Bordeaux, France
marthe.bonamy@u-bordeaux.fr

Oscar Defrain
LIMOS, Université Clermont Auvergne, France
oscar.defrain@uca.fr

Marc Heinrich
LIRIS, Université Claude-Bernard, Lyon, France
marc.heinrich@univ-lyon1.fr

Jean-Florent Raymond
LaS team, Technische Universität Berlin, Germany
raymond@tu-berlin.de

Abstract

It is a long-standing open problem whether the minimal dominating sets of a graph can be enumerated in output-polynomial time. In this paper we prove that this is the case in triangle-free graphs. This answers a question of Kanté et al. Additionally, we show that deciding if a set of vertices of a bipartite graph can be completed into a minimal dominating set is a NP-complete problem.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases Enumeration algorithms, output-polynomial algorithms, minimal dominating set, triangle-free graphs, split graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.16

Related Version `https://arxiv.org/abs/1810.00789`

Funding *Oscar Defrain*: Supported by ANR project GraphEn ANR-15-CE40-0009.
Jean-Florent Raymond: Supported by ERC consolidator grant DISTRUCT-648527.

Acknowledgements The authors wish to thank Paul Ouvrard for extensive discussions on the topic of this paper. We also gratefully acknowledge support from Nicolas Bonichon and the Simon family for the organization of the 3rd Pessac Graph Workshop, where this research was done. Last but not least, we thank Peppie for her unwavering support during the work sessions.

1 Introduction

Countless algorithmic problems in graph theory require to detect a structure with prescribed properties in an input graph. Rather than finding one such object, it is sometimes more desirable to generate all of them. This is for instance useful in certain applications to database search [29], network analysis [13], bioinformatics [22, 5], and cheminformatics [2]. Enumeration algorithms for graph problems seem to have been first mentioned in the early 70's with the pioneer works of Tiernen [27] and Tarjan [26] on cycles in directed graphs and of Akkoyunlu [1]. However, they already appeared in disguise in earlier works [24, 21]. To this date, several intriguing questions on the topic remain unsolved. We refer the reader to [23] for a more in-depth introduction to enumeration algorithms and to [28] for a listing of enumeration algorithms and problems.

The objects we wish to enumerate in this paper are the (inclusion-wise) minimal dominating sets of a given graph. In general, the number of these objects may grow exponentially with the order n of the input graph. Therefore, in stark contrast to decision or optimization problems, looking for a running time polynomially bounded by n is not a reasonable, let alone meaningful, efficiency criterion. Rather, we aim here for algorithms whose running time is polynomially bounded by the size of both the input and output data, called *output-polynomial* algorithms.

Because dominating sets are among the most studied objects in graph theory and algorithms, their enumeration (and counting) have attracted an increasing attention over the past 10 years. The problem of enumerating minimal dominating sets (hereafter referred to as Dom-Enum) has a notable feature: it is equivalent to the extensively studied hypergraph problem Trans-Enum. In Trans-Enum, one is given a hypergraph $\mathcal{H}$ (i.e. a collection of sets, called *hyperedges*) and is asked to enumerate all the minimal *transversals* of $\mathcal{H}$ (i.e. the inclusion-minimal sets of elements that meet every hyperedge). It is not hard to see that Dom-Enum is a particular case of Trans-Enum: the minimal dominating sets of a graph G are exactly the minimal transversals of the hypergraph of closed neighborhoods of G. Conversely, Kanté, Limouzy, Mary, and Nourine proved that every instance of Trans-Enum can be reduced to a co-bipartite[1] instance of Dom-Enum [17]. Currently, the best output-sensitive algorithm for Trans-Enum is due to Fredman and Khachiyan and runs in quasi-polynomial time [9]. It is a long-standing open problem whether this complexity bound can be improved (see for instance the surveys [6, 8]). Therefore, the equivalence between the two problems is an additional motivation to study Dom-Enum, with the hope that techniques from graph theory will be used to obtain new results on the Trans-Enum problem. So far, output-polynomial algorithms have been obtained for Dom-Enum in several classes of graphs, including planar graphs and degenerate graphs [7], classes of graphs of bounded tree-width, clique-width [4], or mim-width [10], path graphs and line graphs [16], interval graphs and permutation graphs [18], split graphs [19], graphs of girth at least 7 [12], chordal graphs [19], and chordal bipartite graphs [11]. A succinct survey of results on Dom-Enum can be found in [20]. The authors of [19] state as an open problem the question to design an output-polynomial algorithm for bipartite graphs (the problem also appeared in [20, 11]). We address this problem with the following result.

▶ **Theorem 1.** *There is an output-polynomial time algorithm enumerating minimal dominating sets in triangle-free graphs.*

In particular, the result holds for enumerating minimal dominating sets in bipartite graphs.

Our algorithm decomposes the graph by iteratively removing closed neighborhoods in the fashion of [7], then constructs partial minimal dominating sets by adding the neighborhoods back one after the other. It relies on the crucial property that in triangle-free graphs, the generation of all potential extensions of a partial minimal dominating set to a new neighborhood is closely related to the enumeration of minimal dominating sets in split graphs, for which tools have already been developed [17]. We note that triangle-free graphs already received attention in the context of enumeration of other objects, for instance maximal independent sets [14, 3], using different techniques.

A natural technique to enumerate valid solutions to a given problem (for instance, sets of vertices satisfying a given property) is to build them element by element. If during the construction one detects that the current partial solution cannot be extended into a valid

[1] The complement of a bipartite graph.

one, then it can be discarded along with all the other partial solutions that contain it. Note that in order to apply this technique, one should be able to decide whether a given partial solution can be completed into a valid one. It turns out that for minimal dominating sets, this problem (that we will denote by Dcs) is NP-complete [15], even when restricted to split graphs [19]. We show that it remains NP-complete in bipartite graphs.

▶ **Theorem 2.** Dcs *restricted to bipartite graphs is NP-complete.*

This implies that Dcs is NP-complete in triangle-free graphs. This suggests that the aforementioned technique is unlikely to be used to improve Theorem 1.

The paper is organized as follows. In Section 2 we give the necessary definitions. We prove Theorems 1 and 2 in Sections 3 and 4, respectively. We conclude with possible future research directions in Section 5.

2 Preliminaries

Graphs. All graphs in this paper are finite, undirected, simple, and loopless. If G is a graph, then $V(G)$ is its set of vertices and $E(G) \subseteq V(G)^2$ is its set of edges. Edges are denoted by xy (or yx) instead of $\{x, y\}$. We assume that vertices are assigned distinct indices; these will be used to choose vertices in a deterministic way, typically selecting the vertex of smallest index. A *clique* (respectively an *independent set*) in a graph G is a set of pairwise adjacent (respectively non-adjacent) vertices. The subgraph of G *induced* by $X \subseteq V(G)$, denoted by $G[X]$, is the graph $(X, E(G) \cap (X \times X))$; $G \setminus X$ is the graph $G[V(G) \setminus X]$.

If the vertex set of a graph G can be partitioned into one part inducing a clique and one part inducing an independent set (respectively two independent sets, two cliques), we say that G is a *split* (respectively *bipartite*, *co-bipartite*) graph. Graphs where every cycle is of length at least 4 are referred to as *triangle-free* graphs. If f is a function, we write $f(n) = \operatorname{poly} n$ when there is a constant $c \in \mathbb{N}$ such that $f(n) = O(n^c)$.

Neighbors and domination. Let G be a graph and $x \in V(G)$. We note $N(x)$ the set of *neighbors* of x in G defined by $N(x) = \{y \in V(G) \mid xy \in E(G)\}$; $N[x]$ is the set of *closed neighbors* defined by $N[x] = N(x) \cup \{x\}$. For a given $X \subseteq V(G)$, we respectively denote by $N[X]$ and $N(X)$ the sets defined by $\bigcup_{x \in X} N[x]$ and $N[X] \setminus X$. Let D be a set of vertices of G. We say that D is *dominating* a subset $S \subseteq V(G)$ if $S \subseteq N[D]$. It is *minimally dominating* S if no proper subset of D dominates S. The set D is a (*minimal*) *dominating set* of G if it (minimally) dominates $V(G)$. The set of all minimal dominating sets of G is denoted by $\mathcal{D}(G)$ and the problem of enumerating $\mathcal{D}(G)$ given G is denoted by Dom-Enum. Let $S \subseteq V(G)$. A vertex $y \in V(G)$ is said to be a *private neighbor* of some $x \in S$ if $y \notin N[S \setminus \{x\}]$. Intuitively, this means that y is not dominated by any other vertex of S. Note that x can be its own private neighbor. The set of private neighbors of $x \in S$ in G is denoted by $\operatorname{Priv}_G(S, x)$ and we drop the subscript when it can be inferred from the context. Observe that S is a minimal dominating set of G if and only if $V(G) \subseteq N[S]$ and for every $x \in S$, $\operatorname{Priv}(S, x) \neq \emptyset$.

Enumeration. The aim of graph enumeration algorithms is to generate a set of objects $\mathcal{X}(G)$ related to a graph G. We say that an algorithm enumerating $\mathcal{X}(G)$ with input an n-vertex graph G is *output-polynomial* if its running time is polynomially bounded by the size of the input and output data, i.e. $n + |\mathcal{X}(G)|$. If an algorithm enumerates $\mathcal{X}(G)$ by spending $\operatorname{poly}(n)$-time (respectively $O(n)$-time) before it outputs the first element, between two output elements, and after it outputs the last element, then we say that it runs with *polynomial delay*

(respectively *linear delay*). It is easy to see that every polynomial delay algorithm is also output-polynomial. Note however that some problems have output-polynomial algorithms but no polynomial delay ones, unless P=NP [25]. When discussing the space used by an enumeration algorithm, we ignore the space where the solutions are output.

3 Minimal domination in triangle-free graphs

In this section, we give an output-polynomial time algorithm to enumerate minimal dominating sets in triangle-free graphs. The algorithm is inspired by the one of [7] and constructs dominating sets one neighborhood at a time.

A *peeling* of a graph G is a sequence $(V_0, \dots, V_p)$ such that $V_p = V(G)$, $V_0 = \emptyset$, and for every $i \in \{1, \dots, p\}$,

$$V_{i-1} = V_i \setminus N[v_i]$$

for some $v_i \in V_i$. We call $(v_1, \dots, v_p)$ the *vertex sequence* of the peeling; note that p is only known after peeling the whole graph.

In the following, we consider a triangle-free graph G and a fixed peeling $(V_0, \dots, V_p)$ with vertex sequence $(v_1, \dots, v_p)$. For every $i \in \{0, \dots, p\}$, we denote by $\mathcal{D}(G, i)$ the set of minimal dominating sets of V_i in G. Recall that these sets may contain vertices of $G - V_i$, which is a crucial point. Then $\mathcal{D}(G, p) = \mathcal{D}(G)$.

▶ **Definition 3.** *Let $i \in \{0, \dots, p-1\}$ and $D \in \mathcal{D}(G, i+1)$. We denote by* $\mathsf{Parent}(D, i+1)$ *the pair (D^*, i) where D^* is obtained from D by successively removing the vertex x of smaller index in D satisfying* $\mathrm{Priv}(D, x) \cap V_i = \emptyset$*, until no such vertex exists.*

Clearly, there is a unique way to build $\mathsf{Parent}(D, i+1)$ given D and i. By construction, the obtained set D^* is a minimal dominating set of V_i. Hence every set in $D(G, i+1)$ can be obtained by completing some D^* in $D(G, i)$; we develop this point below.

▶ **Proposition 4.** *Let $i \in \{0, \dots, p-1\}$ and $D^* \in \mathcal{D}(G, i)$.*
If D^* dominates V_{i+1} then $D^* \in \mathcal{D}(G, i+1)$ and $\mathsf{Parent}(D^*, i+1) = (D^*, i)$.
Otherwise, $D^* \cup \{v_{i+1}\} \in \mathcal{D}(G, i+1)$ and $\mathsf{Parent}(D^* \cup \{v_{i+1}\}, i+1) = (D^*, i)$.

Proof. First note that since $D^* \in \mathcal{D}(G, i)$, $\mathrm{Priv}(D^*, x) \cap V_i \neq \emptyset$ for all $x \in D^*$. Hence $\mathsf{Parent}(D^*, i+1) = (D^*, i)$ whenever D^* dominates V_{i+1}. If D^* does not dominate V_{i+1} then $D = D^* \cup \{v_{i+1}\}$ does. Moreover, $\mathrm{Priv}(D, v_{i+1}) \cap V_{i+1} \neq \emptyset$. Since v_{i+1} is not connected to any vertex in V_i, it cannot steal any private neighbors to the elements of D^*. Hence $\mathrm{Priv}(D, x) \cap V_{i+1} \neq \emptyset$ for all $x \in D$. Now, remark that since v_{i+1} does not steal private neighbors to the elements of D^*, it is indeed itself the only node with no privates in V_i and is removed by the parent function. Hence $\mathsf{Parent}(D^* \cup \{v_{i+1}\}, i+1) = (D^*, i)$. ◀

The Parent relation as introduced in Definition 3 defines a tree on vertex set

$$\{(D, i) \mid i \in \{1, \dots, p\},\ D \in \mathcal{D}(G, i)\},$$

with leaves $\{(D, p) \mid D \in \mathcal{D}(G)\}$, and root $(\emptyset, 0)$ (the empty set being the only dominating set of the empty vertex set V_0). Our algorithm will search this tree in order to enumerate every minimal dominating set of G. Proposition 4 guarantees that for every $i < p$ and every $D^* \in \mathcal{D}(G, i)$, the pair (D^*, i) is the parent of some $(D, i+1)$ with $D \in \mathcal{D}(G, i+1)$ (possibly $D = D^*$). Consequently, every branch of the tree leads to a different minimal dominating set of G. In particular, for every $i < p$, we have $|\mathcal{D}(G, i)| \leq |\mathcal{D}(G, i+1)|$.

Given a set $D^* \in \mathcal{D}(G,i)$, we now focus on the enumeration of every $D \in \mathcal{D}(G,i+1)$ such that $(D,i+1)$ has (D^*,i) for parent. From Proposition 4, we know that either $(D^*,i+1)$ or $(D^* \cup \{v_{i+1}\},i+1)$ has (D^*,i) for parent. Consequently, we refer to $X=\emptyset$ and $X=\{v_{i+1}\}$ as the *trivial extensions* of (D^*,i), and focus on the non-trivial ones.

We call *candidate extension* of (D^*,i) any (inclusion-wise) minimal set $X \subseteq V(G)$ such that $D^* \cup X$ dominates V_{i+1} in G, avoiding the trivial cases where $X \in \{\emptyset,\{v_{i+1}\}\}$. Then, X is a candidate extension of (D^*,i) if and only if $X \notin \{\emptyset,\{v_{i+1}\}\}$, $V_{i+1} \subseteq N[D^* \cup X]$ and, for every $x \in X$, $\mathrm{Priv}(D^* \cup X, x) \cap V_{i+1} \neq \emptyset$. Note that possibly not all candidate extensions of (D^*,i) form with D^* a minimal dominating set of V_{i+1}. In fact, there is no guarantee that any candidate extension forms a minimal dominating set of V_{i+1}: it might be that (D^*,i) has a unique child, given by its trivial extension. We denote by $\mathcal{C}(D^*,i)$ the set of candidate extensions of (D^*,i). We point out that by the minimality assumption, the vertex v_{i+1} appears in no element of $\mathcal{C}(D^*,i)$, as v_{i+1} itself is a trivial extension.

▶ **Lemma 5.** *Let $i \in \{0,\dots,p-1\}$ and $D^* \in \mathcal{D}(G,i)$. Then $|\mathcal{C}(D^*,i)| \leq |\mathcal{D}(G)|$.*

Proof. We argue that for every $X \in \mathcal{C}(D^*,i)$ there is an element of $\mathcal{D}(G,i+1)$ whose intersection with $V(G) \setminus D^*$ is precisely X. This will prove $|\mathcal{C}(D^*,i)| \leq |\mathcal{D}(G,i+1)|$, hence $|\mathcal{C}(D^*,i)| \leq |\mathcal{D}(G)|$ as desired.

Let $X \in \mathcal{C}(D^*,i)$. We consider the set $X \cup D^*$, which dominates V_{i+1}. By definition of $\mathcal{C}(D^*,i)$, we have $\mathrm{Priv}(X \cup D^*, x) \cap V_{i+1} \neq \emptyset$ for every $x \in X$. Therefore, every subset of $X \cup D^*$ that dominates V_{i+1} contains X. Consider an inclusion-wise minimal subset D' of $X \cup D^*$ that dominates V_{i+1}. We have $X \subseteq D'$, hence the conclusion. ◀

Lemma 5 above ensures that $\mathcal{C}(D^*,i)$ is bounded by $\mathcal{D}(G)$. Hence, it is reasonable to test each of the candidate extensions even though D^* might be the parent of only one set in $\mathcal{D}(G,i+1)$. It now suffices to explain how to efficiently enumerate $\mathcal{C}(D^*,i)$ to complete the algorithm (formally described in Theorem 12).

Let $i \in \{0,\dots,p-1\}$ and $D^* \in \mathcal{D}(G,i)$. We define $S = N(v_{i+1}) \cap V_{i+1} \setminus N[D^*]$ and $C = N(S) \setminus \{v_{i+1}\}$. As G is triangle-free and S is included in the neighborhood of v_{i+1}, S is an independent set. Let $Z^i_{D^*}$ be the split graph obtained from $G[C \cup S]$ where C is completed into a clique; note that the independent set S is maximal in $Z^i_{D^*}$ since $C \subseteq N(S)$. For any $X \subseteq V(Z^i_{D^*})$, we define $X_C = X \cap C$ and $X_S = X \cap S$. We set $\mathcal{D}_{S=\emptyset}(Z^i_{D^*}) = \{D \in \mathcal{D}(Z^i_{D^*}) \mid D_S = \emptyset\}$. The following result is implicit in [17].

▶ **Proposition 6.** *Let H be a split graph with maximal stable set S and clique C. Let $X \subseteq V(H)$. Then, $X \in \mathcal{D}(H)$ if and only if $S \subseteq N[X]$ and $\mathrm{Priv}(X,x) \cap S \neq \emptyset$ for all $x \in X$.*

Proof. Let us assume that $S \subseteq N[X]$ and $\mathrm{Priv}(X,x) \cap S \neq \emptyset$ for all $x \in X$. Then, either $X \cap C \neq \emptyset$ or $X \cap C = \emptyset$. In the first case, X dominates C. In the other case, $X = S$ because $S \subseteq N[X]$ and $V(H) = C \cup S$. Remark that $C \subseteq N(S)$ as S is assumed maximal. Hence, X also dominates C. The minimality of X follows from our first assumption. Hence $X \in \mathcal{D}(H)$.

Conversely, let $X \in \mathcal{D}(H)$. Clearly $N[X] \supseteq S$, so we suppose by contradiction that $\mathrm{Priv}(X,x) \cap S = \emptyset$ for some $x \in X$. By minimality of X, we have $\mathrm{Priv}(X,x) \neq \emptyset$, which implies $\mathrm{Priv}(X,x) \subseteq C$. Consequently, we must have $X \cap C = \{x\}$, or else $x \in S$ but is not its own private, in which case it must have a neighbor in C which contradicts $\mathrm{Priv}(X,x) \neq \emptyset$. As $C \subseteq N(S)$, there exists some vertex $y \in S \cap N(x)$. Since $y \notin \mathrm{Priv}(X,x)$ and $X \cap C = \{x\}$, we have $y \in X$. However, in this case $N[y] \subseteq N[x]$ and so $\mathrm{Priv}(X,y) = \emptyset$, which contradicts the minimality of X. ◀

We now characterize $\mathcal{C}(D^*, i)$ depending on whether v_{i+1} has to be dominated by the extension or not. The condition $D^* \in \mathcal{D}(G, i) \setminus \mathcal{D}(G, i+1)$ in the statement below prevents (D^*, i) from having the trivial extension $\emptyset$ –in which case it is the only one.

▶ **Lemma 7.** *Let $i \in \{0, \dots, p-1\}$, $D^* \in \mathcal{D}(G, i) \setminus \mathcal{D}(G, i+1)$ and $Z = Z^i_{D^*}$. Then*

- *either $D^* \cap N(v_{i+1}) \neq \emptyset$ and $\mathcal{C}(D^*, i) = \mathcal{D}(Z)$,*
- *or $D^* \cap N(v_{i+1}) = \emptyset$ and*

$$\mathcal{C}(D^*, i) = (\mathcal{D}(Z) \setminus \mathcal{D}_{S=\emptyset}(Z)) \cup \left\{ Q \cup \{u\} \;\middle|\; \begin{array}{l} Q \in \mathcal{D}_{S=\emptyset}(Z), \\ u \in N(v_{i+1}), \text{ and} \\ \forall x \in Q, \ \mathrm{Priv}(Q \cup \{u\}, x) \cap V_{i+1} \neq \emptyset \end{array} \right\}.$$

Proof. Let us first consider the case $D^* \cap N(v_{i+1}) \neq \emptyset$. Let $X \in \mathcal{C}(D^*, i)$. Since v_{i+1} is dominated by any vertex of $D^* \cap N(v_{i+1})$, only the stable set S of Z is to be dominated by X. In other words X minimally dominates S: $S \subseteq N[X]$ and $\mathrm{Priv}(X, x) \cap S \neq \emptyset$ for all $x \in X$. By Proposition 6, $X \in \mathcal{D}(Z)$, which proves the inclusion $\mathcal{C}(D^*, i) \subseteq \mathcal{D}(Z)$. Conversely, let $X \in \mathcal{D}(Z)$. By Proposition 6, $S \subseteq N[X]$ and $\mathrm{Priv}(X, x) \cap S \neq \emptyset$ for all $x \in X$. Since v_{i+1} is already dominated by D^*, $X \in \mathcal{C}(D^*, i)$. Hence $\mathcal{C}(D^*, i) = \mathcal{D}(Z)$, as desired.

From now on and until the end of the proof we assume that $D^* \cap N(v_{i+1}) = \emptyset$. Let C denote the vertex set of the clique of Z. Let $X \in \mathcal{C}(D^*, i)$. We know that X must be a dominating set of Z. Indeed, by definition of $\mathcal{C}(D^*, i)$, X dominates S, and either $X \cap C \neq \emptyset$, in which case X also dominates C, or $X = S$ and X also dominates C since $C \subseteq N(S)$. There are two cases to consider.

If X is a minimal dominating set of Z, then since X has to dominate v_{i+1}, we have $X \cap S \neq \emptyset$ and consequently $X \in \mathcal{D}(Z) \setminus \mathcal{D}_{S=\emptyset}(Z)$.

Otherwise, X is not a *minimal* dominating set of Z. This implies that it has a vertex u with no private neighbor in Z. By definition of $\mathcal{C}(D^*, i)$, this means that $\mathrm{Priv}(D^* \cup X, u) \cap V_{i+1} = \{v_{i+1}\}$. Therefore there is exactly one such vertex. Then, if we write $Q = X \setminus \{u\}$, Q is a minimal dominating set of Z. Since v_{i+1} is a private neighbor of u, we must have $Q \cap S = \emptyset$, and consequently $Q \in \mathcal{D}_{S=\emptyset}(Z)$. Finally, by definition of $C(D^*, i)$, for any $x \in Q \subset X$, we have $\mathrm{Priv}(X, x) \cap V_{i+1} \neq \emptyset$. This shows that we have

$$X \in \left\{ Q \cup \{u\} \;\middle|\; \begin{array}{l} Q \in \mathcal{D}_{S=\emptyset}(Z), \\ u \in N(v_{i+1}), \text{ and} \\ \forall x \in Q, \ \mathrm{Priv}(Q \cup \{u\}, x) \cap V_{i+1} \neq \emptyset \end{array} \right\}, \tag{1}$$

and proves the following inclusion:

$$\mathcal{C}(D^*, i) \subseteq (\mathcal{D}(Z) \setminus \mathcal{D}_{S=\emptyset}(Z)) \cup \left\{ Q \cup \{u\} \;\middle|\; \begin{array}{l} Q \in \mathcal{D}_{S=\emptyset}(Z), \\ u \in N(v_{i+1}), \text{ and} \\ \forall x \in Q, \ \mathrm{Priv}(Q \cup \{u\}, x) \cap V_{i+1} \neq \emptyset \end{array} \right\}.$$

To prove the reverse inclusion, we first consider $X \in \mathcal{D}(Z) \setminus \mathcal{D}_{S=\emptyset}(Z)$. By Proposition 6, $S \subseteq N[X]$ and $\mathrm{Priv}(X, x) \cap S \neq \emptyset$ for all $x \in X$. Since $X \cap S \neq \emptyset$, $S \cup \{v_{i+1}\} \subseteq N[X]$. Thus $X \in \mathcal{C}(D^*, i)$. Now we consider a set X of the form $Q \cup \{u\}$, for some $Q \in \mathcal{D}_{S=\emptyset}(Z)$ and $u \in N(v_{i+1})$ such that $\forall x \in Q$, $\mathrm{Priv}(Q \cup \{u\}, x) \cap V_{i+1} \neq \emptyset$. By Proposition 6, $\mathrm{Priv}(Q, x) \cap S \neq \emptyset$ for all $x \in Q$. Since $\mathrm{Priv}(Q \cup \{u\}, x) \cap V_{i+1} \neq \emptyset$ for all $x \in Q$ and $v_{i+1} \in \mathrm{Priv}(X, u)$, $\mathrm{Priv}(X, x) \cap V_{i+1} \neq \emptyset$ for all $x \in X$. Since $S \cup \{v_{i+1}\} \subseteq N[X]$, $X \in \mathcal{C}(D^*, i)$. This proves the reverse inclusion and concludes the proof. ◀

In [17], authors give a polynomial delay algorithm to enumerate minimal dominating sets in split graphs.

▶ **Theorem 8** ([17]). *There is an algorithm that, given a split graph H with n vertices and m edges, outputs with $O(n+m)$ delay every minimal dominating set of H, using $O(n^2)$ space.*

The above algorithm relies on the observation that for every split graph H, the set $\mathcal{D}_C(H) = \{D_C \mid D \in \mathcal{D}(H)\}$ is in bijection with $\mathcal{D}(H)$ and it forms an independence system. A family of sets $\mathcal{S}$ is an *independence system* if $S \in \mathcal{S}$ implies that $S \setminus \{s\} \in \mathcal{S}$ for all $s \in S$. We show that there is a polynomial delay algorithm to enumerate $\mathcal{C}(D^*, i)$ given $i \in \{1, \dots, p-1\}$ and $D^* \in \mathcal{D}(G, i)$ using the same observations.

▶ **Proposition 9** ([17]). *Let H be a split graph with maximal stable set S and clique C and let D be a minimal dominating set of H. Then $D_S = S \setminus N(D_C)$.*

▶ **Proposition 10** ([17]). *Let H be a split graph with maximal stable S and clique C. Then:*

1. $\mathcal{D}_C(H) = \{A \subseteq C \mid \forall x \in A,\ \mathrm{Priv}(A, x) \neq \emptyset\}$,
2. *$\mathcal{D}_C(H)$ and $\mathcal{D}(H)$ are in bijection,*
3. *$\mathcal{D}_C(H)$ is an independence system.*

▶ **Lemma 11.** *There is an algorithm that, given $i \in \{0, \dots, p-1\}$ and $D^* \in \mathcal{D}(G, i)$, enumerates $\mathcal{C}(D^*, i)$ in output-polynomial time $O(\mathrm{poly}(n) \cdot |\mathcal{C}(D^*, i)|)$ and using at most $\mathrm{poly}(|V(G)|)$ space.*

Proof. Lemma 7 allows us to consider two cases depending on whether v_{i+1} has a neighbor in D^* or not. Let $Z = Z^i_{D^*}$. As usual we denote by S and C the maximal stable set and the clique of Z, respectively.

If $D^* \cap N(v_{i+1}) \neq \emptyset$, then by Lemma 7 $\mathcal{C}(D^*, i) = \mathcal{D}(Z)$, and we can enumerate the elements of $\mathcal{C}(D^*, i)$ with polynomial delay using the algorithm of Theorem 8 on $\mathcal{D}(Z)$.

In the case where $D^* \cap N(v_{i+1}) = \emptyset$, we start enumerating $\mathcal{D}_C(Z)$. This can be done with polynomial delay and space as in the proof of Theorem 8, using the fact that $\mathcal{D}_C(Z)$ is an independence system and that testing if an arbitrary set A belongs to $\mathcal{D}_C(Z)$ can be done in polynomial time using Lemma 10. That is, we construct elements of $\mathcal{D}_C(Z)$ from the empty set to every inclusion-wise maximal $A \in \mathcal{D}_C(Z)$. Repetitions are avoided using a linear ordering on vertices of C; see [17] for details. Then, for every set $A \in \mathcal{D}_C(Z)$ output by the above algorithm, we check in polynomial time if it dominates Z. If it does not, then we extend A into its unique corresponding minimal dominating set $D \in \mathcal{D}(Z)$ such that $D \cap C = A$ (i.e. $D = A \cup S \setminus N(A)$), and output D. Otherwise, for every $u \in N(v_{i+1})$ such that for all $x \in A$, $\mathrm{Priv}(A \cup \{u\}, x) \cap V_{i+1} \neq \emptyset$ (which can be tested in time polynomial in the order of Z), we output $A \cup \{u\}$. Lemma 7 guarantees that the above algorithm indeed outputs $\mathcal{C}(D^*, i)$.

Note that the only elements $D \in \mathcal{D}(Z)$ which do not lead to an element of $\mathcal{C}(D^*, i)$ are the $D \in \mathcal{D}_{S=\emptyset}(Z)$ for which no vertex $u \in N(v_{i+1})$ satisfies the desired conditions. However, we will show that $|\mathcal{D}_{S=\emptyset}(Z)| \leq n|\mathcal{D}(Z) \setminus \mathcal{D}_{S=\emptyset}(Z)|$. Indeed, consider the map f that, given $D \in \mathcal{D}_{S=\emptyset}(Z)$ removes one arbitrary vertex from D, and completes the dominating set by adding the vertices in the independent set which are no longer dominated. Then, f maps elements of $\mathcal{D}_{S=\emptyset}(Z)$, to the set $\mathcal{D}(Z) \setminus \mathcal{D}_{S=\emptyset}(Z)$. Moreover, every element in this second set is the image of at most $|C| \leq n$ elements by f. This implies the desired bound.

Consequently, this means that while enumerating $\mathcal{D}(Z)$, we might throw out a fraction at most $\frac{n}{n+1}$ of all the solutions we found which do not lead to elements in $\mathcal{C}(D^*, i)$. This shows that the algorithm has output-polynomial time. ◀

We are now ready to prove Theorem 1, that we restate here in a more accurate form.

▶ **Theorem 12.** *There is an algorithm that, given a triangle-free graph G on n vertices, outputs $\mathcal{D}(G)$ in total time* $\mathrm{poly}(n) \cdot |\mathcal{D}(G)|^2$ *and using at most* $\mathrm{poly}\, n$ *space.*

Proof. We first arbitrarily choose a peeling $(V_0, \ldots, V_p)$ of our input graph G with vertex sequence $(v_1, \ldots, v_p)$. This takes time $\mathrm{poly}\, n$.

Recall that the Parent relation defines a tree T on vertex set

$$\{(D, i) \mid i \in \{1, \ldots, p\},\ D \in \mathcal{D}(G, i)\},$$

with leaves $\{(D, p) \mid D \in \mathcal{D}(G)\}$ and root $(\emptyset, 0)$. Let us describe how to enumerate the children in T of (D^*, i) for every given vertex $D^* \in \mathcal{D}(G, i)$. If D^* dominates V_{i+1}, then $(D^*, i+1)$ is the only pair whose parent is (D^*, i). Otherwise, we proceed as follows:

1. output the trivial child $D^* \cup \{v_{i+1}\}$;
2. start (or resume, if it had already been started) the algorithm of Lemma 11 and pause it after one element X of $C(D^*, i)$ has been output;
3. if $D^* \cup X$ is not a minimal dominating set of V_{i+1} in G, or if it is but $\mathsf{Parent}(D^* \cup X, i+1) \neq (D^*, i)$, discard X and loop to (2);
4. output $D^* \cup X$ and loop to (2).

The algorithm terminates when the algorithm of Lemma 11 in step (2) completes the enumeration of $C(D^*, i)$. The correctness of the algorithm is a consequence of the following inclusions:

$$\begin{aligned}\{D \in \mathcal{D}(G, i+1) \mid \mathsf{Parent}(D, i+1) = (D^*, i)\} \subseteq &\{D \in \mathcal{D}(G, i+1) \mid D^* \subseteq D\} \\ \subseteq &\{D^* \cup X \mid X \in \mathcal{C}(D^*, i)\} \\ &\cup \{D^* \cup \{v_{i+1}\}\} \\ &\cup \{D^*\}\end{aligned}$$

Notice that it uses at most $\mathrm{poly}\, n$ space, since we only store the data of the algorithm of Lemma 11, of size at most $\mathrm{poly}\, n$, and the data to perform step (3), which is clearly also polynomial in n.

In order to enumerate $\mathcal{D}(G)$, i.e. the set of leaves of T, we perform a DFS and output each visited leaf. For each vertex of T, enumerating its children can be done in at most $\mathrm{poly}(n) \cdot |\mathcal{D}(G)|$ steps with the above algorithm, according to Lemmas 5 and 11. Besides, the number of vertices of T at distance i from the root is at most its number of leaves, hence T has at most $O(n \cdot |\mathcal{D}(G)|)$ vertices. Therefore we can enumerate $\mathcal{D}(G)$ in $\mathrm{poly}(n) \cdot |\mathcal{D}(G)|^2$ steps. Regarding the space, we observe that whenever we visit a vertex, we do not need to compute the whole set of its children. Instead, it is enough in order to continue the DFS to compute the next unvisited child only, which can be done using the algorithm above (and pausing it afterward). Therefore, when we visit some $(D, i) \in V(T)$, we only need to store the data of the $i - 1$ (paused) algorithms enumerating the children of the ancestors of (D, i) and the data of the algorithm enumerating the children of D, i.e. $i \cdot \mathrm{poly}\, n$ space. Therefore the described algorithm uses polynomial space, as claimed. ◀

4 The extension problem is hard in bipartite graphs

We recall that Dcs denotes the problem of deciding, given a graph G and a set $A \subseteq V(G)$, whether there exists a minimal dominating set D of G such that $A \subseteq D$. This problem is known to be NP-complete for general graphs [15]. It has later been proved that the variant where we search for a minimal dominating set containing A, and avoiding a given vertex

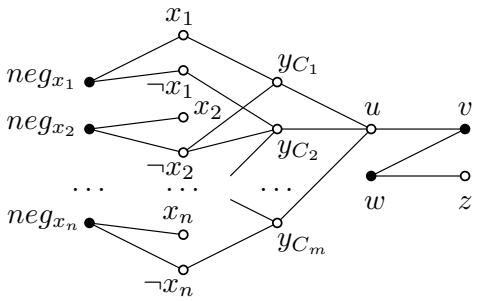


Figure 1 A bipartite graph G and a set $A \subseteq V(G)$ constructed from an instance of SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. Black vertices constitute the set A. Then A can be extended into a minimal dominating set D of G if and only if there is a truth assignment of the variable satisfying all the clauses.

set B remains intractable even on split graphs [19]. We show that Dcs is still hard for bipartite graphs and thus triangle-free graphs. As a consequence, one cannot expect to improve Theorem 1 by testing if subsets of $V(G)$ can be extended into minimal dominating sets of G. The following is a restatement of Theorem 2.

▶ **Theorem 13.** Dcs *restricted to bipartite graphs is NP-complete.*

Proof. Since Dcs is NP-complete in the general case, it is clear that Dcs is in NP even when restricted to bipartite graphs. Let us now present a reduction from SAT.

Given an instance $\mathcal{I}$ of SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we construct a bipartite graph G and a set $A \subseteq V(G)$ such that there exists a minimal dominating set containing A if and only if there exists a truth assignment that satisfies all the clauses. The graph G has vertex partition (X, Y), defined as follows.

The first part X contains two special vertices u and w, and for every variable x_i, one vertex for each of the literals x_i and $\neg x_i$. The second part Y contains one vertex y_{C_j} per clause C_j, one vertex neg_{x_i} per variable x_i, and two special vertices v and z. For every $i \in \{1, \ldots, n\}$ we make neg_{x_i} adjacent to the two literals x_i and $\neg x_i$ and for every $j \in \{1, \ldots, m\}$ we make y_{C_j} adjacent to u and to every literal C_j contains. Finally, we add edges to form the path $uvwz$ and set $A = \{neg_{x_1}, \ldots, neg_{x_n}, v, w\}$. Clearly this graph can be constructed in polynomial time from $\mathcal{I}$. The construction is illustrated in Figure 1.

Let us show that A can be extended into a minimal dominating set of G if and only if $\mathcal{I}$ has a truth assignment that satisfies all the clauses. The proof is split into two claims. A *partial assignment* of $\mathcal{I}$ is a truth assignment of a subset of the variables $x_1, \ldots, x_n$. Observe that a partial assignment may satisfy all the clauses (i.e. the values of the non-assigned variables do not matter). A partial assignment that satisfies all the clauses is called a *minimal assignment* if no proper subset of the assigned variables admits such a partial assignment.

▷ Claim 14. Let $S \subseteq \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ be a set containing at most one literal for each variable. Then S minimally dominates $\{y_{C_1}, \ldots, y_{C_m}\}$ if and only if its elements form a minimal assignment of $\mathcal{I}$.

Proof of Claim 14. Let S be as above and let $j \in \{1, \ldots, m\}$. Since $y_{C_j} \notin S$, the set S contains a neighbor x of y_{C_j}. By construction, x is a literal appearing in C_j. Hence a partial assignment of the variables of $\mathcal{I}$ satisfying all its clauses is given by the literals present in S. Moreover, x has a private neighbor $y_{C_{j'}}$, by minimality of S. The assignment given by S is hence minimal: not specifying the value of the variable of x would leave the clause $C_{j'}$ unsatisfied. ◁

▷ Claim 15. If D is a minimal dominating set of G containing A, then $D \setminus A \subseteq \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ and it contains at most one literal for each variable.

Proof of Claim 15. Notice that $\text{Priv}(A, v) = \{u\}$. If y_{C_j} belongs to D for some $j \in \{1, \ldots, m\}$, then $\text{Priv}(D, v) = \emptyset$, a contradiction to the minimality of D. For similar reasons $u, z \notin D$. Hence $D \cap \{u, z, y_{C_1}, \ldots, y_{C_m}\} = \emptyset$. Besides, for every $i \in \{1, \ldots, m\}$, D contains at most one of x_i and $\neg x_i$, as otherwise $\text{Priv}(D, neg_{x_i})$ would be empty, again contradicting the minimality of D. This proves the claim. ◁

If A can be extended into a minimal dominating set D of G, then by combining the two claims above, we deduce that $\mathcal{I}$ has truth assignment that satisfies all clauses. Conversely, if $\mathcal{I}$ has such a truth assignment, then there is a set S as in the statement of Claim 14. In $S \cup A$, every element of S has a private neighbor, as a consequence of the minimality of S and the fact that no element of A has a neighbor among the clause variables. Besides, each of $neg_{x_1}, \ldots, neg_{x_n}$ has a private neighbor (because S contains at most one of the two literals for each variable) and it is easy to see that the same holds for v and w. Hence $S \cup A$ is a minimal dominating set of G.

Given an instance $\mathcal{I}$ of SAT, we constructed in polynomial time an instance (G, A) of Dcs that is equivalent to $\mathcal{I}$. This proves that Dcs is NP-hard. ◀

5 Conclusion

In this paper, we proved that the set of minimal dominating sets of a triangle-free graph, hence bipartite graph, can be enumerated in output-polynomial time. It remains open whether a polynomial delay algorithm exists for these classes.

The most general open problem on the topic discussed in this paper is whether the minimal dominating sets of a co-bipartite graph can be enumerated in output-polynomial time. Indeed, as noted in the introduction this would imply that such an algorithm also exists for the general case. Other classes where no output-polynomial time algorithms are known include unit disk graphs and graphs of bounded expansion, according to [20, 11].

References

1 Eralp Abdurrahim Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2(1):1–6, 1973.

2 John M. Barnard. Substructure searching methods: Old and new. *Journal of Chemical Information and Computer Sciences*, 33(4):532–538, 1993.

3 Jesper Makholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004.

4 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

5 Peter Damaschke. Parameterized Enumeration, Transversals, and Imperfect Phylogeny Reconstruction. In Rod Downey, Michael Fellows, and Frank Dehne, editors, *Parameterized and Exact Computation*, pages 1–12, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

6 Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and AI. In *European Workshop on Logics in Artificial Intelligence*, pages 549–564. Springer, 2002.

7 Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003. `arxiv:cs/0204009`.

8 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.

9 Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

10 Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve H. Sæther, and Yngve Villanger. Output-Polynomial Enumeration on Graphs of Bounded (Local) Linear MIM-Width. *Algorithmica*, 80(2):714–741, February 2018. arxiv:1509.03753. doi:10.1007/s00453-017-0289-1.

11 Petr A. Golovach, Pinar Heggernes, Mamadou M. Kanté, Dieter Kratsch, and Yngve Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Applied Mathematics*, 199:30–36, 2016. Special Issue: Sixth Workshop on Graph Classes, Optimization, and Width Parameters 2013. doi:10.1016/j.dam.2014.12.010.

12 Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An Incremental Polynomial Time Algorithm to Enumerate All Minimal Edge Dominating Sets. *Algorithmica*, 72(3):836–859, July 2015. doi:10.1007/s00453-014-9875-7.

13 Joshua A. Grochow and Manolis Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Annual International Conference on Research in Computational Molecular Biology*, pages 92–106. Springer, 2007.

14 Mihály Hujtera and Zsolt Tuza. The Number of Maximal Independent Sets in Triangle-Free Graphs. *SIAM Journal on Discrete Mathematics*, 6(2):284–288, 1993. doi:10.1137/0406022.

15 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. Enumeration of minimal dominating sets and variants. In *International Symposium on Fundamentals of Computation Theory*, pages 298–309. Springer, 2011. arxiv:1407.2053.

16 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the neighbourhood helly of some graph classes and applications to the enumeration of minimal dominating sets. In *International Symposium on Algorithms and Computation*, pages 289–298. Springer, 2012.

17 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the Enumeration of Minimal Dominating Sets and Related Notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014. arxiv:1407.2053.

18 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. On the enumeration and counting of minimal dominating sets in interval and permutation graphs. In *International Symposium on Algorithms and Computation*, pages 339–349. Springer, 2013.

19 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 138–153. Springer, 2015. arxiv:1407.2036.

20 Mamadou Moustapha Kanté and Lhouari Nourine. Minimal Dominating Set Enumeration. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–5. Springer US, Boston, MA, 2014. doi:10.1007/978-3-642-27848-8_721-1.

21 M. P. Marcus. Derivation of Maximal Compatibles Using Boolean Algebra. *IBM Journal of Research and Development*, 8(5):537–538, November 1964. doi:10.1147/rd.85.0537.

22 Andrea Marino. An Application: Biological Graph Analysis. In *Analysis and Enumeration: Algorithms for Biological Graphs*, pages 37–44. Atlantis Press, Paris, 2015. doi:10.2991/978-94-6239-097-3_3.

23 Andrea Marino. Enumeration Algorithms. In *Analysis and Enumeration: Algorithms for Biological Graphs*, pages 13–35. Atlantis Press, Paris, 2015. doi:10.2991/978-94-6239-097-3_2.

24 M. C. Paull and S. H. Unger. Minimizing the Number of States in Incompletely Specified Sequential Switching Functions. *IRE Transactions on Electronic Computers*, EC-8(3):356–367, September 1959. doi:10.1109/TEC.1959.5222697.

25 Yann Strozecki. *Enumeration complexity and matroid decomposition.* PhD thesis, Paris 7, 2010.

26 Robert Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing*, 2(3):211–216, 1973.

27 James C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Communications of the ACM*, 13(12):722–726, 1970.

28 Kunihiro Wasa. Enumeration of enumeration algorithms. *Preprint* `arxiv:1605.05102`, 2016. See also `https://kunihirowasa.github.io/enum/index` (accessed on September 2018).

29 Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777. ACM, 2005.

Sparsification of Binary CSPs

Silvia Butti
Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain
silvia.butti@upf.edu

Stanislav Živný
Department of Computer Science, University of Oxford, UK
standa.zivny@cs.ox.ac.uk

Abstract

A cut ε-sparsifier of a weighted graph G is a re-weighted subgraph of G of (quasi)linear size that preserves the size of all cuts up to a multiplicative factor of ε. Since their introduction by Benczúr and Karger [STOC'96], cut sparsifiers have proved extremely influential and found various applications. Going beyond cut sparsifiers, Filtser and Krauthgamer [SIDMA'17] gave a precise classification of which binary Boolean CSPs are sparsifiable. In this paper, we extend their result to binary CSPs on arbitrary finite domains.

2012 ACM Subject Classification Theory of Computation → Graph algorithms analysis

Keywords and phrases constraint satisfaction problems, minimum cuts, sparsification

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.17

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 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.
Silvia Butti: Work mostly done while at the University of Oxford.
Stanislav Živný: Stanislav Živný was supported by a Royal Society University Research Fellowship.

1 Introduction

The pioneering work of Benczúr and Karger [4] showed that every edge-weighted undirected graph $G = (V, E, w)$ admits a cut-sparsifier. In particular, assuming that the edge weights are positive, for every $0 < \varepsilon < 1$ there exists (and in fact can be found efficiently) a re-weighted subgraph $G_\varepsilon = (V, E_\varepsilon \subseteq E, w_\varepsilon)$ of G with $|E_\varepsilon| = O(\varepsilon^{-2} n \log n)$ edges such that

$$\forall S \subseteq V, \qquad \mathsf{Cut}_{G_\varepsilon}(S) \in (1 \pm \varepsilon)\mathsf{Cut}_G(S),$$

where $n = |V|$ and $\mathsf{Cut}_G(S)$ denotes the total weight of edges in G with exactly one endpoint in S. The bound on the number of edges was later improved to $O(\varepsilon^{-2}n)$ by Batson, Spielman, and Srivastava [3]. Moreover, the bound $O(\varepsilon^{-2}n)$ is known to be tight by the work of Andoni, Chen, Krauthgamer, Qin, Woodruff, and Zhang [2].

The original motivation for cut sparsification was to speed up algorithms for cut problems and graph problems more generally. The idea turned out to be very influential, with several generalisations and extensions, including, for instance, sketching [1, 2], sparsifiers for cuts in hypergraphs [9, 11], and spectral sparsification [15, 14, 13, 8, 12].

Filtser and Krauthgamer [7] considered the following natural question: which binary Boolean CSPs are sparsifiable? In order to state their results as well as our new results, we will now define binary constraint satisfaction problems.

An instance of the binary[1] *constraint satisfaction problem* (CSP) is a quadruple $I = (V, D, \Pi, w)$, where V is a set of variables, D is a finite set called the *domain*,[2] Π is a set of constraints, and $w : \Pi \to \mathbb{R}_+$ are positive weights for the constraints. Each constraint $\pi \in \Pi$ is a pair $((u, v), P)$, where $(u, v) \in V^2$, called the constraint *scope*, is a pair of distinct variables from V, and $P : D^2 \to \{0, 1\}$ is a binary predicate. A CSP instance is called *Boolean* if $|D| = 2$, i.e., if the domain is of size two.[3]

For a fixed binary predicate P, we denote by $\mathrm{CSP}(P)$ the class of CSP instances in which all constraints use the predicate P. Note that if we take $D = \{0, 1\}$ and P defined on D^2 by $P(x, y) = 1$ iff $x \neq y$ then $\mathrm{CSP}(P)$ corresponds to the cut problem.

We say that a constraint $\pi = ((u, v), P)$ is *satisfied* by an assignment $A : V \to D$ if $P(A(u), A(v)) = 1$. The value of an instance $I = (V, D, \Pi, w)$ under an assignment $A : V \to D$ is defined to be the total weight of satisfied constraints:

$$\mathrm{Val}_I(A) = \sum_{\pi=((u,v),P)\in\Pi} w(\pi)P(A(u), A(v)).$$

For $0 < \varepsilon < 1$, an ε-sparsifier of $I = (V, D, \Pi, w)$ is a re-weighted subinstance $I_\varepsilon = (V, D, \Pi_\varepsilon \subseteq \Pi, w_\varepsilon)$ of I such that

$$\forall A : V \to D, \qquad \mathrm{Val}_{I_\varepsilon}(A) \in (1 \pm \varepsilon)\,\mathrm{Val}_I(A).$$

The goal is to obtain a sparsifier with the minimum number of constraints, i.e., $|\Pi_\varepsilon|$.

A binary predicate P is called *sparsifiable* if for every instance $I \in \mathrm{CSP}(P)$ on $n = |V|$ variables and for every $0 < \varepsilon < 1$ there is an ε-sparsifier for I with $O(\varepsilon^{-2}n)$ constraints.

We call a (not necessarily Boolean or binary) predicate P a *singleton* if $|P^{-1}(1)| = 1$.

Filtser and Krauthgamer showed, among other results, the following classification. Let P be a binary Boolean predicate. Then, P is sparsifiable if and only if P is not a singleton.[4] In other words, the only predicates that are not sparsifiable are those with support of size one.

Contributions. As our main contribution, we identify in Theorem 2 the precise borderline of sparsifiability for binary predicates on arbitrary finite domains, thus extending the work from [7] on Boolean predicates. Let P be a binary predicate defined on an arbitrary finite domain D. Then, P is sparsifiable if and only if P does not "contain" a singleton subpredicate. More precisely, we say that P "contains" a singleton subpredicate if there are two (not necessarily disjoint) subdomains $B, C \subseteq D$ with $|B| = |C| = 2$ such that the restriction of P onto $B \times C$ is a singleton predicate.

The crux of Theorem 2 is the sparsifiability part, which is established by a reduction to cut sparsifiers. Unlike in the classification of binary Boolean predicates from [7], we do not rely on a case analysis that differs for different sparsifiable predicates but instead give a simpler argument for all sparsifiable predicates. The idea is to reduce (the graph of) any CSP instance, as was done in [7], via the so-called bipartite double cover [5]. However, there is no natural assignment in the reduced graph (as it was in the Boolean case in [7]). In order to overcome this, we define a graph G^P whose edges correspond to the support of the

[1] Some papers use the term *two-variable*.
[2] Some papers use the term *alphabet*.
[3] Some papers use the term *binary* to mean domains of size two. In this paper, *Boolean* always refers to a domain of size two and *binary* always refers to the arity of the constraint(s).
[4] Filtser and Krauthgamer use the term *valued CSPs* for what we defined as CSPs. We prefer CSPs in order to distinguish them from the much more general framework of valued CSPs studied in [10].

predicate P. Using a simple combinatorial argument, we show (in Proposition 7) that, under the assumption that P does not "contain" a singleton subpredicate, the bipartite complement of G^P is a collection of bipartite cliques. This special structure allows us to find a good assignment in the reduced graph.

In view of Filtser and Krauthgamer's work [7], one might conjecture that P is sparsifiable if and only if P is not a singleton. While it is easy to show that if a (possibly non-binary and non-Boolean) predicate P is a singleton then P is not sparsifiable, our results show that the borderline of sparsifiability lies elsewhere. In particular, by Theorem 2, there are binary non-Boolean predicates that are not sparsifiable but are not singletons. Also, there are non-binary Boolean predicates that are not sparsifiable but are not singletons.

We remark that the term "sparsification" is also used in an unrelated line of work in which the goal is, given a CSP instance, to reduce the number of constraints without changing satisfiability of the instance; see, e.g., [6].

2 Classification of Binary Predicates

Throughout the paper we denote by $n = |V|$ the number of variables of a given CSP instance.

The following classification of binary Boolean predicates is from [7].

▶ **Theorem 1** ([7, Theorem 3.7]). *Let $P : \{0,1\}^2 \to \{0,1\}$ be a binary Boolean predicate. Let $0 < \varepsilon < 1$.*

1. *If P is a singleton then there exists an instance I of $\mathrm{CSP}(P)$ such that every ε-sparsifier of I has $\Omega(n^2)$ constraints.*
2. *Otherwise, for every instance I of $\mathrm{CSP}(P)$ there exists an ε-sparsifier of I with $O(\varepsilon^{-2}n)$ constraints.*

We denote by $\binom{D}{2} = \{B \subseteq D : |B| = 2\}$ the set of two-element subsets of D. For a binary predicate $P : D^2 \to \{0,1\}$ and $B, C \in \binom{D}{2}$, $P|_{B\times C}$ denotes the restriction of P onto $B \times C$.

The following is our main result, generalising Theorem 1 to arbitrary finite domains.

▶ **Theorem 2 (Main).** *Let $P : D^2 \to \{0,1\}$ be a binary predicate, where D is a finite set with $|D| \geq 2$. Let $0 < \varepsilon < 1$.*

1. *If there exist $B, C \in \binom{D}{2}$ such that $P|_{B\times C}$ is a singleton then there exists an instance I of $\mathrm{CSP}(P)$ such that every ε-sparsifier of I has $\Omega(n^2)$ constraints.*
2. *Otherwise, for every instance I of $\mathrm{CSP}(P)$ there exists an ε-sparsifier of I with $O(\varepsilon^{-2}n)$ constraints.*

The rest of this section is devoted to proving Theorem 2.

First we introduce some useful notation. We set $[r] = \{0, 1, \ldots, r-1\}$. We denote by $X \sqcup Y$ the disjoint union of X and Y. For any $r \geq 2$, we define r-$\mathsf{Cut} : [r]^2 \to \{0,1\}$ by r-$\mathsf{Cut}(x, y) = 1$ if and only if $x \neq y$.

Given an instance $I = (V, D, \Pi, w) \in \mathrm{CSP}(P)$, we denote by G^I the *corresponding graph* of I; that is, $G^I = (V, E, w)$ is a weighted directed graph with $E = \{(u, v) : ((u, v), P) \in \Pi\}$ and $w(u, v) = w((u, v), P)$. Conversely, given a weighted directed graph $G = (V, E, w)$ and a predicate $P : D^2 \to \{0,1\}$, the *corresponding* $\mathrm{CSP}(P)$ *instance* is $I^{G,P} = (V, D, \Pi, w)$, where $\Pi = \{(e, P) : e \in E\}$ and $w(e, P) = w(e)$. Hence, we can equivalently talk about instances of $\mathrm{CSP}(P)$ or (weighted directed) graphs. Thus, an *ε-P-sparsifier* of a graph $G = (V, E, w)$ is a subgraph $G_\varepsilon = (V, E_\varepsilon \subseteq E, w_\varepsilon)$ whose corresponding $\mathrm{CSP}(P)$ instance $I^{G_\varepsilon,P}$ is an ε-sparsifier of the corresponding $\mathrm{CSP}(P)$ instance $I^{G,P}$ of G.

Case (1) of Theorem 2 is established by the following result.

▶ **Theorem 3.** *Let $P : D^2 \to \{0,1\}$ be a binary predicate. Assume that there exist $B, C \in \binom{D}{2}$ such that $P|_{B\times C}$ is a singleton. For any n there is a* CSP(P) *instance I with $2n$ variables and n^2 constraints such that for any $0 < \varepsilon < 1$ it holds that any ε-sparsifier of I has n^2 constraints.*

Proof. Suppose $B = \{b, b'\}$, $C = \{c, c'\}$ and assume without loss of generality that ${P|_{B\times C}}^{-1}(1) = \{(b, c)\}$; that is, the support of $P|_{B\times C}$ is equal to $\{(b, c)\}$. Consider a CSP(P) instance $I = (V, D, \Pi, w)$, where

- $V = X \sqcup Y$, $X = \{x_1, \ldots, x_n\}$, and $Y = \{y_1, \ldots, y_n\}$;
- $\Pi = \{\pi_{ij} = ((x_i, y_j), P) : 1 \le i, j \le n\}$;
- w are arbitrary positive weights.

We have $|\Pi| = n^2$. We note that B and C may not be disjoint. We consider the family of assignments $A_{ij} : V \to B \cup C$ for $1 \le i, j \le n$ such that $A_{ij}(x_i) = b$, $A_{ij}(x) = b'$ for every $x \in X \setminus \{x_i\}$, $A_{ij}(y_j) = c$, and $A_{ij}(y) = c'$ for every $y \in Y \setminus \{y_j\}$. Then, we have

$$P(A_{ij}(u, v)) = \begin{cases} P(b, c) = 1 & \text{if } u = x_i, v = y_j, \\ P(b, c') = 0 & \text{if } u = x_i, v \in Y \setminus \{y_j\}, \\ P(b', c) = 0 & \text{if } u \in X \setminus \{x_i\}, v = y_j, \\ P(b', c') = 0 & \text{if } u \in X \setminus \{x_i\}, v \in Y \setminus \{y_j\}. \end{cases}$$

Therefore,

$$\mathrm{Val}_I(A_{ij}) = \sum_{\pi\in\Pi} w(\pi)P(A_{ij}(\pi)) = w(\pi_{ij}) > 0.$$

Hence, if $I_\varepsilon = (V, D, \Pi_\varepsilon, w_\varepsilon)$ is an ε-sparsifier of I, we must have that $\pi_{ij} \in \Pi_\varepsilon$ for every $1 \le i, j \le n$, as otherwise we would have

$$\mathrm{Val}_{I_\varepsilon}(A_{ij}) = \sum_{\pi\in\Pi_\varepsilon} w_\varepsilon(\pi)P(A_{ij}(\pi)) = 0 \notin (1 \pm \varepsilon)\,\mathrm{Val}_I(A_{ij}).$$

Therefore, we have $\Pi_\varepsilon = \Pi$ and hence $|\Pi_\varepsilon| = |\Pi| = n^2$. ◀

The main tool used in the proof of Theorem 1 (2) from [7] is a graph transformation known as the bipartite double cover [5], which allows for a reduction to cut sparsifiers [3].

▶ **Definition 4.** *For a weighted directed graph $G = (V, E, w)$, the* bipartite double cover *of G is the weighted directed graph $\gamma(G) = (V^\gamma, E^\gamma, w^\gamma)$, where*

- $V^\gamma = \{v^{(0)}, v^{(1)} : v \in V\}$;
- $E^\gamma = \{(u^{(0)}, v^{(1)}) : (u, v) \in E\}$;
- $w^\gamma(u^{(0)}, v^{(1)}) = w(u, v)$.

Given an assignment $A : V \to [r]$, we let $\mathcal{A} = (A_0, \ldots, A_{r-1})$ be the induced r-partition of V, where $A_j = A^{-1}(j)$. For a binary predicate $P : [r]^2 \to \{0, 1\}$ and an instance $I = (V, [r], \Pi, w) \in$ CSP(P), we define $\mathrm{Val}_I(\mathcal{A}) = \mathrm{Val}_I(A)$. Moreover, for a weighted directed graph G and a binary predicate P, we define $\mathrm{Val}_{G,P}(\mathcal{A}) = \mathrm{Val}_{I^{G,P}}(\mathcal{A})$. We denote the set of all r-partitions of V by $Part_r(V)$.

For any r-partition $\mathcal{A} = (A_0, \ldots, A_{r-1})$ of the vertices of V, let $A_i^{(j)} = \{v^{(j)} : v \in A_i\}$. Thus $\mathcal{A}^\gamma = (A_0^{(0)}, A_0^{(1)}, \ldots, A_{r-1}^{(0)}, A_{r-1}^{(1)})$ is a $2r$-partition of the vertices of V^γ.

We use an argument from the proof of Theorem 1 (2) from [7] and apply it to non-Boolean predicates.

▶ **Proposition 5.** *Let $P : [r]^2 \to \{0,1\}$ and $P' : [r']^2 \to \{0,1\}$ be binary predicates. Suppose that there is a function $f_P : Part_r(V) \to Part_{r'}(V^\gamma)$ such that for any weighted directed graph G on V and for any r-partition $\mathcal{A} \in Part_r(V)$ it holds that*

$$\mathrm{Val}_{G,P}(\mathcal{A}) = \mathrm{Val}_{\gamma(G),P'}(f_P(\mathcal{A})),$$

where $\gamma(G) = (V^\gamma, E^\gamma, w^\gamma)$ is the bipartite double cover of G. If there is an ε-P'-sparsifier of $\gamma(G)$ of size $g(n)$ then there is an ε-P-sparsifier of G of size $O(g(n))$.

Proof. Given $G = (V, E, w)$, let $\gamma(G)_\varepsilon = (V, E^\gamma_\varepsilon, w^\gamma_\varepsilon)$ be an ε-P'-sparsifier of the bipartite double cover $\gamma(G)$ of G. Define a subgraph $G_\varepsilon = (V, E_\varepsilon, w_\varepsilon)$ of G by $E_\varepsilon = \{(u,v) : (u^{(0)}, v^{(1)}) \in E^\gamma_\varepsilon\}$ and $w_\varepsilon(u,v) = w^\gamma_\varepsilon(u^{(0)}, v^{(1)})$. Notice that $\gamma(G_\varepsilon) = \gamma(G)_\varepsilon$, $E_\varepsilon \subseteq E$, and $E^\gamma = O(|E|)$.

Then, we have

$$\begin{aligned}\mathrm{Val}_{G_\varepsilon,P}(\mathcal{A}) &= \mathrm{Val}_{\gamma(G_\varepsilon),P'}(f_P(\mathcal{A})) \\ &= \mathrm{Val}_{\gamma(G)_\varepsilon,P'}(f_P(\mathcal{A})) \in (1 \pm \varepsilon)\,\mathrm{Val}_{\gamma(G),P'}(f_P(\mathcal{A})) = (1 \pm \varepsilon)\,\mathrm{Val}_{G,P}(\mathcal{A})\end{aligned}$$

and

$$|E_\varepsilon| \le |E^\gamma_\varepsilon| = O\Big(\frac{|V^\gamma|}{\varepsilon^2}\Big) = O\Big(\frac{|V|}{\varepsilon^2}\Big),$$

implying that G_ε is also an ε-P-sparsifier of G.

Moreover, $|E_\varepsilon| \le |E^\gamma_\varepsilon| = g(n)$ implies $|E_\varepsilon| = O(g(n))$. ◀

We now focus on proving Case (2) of Theorem 2. Assume that for any $B, C \in \binom{D}{2}$, $P|_{B\times C}$ is not a singleton. Our strategy is to show that in this case the value of a CSP(P) instance under any assignment can be expressed as the value of a corresponding CSP(ℓ-Cut) instance (for some $\ell \le 2|D|$) under the same assignment.

For an undirected graph $G = (V, E)$ and a subset $U \subseteq V$, we denote the vertex-induced subgraph on U by $G[U]$ and its edge set by $E[U]$. For a possibly disconnected undirected graph G, we denote the connected component containing a vertex v by $G_v = (V(G_v), E(G_v))$. Finally, we denote the degree of vertex v in graph G by $d_G(v)$.

▶ **Definition 6.** *Let $G = (U \sqcup V, E)$ be an undirected bipartite graph. The* bipartite complement *$\overline{G} = (U \sqcup V, \overline{E})$ of G has the following edge set:*

$$\overline{E} = \{\{u,v\} : u \in U, v \in V, \{u,v\} \notin E\}.$$

The following property of bipartite graphs will be crucial in the proof of Theorem 8.

▶ **Proposition 7.** *Let $G = (U \sqcup V, E)$ be a bipartite graph with $|U| = |V| = r$, $r \ge 2$. Assume that for any $u, u' \in U$ and $v, v' \in V$ we have $|E[\{u, u', v, v'\}]| \ne 1$. Then, for any $v \in U \sqcup V$ with $d_{\overline{G}}(v) > 0$, $\overline{G}_v$ is a complete bipartite graph with partition classes $\{U \cap V(\overline{G}_v)\}$ and $\{V \cap V(\overline{G}_v)\}$.*

Proof. For contradiction, assume that there are $u \in U$ and $v \in V$ such that $\{u,v\} \notin \overline{E}$ but u and v belong to the same connected component of $\overline{G}$. Choose u and v with the shortest possible distance between them. Let $u = u_0, u_1, \ldots, u_k = v$ be a shortest path between u and v in $\overline{G}$, where $k \ge 3$ is odd. We will show that $|\overline{E}[\{u_0, u_1, u_{k-1}, u_k\}]| = 3$, which contradicts the assumption that $|E[\{u_0, u_1, u_{k-1}, u_k\}]| \ne 1$.

If $k = 3$ then the claim holds since we assumed that $\{u_0, u_1\}, \{u_1, u_2\}, \{u_2, u_3\} \in \overline{E}$ and $\{u_0, u_3\} \notin \overline{E}$.

Let $k \geq 5$. We will be done if we show that $\{u_1, u_{k-1}\} \in \overline{E}$, as by our assumptions $\{u_0, u_1\}, \{u_{k-1}, u_k\} \in \overline{E}$ and $\{u_0, u_k\} \notin \overline{E}$. To this end, note that $\{u_0, u_{k-2}\} \in \overline{E}$ as otherwise u_0 and u_{k-2} would be a pair of vertices with the required properties but of distance $k-2$, contradicting our choice of u and v. Thus, $\{u_1, u_{k-1}\} \in \overline{E}$ as otherwise we would have $|\overline{E}[\{u_0, u_1, u_{k-2}, u_{k-1}\}]| = 3$, which contradicts $|E[\{u_0, u_1, u_{k-2}, u_{k-1}\}]| \neq 1$. ◀

Case (2) of Theorem 2 is established by the following result.

▶ **Theorem 8.** *Let $P : D^2 \to \{0,1\}$ be a binary predicate such that for any $B, C \in \binom{D}{2}$ we have that $P|_{B \times C}$ is not a singleton. Then, for every $0 < \varepsilon < 1$ and every instance I of* CSP(P) *there is a sparsifier of I with $O(\varepsilon^{-2} n)$ constraints.*

Proof. Let $I = (V, D, \Pi, w)$ be an instance of CSP(P) with $r = |D|$. Without loss of generality, we assume that $D = [r]$. Let $G = G^I = (V, E, w)$ be the corresponding (weighted directed) graph of I, and let $\gamma(G) = (V^\gamma, E^\gamma, w^\gamma)$ be the bipartite double cover of G. Recall that for an assignment $A : V \to [r]$, we denote $A_i = A^{-1}(i)$. Thus, $\mathcal{A} = (A_0, \ldots, A_{r-1})$ forms an r-partition of V.

Our goal is to show the existence of a function $f_P : Part_r(V) \to Part_\ell(V^\gamma)$ (for some fixed $\ell \leq 2r$) such that

$$\forall A : V \to [r], \qquad \mathrm{Val}_{G,P}(\mathcal{A}) = \mathrm{Val}_{\gamma(G),\ell\text{-}\mathsf{Cut}}(f_P(\mathcal{A})). \tag{1}$$

Assuming the existence of f_P, we can finish the proof as follows. Batson, Spielman, and Srivastava established the existence of a sparsifier of size $O(\varepsilon^{-2} n)$ for any instance of CSP(2-Cut) [3]. By [7, Section 6.2], this implies the existence of a sparsifier of size $O(\varepsilon^{-2} n)$ for any instance of CSP(ℓ-Cut). Consequently, by Proposition 5 and (1), there is a sparsifier of size $O(\varepsilon^{-2} n)$ for the instance $I^{G,P} = I$.

It remains to show the existence of f_P satisfying (1).

In the proof of Theorem 1 (2) in [7], such functions are given for a binary Boolean predicate P with support size $|P^{-1}(1)| \in \{0, 2, 4\}$. In what follows we give a construction of f_P for an *arbitrary* binary predicate $P : [r]^2 \to \{0,1\}$ with $r \geq 2$ from the statement of the theorem.

Although the bipartite double cover is commonly defined as a directed graph, in this proof we will consider the *undirected* bipartite double cover $\gamma(G)$ of G.[5] We also define an auxiliary graph $G^P = (V^P, E^P)$, where

$$V^P = \{v_0, v'_0, \ldots, v_{r-1}, v'_{r-1}\},$$

$$E^P = \{\{v_i, v'_j\} : P(i,j) = 1\}.$$

Let ℓ be the number of connected components of $\overline{G^P}$, the bipartite complement of G^P. By definition, $\ell \leq |V^P| = 2r$.

We need to find a function $f_P : Part_r(V) \to Part_\ell(V^\gamma)$ that satisfies (1) for all $\mathcal{A} \in Part_r(V)$. Such a function corresponds to a map $c : V^P \to [\ell]$ on the vertices of G^P with the following property:

$$\forall i, j \in [r] \quad \begin{cases} \{v_i, v'_j\} \in E^P \implies c(v_i) \neq c(v'_j) \\ \{v_i, v'_j\} \notin E^P \implies c(v_i) = c(v'_j). \end{cases}$$

[5] We had defined the bipartite double cover as a directed graph. However, here it is easier to deal with undirected graphs, as since ℓ-Cut is a symmetric predicate, the direction of the edges makes no difference. Furthermore, notice that by the way the bipartite double cover is constructed, removing the direction does *not* turn the graph into a multigraph.

We call such maps *colourings*. Indeed, the colouring c induces, for $\mathcal{A}$, an assignment $A^\gamma : V^\gamma \to [\ell]$ of the vertices of $\gamma(G)$ which satisfies

$$A^\gamma(u) = c(v_{A(u)}) \quad \text{and} \quad A^\gamma(u') = c(v'_{A(u)})$$

and which, in turn, induces a partition $\{U_i\}_{i=0}^{\ell-1}$ of V^γ with $U_i = (A^\gamma)^{-1}(i)$. We define $f_P(\mathcal{A}) = (U_0, \dots, U_{\ell-1})$. Now for any $u, v \in V$ and for any assignment $A : V \to [r]$, we have

$$\begin{aligned} P(A(u), A(v)) = 1 &\iff \{v_{A(u)}, v'_{A(v)}\} \in E^P \\ &\iff c(v_{A(u)}) \neq c(v'_{A(v)}) \\ &\iff A^\gamma(u) \neq A^\gamma(v') \\ &\iff \ell\text{-}\mathsf{Cut}(A^\gamma(u), A^\gamma(v')) = 1. \end{aligned}$$

Moreover, by the definition of the graph bipartite double cover, we have $w(u, v) = w^\gamma(u, v')$ for all $u, v \in V$, implying that

$$\begin{aligned} \mathrm{Val}_{G,P}(\mathcal{A}) = \mathrm{Val}_{G,P}(A_0, \dots, A_{r-1}) &= \sum_{(u,v)\in E} w(u, v) P(A(u), A(v)) \\ &= \sum_{(u,v')\in E^\gamma} w^\gamma(u, v')\ell\text{-}\mathsf{Cut}(A^\gamma(u), A^\gamma(v')) = \mathrm{Val}_{\gamma(G),\ell\text{-}\mathsf{Cut}}(A^\gamma) \\ &= \mathrm{Val}_{\gamma(G),\ell\text{-}\mathsf{Cut}}(U_0, \dots, U_{\ell-1}) = \mathrm{Val}_{\gamma(G),\ell\text{-}\mathsf{Cut}}(f_P(\mathcal{A})) \end{aligned}$$

as required.

While a colouring does not exist for an arbitrary bipartite graph, we now argue that a colouring does exist if the auxiliary graph G^P arises from a predicate P from the statement of the theorem. Since for any $B, C \in \binom{[r]}{2}$ we have $|P|_{B\times C}{}^{-1}(1)| \neq 1$, G^P satisfies the assumptions of Proposition 7. Therefore, the ℓ separate connected components which form its bipartite complement $\overline{G^P}$ are complete bipartite graphs. We can assign one of the ℓ colours to each connected component to get a colouring for the graph G^P. ◀

3 Conclusion

For simplicity, we have only presented our main result on binary CSPs over a single domain. However, it is not difficult to extend our result to the so-called *multisorted* binary CSPs, in which different variables come with possibly different domains.

We have classified *binary* CSPs (on finite domains) but much more work seems required for a full classification of non-binary CSPs. We have made some initial steps.

For any $k \geq 3$, the k-ary Boolean "not-all-equal" predicate k-$\mathsf{NAE} : \{0,1\}^k \to \{0,1\}$ is defined by $k\text{-}\mathsf{NAE}^{-1}(0) = \{(0, \dots, 0), (1, \dots, 1)\}$. Kogan and Krauthgamer showed that the k-NAE predicates, which correspond to hypergraph cuts, are sparsifiable [9, Theorem 3.1]. By extending bipartite double covers for graphs in a natural way to k-partite k-fold covers, we obtain sparsifiability for the class of k-ary predicates that can be rewritten in terms of k-NAE. On the other hand, we identify a whole class of predicates that are not sparsifiable, namely those k-ary predicates that contain a singleton ℓ-cube for some $\ell \leq k$. However, there are predicates which do not fall in either of these two categories; that is, predicates that cannot be proved sparsifiable via k-partite k-fold covers but also cannot be proved non-sparsifiable via the current techniques. An example of such predicates are the "parity" predicates.

References

1. Kook Jin Ahn and Sudipto Guha. Graph Sparsification in the Semi-streaming Model. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09), Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 328–338. Springer, 2009. doi:10.1007/978-3-642-02930-1_27.
2. Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On Sketching Quadratic Forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (ITCS'16)*, pages 311–319, 2016. doi:10.1145/2840728.2840753.
3. Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan Sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012. doi:10.1137/090772873.
4. András A. Benczúr and David R. Karger. Approximating s-t Minimum Cuts in $\tilde{O}(n^2)$ Time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC'96)*, pages 47–55, 1996. doi:10.1145/237814.237827.
5. Richard A. Brualdi, Frank Harary, and Zevi Miller. Bigraphs versus digraphs via matrices. *Journal of Graph Theory*, 4(1):51–73, 1980. doi:10.1002/jgt.3190040107.
6. Hubie Chen, Bart M. P. Jansen, and Astrid Pieterse. Best-case and Worst-case Sparsifiability of Boolean CSPs. In *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC'18)*, 2018. arXiv:1809.06171.
7. Arnold Filtser and Robert Krauthgamer. Sparsification of Two-Variable Valued Constraint Satisfaction Problems. *SIAM Journal on Discrete Mathematics*, 31(2):1263–1276, 2017. doi:10.1137/15M1046186.
8. Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC'11)*, pages 71–80. ACM, 2011. doi:10.1145/1993636.1993647.
9. Dmitry Kogan and Robert Krauthgamer. Sketching Cuts in Graphs and Hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS'15)*, pages 367–376, 2015. doi:10.1145/2688073.2688093.
10. Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The Complexity of General-Valued CSPs. *SIAM Journal on Computing*, 46(3):1087–1110, 2017. doi:10.1137/16M1091836.
11. Ilan Newman and Yuri Rabinovich. On Multiplicative Lambda-Approximations and Some Geometric Applications. *SIAM Journal on Computing*, 42(3):855–883, 2013. doi:10.1137/100801809.
12. Tasuko Soma and Yuichi Yoshida. Spectral Sparsification of Hypergraphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, 2019.
13. Daniel A. Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.
14. Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 81–90. ACM, 2004. doi:10.1145/1007352.1007372.
15. Daniel A. Spielman and Shang-Hua Teng. Spectral Sparsification of Graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

Tractable QBF by Knowledge Compilation

Florent Capelli
Université de Lille, Inria, UMR 9189 – CRIStAL – Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
florent.capelli@univ-lille.fr

Stefan Mengel
CNRS, CRIL UMR 8188, Lens, France
mengel@cril.fr

Abstract

We generalize several tractability results concerning the tractability of Quantified Boolean Formulas (QBF) with restricted underlying structure. To this end, we introduce a notion of width for structured DNNF which are a class of Boolean circuits heavily studied in knowledge compilation, a subarea of artificial intelligence. We then show that structured DNNF allow quantifier elimination with a size blow-up depending only on the width of the DNNF and not its size. Using known algorithms transforming restricted CNF-formulas into deterministic DNNF, we apply this result to generalize several results for counting and decision on QBF. We also complement these results with lower bounds that show that our definitions and results are essentially optimal in several senses.

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

Keywords and phrases QBF, knowledge compilation, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.18

Related Version A full version of the paper is available at [7], `https://arxiv.org/abs/1807.04263`.

Funding This work was partially supported by the French Agence Nationale de la Recherche, AGGREG project reference ANR-14-CE25-0017-01.

Acknowledgements The authors would like to thank Mikaël Monet for helpful comments on an early version of this paper.

1 Introduction

It is well known that restricting the interaction between variables and clauses in CNF-formulas makes several hard problems on them tractable. For example, the propositional satisfiability problem SAT and its counting version #SAT can be solved in time $2^{O(k)}|F|$ when F is a CNF formula whose primal graph is of treewidth k [27, 24]. Many extensions of this result have been shown these last ten years for more general graph measures such as modular treewidth or cliquewidth [15, 19, 26, 23]. We here generalize in a different direction by considering decision and counting for quantified Boolean formulas (QBF) with a bounded number of quantifier alternations, i.e., we consider problems higher up in the polynomial hierarchy than SAT, resp. higher in the counting hierarchy than #SAT. It is already known that QBF as well as projected model counting, i.e., model counting for QBF with free variables and one block of existentially quantified variables, are both fixed-parameter tractable parameterized by treewidth [8, 14]. Here we generalize both these results by showing that counting the models of QBF with free variables is fixed-parameter tractable parameterized by treewidth for any bounded number of quantifier alternations. Moreover, the same is true for the strictly more general parameter of signed cliquewidth [15].

Our approach to showing these results is completely different from those used so far in the literature for treewidth restrictions of problems harder than the NP, resp. #P: we do not perform dynamic programming as e.g. in [8, 14, 12, 1]. Instead, we encode all models

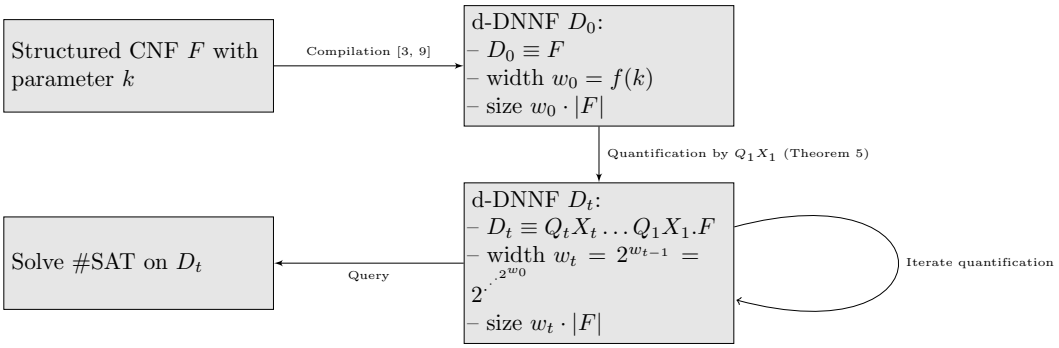


Figure 1 The overall scheme for proving tractability results on structured quantified CNF.

of the underlying CNF-formula of the given QBF into a data structure called *complete structured d-DNNF* [21], a class of circuits originating from knowledge compilation, a subarea of artificial intelligence [11]. Afterwards, we perform quantifier elimination on this data structure. When all quantifiers are eliminated, we can answer the query on the input QBF by standard algorithms for d-DNNF. Figure 1 illustrates the overall strategy.

One crucial advantage of our approach is that the first step, the transformation into d-DNNF also called *compilation*, is essentially already solved in the literature: in [3], Bova et al. recently showed that the traces of most known algorithms for structural restrictions of #SAT are essentially d-DNNF. Thus we can take these algorithms as building blocks and get the compiled representations for free without doing any additional dynamic programming.

It thus only remains to eliminate quantifiers on d-DNNF. Unfortunately, there are unconditional, exponential lower bounds showing that in general quantifier elimination on d-DNNF is impossible without blowing up the size of the representation [22]. We avoid this problem by identifying a notion of *width* for complete structured d-DNNF that is modeled after the classical width of complete OBDD. We go on to show that the size explosion during the quantifier elimination is in fact not in the size of the input but only in its width by giving a relatively simple algorithm inspired by determinization of finite automata. Since several of the compilation algorithms mentioned above yield d-DNNF whose width is independent of the input size, we get an algorithm for several restricted classes of QBF.

The resulting algorithm can be used to show that the number of models of a partially quantified CNF-formula F of treewidth k with t blocks of quantifiers can be computed in time $2^{\cdot^{\cdot^{\cdot^{2^{O(k)}}}}} |F|$ with $t+1$ exponentiations. This generalizes the result of [8] where the fixed-parameter tractability of QBF on such formulas was shown with a comparable complexity. Moreover, it generalizes the very recent result of [14] on model counting in the presence of a single existential variable block. Finally, our algorithm also applies to the more general notions of incidence treewidth and signed cliquewidth.

We complement our algorithm with lower bounds that show that our construction is essentially optimal in several respects.

The paper is organized as follows: Section 2 introduces the necessary preliminaries. Section 3 showcases our approach in a simple setting by proving that quantifier elimination can be efficiently done on small width complete OBDD. Section 4 first introduces our width notion on complete structured d-DNNF, shows some of its basic properties and then generalizes the result of Section 3 in this setting, giving our main result. The rest of the paper is dedicated to corollaries of this result proven in Section 4 and explores the limits

and optimality of our approach. Section 5 is dedicated to proving parameterized tractability results for QBF. Section 6 contains several results showing that our definition of bounded width d-DNNF cannot be weakened in several directions while still supporting efficient quantifier elimination. Finally, we close with a conclusion.

2 Preliminaries

By $\exp^\ell(p)$ we denote the iterated exponentiation function that is defined by $\exp^0(p) := p$ and $\exp^{\ell+1}(p) := 2^{\exp^\ell(p)}$.

CNF and QBF. We assume that the reader is familiar with the basics of Boolean logic and fix some notation. For a Boolean function F and a partial assignment τ to the variables of F, denote by $F[\tau]$ the function we get from F by fixing the variables of τ according to τ. For two assignments τ, σ on disjoint sets of variables we write $\tau \cup \sigma$ for the assignment on all variables of τ and σ that extends both of the assignments. A *literal* is a Boolean variable or its negation. A *clause* is a disjunction of literals and finally a *formula in conjunctive normal form* (short *CNF formula*) is a conjunction of clauses. We define the *size* $|C|$ of a clause C as the number of literals appearing in it. The size $|F|$ of a formula F is then defined as $\sum_C |C|$ where the sum is over the clauses in F.

A *Quantified Boolean Formula* (short *QBF*) $F = Q_1X_1Q_2X_2 \dots Q_\ell X_\ell \; F'$ is a CNF formula F' together with a *quantified prefix* $Q_1X_1Q_2X_2 \dots \exists X_\ell$ where $X_1, \dots, X_\ell$ are disjoint subsets of variables of F', Q_i is either $\exists$ or $\forall$ and $Q_{i+1} \neq Q_i$. The number of blocks ℓ is called the *quantifier alternation*. W.l.o.g, we assume that Q_ℓ, the most nested quantifier, is always an $\exists$-quantifier. The *quantified variables* of F are defined as $\bigcup_{i=1}^{\ell} X_i$ and the *free variables* of F are the variables of F that are not quantified. A quantified CNF naturally induces a Boolean function on its free variables.

Representations of Boolean functions. We present several representations studied in the area of knowledge compilation in a rather succinct fashion. For more details and discussion, the interested reader is referred to [11, 21].

A Boolean circuit C is defined to be in *negation normal form* (short an NNF) if $\neg$-gates appear in it only directly above the inputs. We assume that in all circuits we consider all $\wedge$-gates have exactly two inputs while all $\vee$-gates have an arbitrary positive number of inputs. An $\wedge$-gate in an NNF is called *decomposable* if, for its inputs g_1, g_2 the subcircuits rooted in g_1 and g_2 are on disjoint variable sets. A circuit in *decomposable negation normal form* (short a DNNF) is an NNF in which all gates are decomposable [9]. An $\vee$-gate g in an NNF is called *deterministic* if there is no assignment to the variables of the circuit that makes two children of g true. A DNNF is said to be *deterministic* (short a *d-DNNF*) if all its $\vee$-gates are deterministic.

A *binary decision diagram* (short *BDD*) is a directed acyclic graph with the following properties: there is one source and two sinks, one of each labeled with 0 and 1. The non-sink nodes are labeled with Boolean variables and have two outgoing edges each, one labeled with 0 the other with 1. A BDD B computes a function as follows: for every assignment a to the variables of B, one constructs a source-sink path by starting in the source and in every node labeled with a Boolean variable X following the edge labeled with $a(X)$. The label of the sink reached this way is then the value computed by B on a.

A BDD is called a *free* BDD (short *FBDD*) if on every source-sink path every variable appears at most once. If on every path the variables are seen in a fixed order π, then the FBDD is called an ordered BDD (short *OBDD*).

An FBDD is called *complete* if on every source-sink path every variable appears exactly once. This notion also applies to OBDD in the obvious way. A *layer* of a variable X in a complete OBDD B is the set of all nodes labeled with X. The *width* of B is the maximum size of its layers. Note that for every OBDD one can construct a complete OBDD computing the same function in polynomial time, but it is known that it is in general unavoidable to increase the number of nodes labeled by a variable by a factor linear in the number of variables [2].

For any representation D of a Boolean function in one of the above forms, we denote by $\mathsf{var}(D)$ the set of variables appearing in D.

Graphs of CNF formulas. There are two graphs commonly assigned to CNF formulas: the *primal graph* of a CNF formula F is the graph that has as its vertices the variables of F and there is an edge between two vertices x, y if and only if there is clause in F where both variables x and y appear. The *incidence graph* of F has as vertices the variables and the clauses of F and there is an edge between two nodes x and C if and only if x is a variable, C is a clause, and x appears in C.

We will consider several width measures on graphs like treewidth and pathwidth. Since we do not actually need the definitions of these measures but only depend on known results on them, we spare the readers these rather technical definitions and give pointers to the literature in the respective places.

3 Warm-up: Quantification on OBDD

In this section, we will illustrate the main ideas of our approach on the simpler case of OBDD. To this end, fix an OBDD G in variables $x_1, \ldots, x_n$ in that order. Now let Z be a set of variables. We want to compute an OBDD that encodes $\exists Z\, G$, i.e., we want to forget the variables in Z.

Note that it is well-known that OBDD do not allow arbitrary forgetting of variables without an exponential blow-up, see [11]. Here we make the observation that this exponential blow-up is in fact not in the *size* of the considered OBDD but in the *width* which for many interesting cases is far lower.

▶ **Lemma 1.** *Let G be a complete OBDD of width w and Z be a subset of its variables. Then there is an OBDD of width at most 2^w that computes the function of $\exists Z\, G$.*

Proof. The technique is essentially the power set construction used in the determinization of finite automata. Let V_x for a variable x denotes the set of nodes labeled by x. For every x not in Z, our new OBDD G' will have a node N_S labeled by x for every subset $S \subseteq V_x$. The invariant during the construction will be that a partial assignment a to the variables in $\mathsf{var}(G) \setminus Z$ that come before x in G leads to N_S if and only if S is the set of nodes in V_x which can be reached from the source by an extension of a on the variables of Z. We make the same construction for the 0- and 1-sink of G: G' gets three sinks 0, 1 and 01 which encode which sinks of G can be reached with extensions of an assignment a. Note that if we can construct such a G', we are done by merging the sinks 1 and 01.

The construction of G' is fairly straightforward: consider a variable x not in Z and let x' be the next variable not in Z. For every node $N \in V_x$, we compute the set of nodes N^+ labeled with x' that we can reach by following the 1-edge of N and the set of N^- nodes labeled with x' that we can reach by following the 0-edge of N. Then, for every $S \subseteq V_x$ we define the 1-successor of N_S as $N_{S'}$ where $S' = \bigcup_{N \in S} N^+$. The 0-successors are defined analogously. ◀

We remark that in [13] a related result is shown: for a CNF-formula F of pathwidth k and every subset Z of variables, one can construct an OBDD of size $2^{2^k}|F|$ computing $\exists Z\ F$. This result follows easily from Lemma 1 by noting that for a CNF F of pathwidth k one can construct a complete OBDD of width 2^k. We note that our approach is more flexible than the result in [13] because we can iteratively add more quantifier blocks since $\forall Z\ D \equiv \neg(\exists Z \neg D)$ and negation in OBDD can be easily performed without size increase. For example, one directly gets the following corollary.

▶ **Corollary 2.** *There is an algorithm that, given a QBF restricted to ℓ quantifier alternations and of pathwidth k, decides if F is true in time $O(\exp^{\ell}(p)|F|)$.*

Note that Corollary 2 is already known as it is a special case of the corresponding result for treewidth in [8]. However, we will show that a similar approach to that of Lemma 1 can be used to derive several generalizations of the result of [8]: we show that we can add quantification to bounded width structured d-DNNF, a generalization of OBDD (see Section 4). Since several classes of CNF formulas are known to yield bounded width structured d-DNNF [3], this directly yields QBF algorithms for these classes, see Section 5 for details.

4 Bounded width complete structured DNNF

Before formulating and proving our main result, we first introduce our central data structure called *complete structured DNNF* as a generalization of OBDD and a restriction of the structured DNNF from [21]. We introduce a width notion for it and show how to deal with constants in the setting. After these preparations, we then show our main result on eliminating quantifiers in Section 4.4.

4.1 Complete structured DNNF

A *vtree* T for a set of variables X is a rooted tree where every non-leaf node has exactly two children and the leaves of T are in one-to-one correspondence with X. A *complete structured* DNNF (D, T, λ) is a DNNF D together with a vtree T for $\mathsf{var}(D)$ and a labeling λ of the nodes of T with sets of gates of D such that:

- If t is a leaf of T labeled with variable $x \in X$ then $\lambda(t)$ contains only input gates of D labeled with either x, $\neg x$.
- For every gate u of D, there exists a unique node t_u of T such that $u \in \lambda(t_u)$.
- There is no non-leaf node t of T such that $\lambda(t)$ contains an input gate of D.
- For every $\wedge$-gate u with inputs v_1, v_2, we have $t_{v_1} \neq t_{v_2}$.
- For every edge (u, v) of D:
  - Either v is an $\wedge$-gate, u is an $\vee$-gate or an input gate and t_u is the child of t_v.
  - Or v is an $\vee$-gate, u is an $\wedge$-gate and $t_u = t_v$.

Intuitively, T can be seen as a coarse structure of D, as depicted on Figure 2: We structure the gates of D into blocks $\lambda(t)$ that are associated to nodes t of T. Every such $\lambda(t)$ computes a 2DNF where every term has one input from $\lambda(t_1)$ and $\lambda(t_2)$, respectively, where t_1, t_2 are the children of t. In the following, when we do not directly deal with vtree and its labeling, we may refer to a complete structured DNNF (D, T, λ) by only mentioning the circuit D. It is then always understood that T and λ with the desired properties exist.

We note that there is a syntactic transformation of complete OBDD into complete structured d-DNNF. It proceeds iteratively from the sinks to the source introducing a gate g_v for every node v: for every sink, g_v is an input gate with the same label as the sink. For every other node v with label x, a 0-edge to u and a 1-edge to u', we introduce a subcircuit

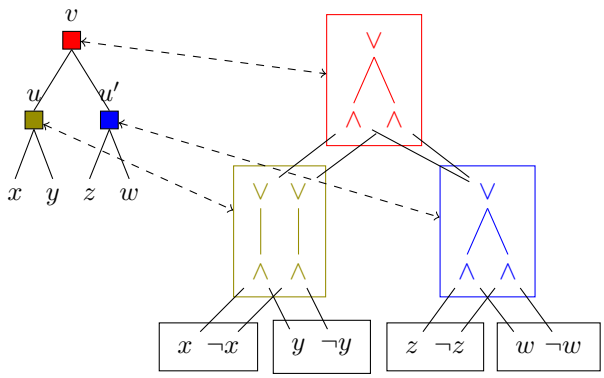


Figure 2 A vtree T and a complete structured DNNF (D, T, λ), where λ for the nodes v, u, w is represented with colors and dashed arrows.

computing $(g_u \wedge \neg x) \vee (g_{u'} \wedge x)$. It is easy to check that the resulting circuit is a structured complete d-DNNF computing the right function and whose vtree consists of a tree in which for every internal node one of the children is a leaf[1].

4.2 Width

We define the *width* of a complete structured DNNF (D, T, λ) as $\max_{t \in V(T)} |\{v \in \lambda(t) \mid v \text{ is an } \vee\text{-gate}\}|$. For example, the DNNF pictured on Figure 2 has width 2 since $\lambda(u)$ contains 2 $\vee$-gates and $\lambda(u')$ and $\lambda(v)$ contain less $\vee$-gates.

Note that for the width we do not take into account $\wedge$-gates. This is for several reasons: first, only considering $\vee$-gates simplifies some of the arguments later on and gives cleaner results and proofs. Moreover, it is not hard to see that when rewriting OBDD as DNNF as sketched above, the width of the original OBDD is exactly the width of the resulting circuit. The same is also true for the width of SDD [5], another important representation of Boolean functions [10]. Thus, width defined only on $\vee$-gates allows a tighter connection to the literature. Finally, the following observation shows that the number of $\wedge$-gates in a complete structured DNNF as we define it is highly connected to the width.

▶ **Observation 3.** *Let (D, T, λ) be a complete structured DNNF of width $w \geq 2$. We can in linear time in $|D|$ compute a complete structured DNNF (D', T, λ') of width w and equivalent to D such for every node t of T, we have $|\lambda'(t)| \leq (w^2 + w)$. Moreover, D' is of size at most $2(w + w^2)|\mathsf{var}(D)|$.*

Proof. For the first statement, note that by definition there are at most w $\vee$-gates in $\lambda(t)$. Now, the inputs of every $\wedge$-gate of $\lambda(t)$ are either $\vee$-gates or input gates in $\lambda(t_1)$ and $\lambda(t_2)$ where t_1, t_2 are the children of t in T. Thus, there are at most w^2 possible ways of connecting these $\wedge$-gates to their inputs. So if we eliminate $\wedge$-gates that have identical inputs and keep for every combination at most one of them, we get D' with the desired size bound on $\lambda'(t)$. However, we can neither naively compare the children of all $\wedge$-gates nor order the $\wedge$-gates by their children to eliminate $\wedge$-gates with identical inputs since both approaches would violate the linear time requirement.

[1] Note that strictly speaking the constructed circuit is not a complete structured d-DNNF as defined above because it contains constants as input gates. This slight complication will be taken care of in Lemma 4 below.

To avoid this slight complication, we proceed as follows: in a first step, we count the $\wedge$-gates in $\lambda(t)$. If there are at most w^2 of them, we satisfy the required upper bound, so we do nothing. Otherwise, we create an array of size w^2 indexed by the pairs of potential inputs of $\wedge$-gates in $\lambda(t)$. We initialize all cells to some null-value. Now we iterate over the $\wedge$-gates in $\lambda(t)$ and do the following for every such gate u: if the cell indexed by the children of u is empty, we store u in that cell and continue. If there is already a gate u' in the cell, we connect all gates that u feeds into to u' and delete u afterwards. It is easy to see that the resulting algorithm runs in linear time, computes a D' equivalent to D and satisfies the size bounds on $\lambda(t)$.

Since T is a tree where every node but the leaves has exactly 2 children, the number of nodes in T is at most $2|\mathsf{var}(D)|$. Now, because of $|\lambda'(t)| \leq w^2 + w$ for every t in T, the bound on $|D'|$ follows directly. ◀

We remark that complete structured DNNF as defined above are more restrictive than structured DNNF as defined in [21]. The definition of [21] only gives a condition on the way decomposable $\wedge$-gates can partition variables, following the vtree. However, it is possible to to extend the classical construction to turn any OBBD into a complete one to transform any structured DNNF in the sense of [21] into the form we define above with only a polynomial increase in size. However, even for OBDD this rewriting may increase the width arbitrarily at least when not changing the order [2]. Moreover, the construction for structured DNNF is rather tedious and complicated, so we will not follow this direction here.

4.3 Eliminating Constants

Our definition of complete structured DNNF does not allow constant inputs. This is in general not a problem as constants can be propagated in the circuits and thus eliminated. However, it is not directly clear how this propagation affects the width in our setting. Moreover, most of our algorithms are easier to describe by allowing constants. So let us spend some time to deal with constants in our setting. To this end, we introduce the notion of *extended vtrees*. An extended vtree T on a variable set X is defined as a vtree in which we allow some leaves to be unlabeled. Every variable of X must be the label of exactly one leaf still. A complete structured DNNF (D, T, λ) is defined as for an extended vtree with the additional requirement that for every unlabeled leaf ℓ of T, $\lambda(\ell)$ is a set of constant inputs of D.

We now show that we can remove the unlabeled leaves without increasing the width.

▶ **Lemma 4.** *There is a linear time algorithm that, given a complete structured DNNF (resp. d-DNNF) (D, T, λ) of width w where T is an extended vtree, computes a complete structured DNNF (resp. d-DNNF) (D', T', λ') of width w that is equivalent to D where T' is non-extended.*

Proof. Given an extended vtree T and a leaf ℓ, let $T \setminus \ell$ be the vtree obtained by removing the leaf ℓ of T and by merging the father and the sibling of ℓ in T. We first show that there is an algorithm that, given a complete structured DNNF (resp. d-DNNF) (D, T, λ) of width w and a non-labeled leaf ℓ of T, computes in linear time in $|\lambda(t)|$ an equivalent complete structured DNNF $(D', T \setminus \ell, \lambda')$ of width at most w. Iterating the construction and observing that every $\lambda(t)$ is treated only once, we get the claim of the lemma.

Let t be the father and t_s the sibling of ℓ in T. We let t' be the vertex of $T \setminus \ell$ obtained by merging t and t_s (see Figure 3). The idea of the transformation is depicted on Figures 4.

By definition, all gates of $\lambda(t)$ that are connected to gates in $\lambda(\ell)$ are $\wedge$-gates. We remove every $\wedge$-gate of $\lambda(t)$ connected to constant 0 as they are equivalent to 0 and are connected

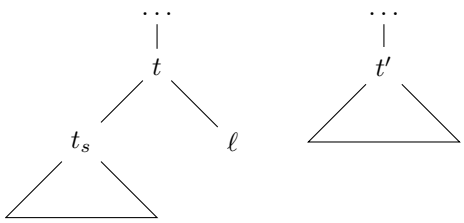


Figure 3 The trees T and $T \setminus \ell$ with notations.

to $\vee$-gates of $\lambda(t)$. We next deal with the $\wedge$-gates of $\lambda(t)$ connected to the constant 1. For every such gate v, we connect its other input to all outputs of v which by definition are all $\vee$-gates in $\lambda(t)$. This does not change the functions computed by the outputs of v and does not affect the determinism of the DNNF.

If t_s is not a leaf of T, then $\vee$-gates of $\lambda(t)$ are connected to $\vee$-gates of $\lambda(t_s)$. Without changing the function computed nor determinism, we can connect the $\vee$-gates of $\lambda(t)$ directly to the input of its inputs and thus remove every $\vee$-gate of $\lambda(t_s)$. Now the circuit has the following form: $\vee$-gates of $\lambda(t)$ are connected to $\wedge$-gates of $\lambda(t_s)$. We thus define $\lambda'(t')$ as the remaining $\vee$-gates of $\lambda(t)$ and $\wedge$-gates of $\lambda(t_s)$ and get a complete structured DNNF for $T \setminus \ell$. The number of $\vee$-gates in $\lambda(t')$ is less than in $\lambda(t)$ so the width has not increased.

If t_s is a leaf labeled by a variable x, then every $\vee$-gate g in $\lambda(t)$ is connected to input gates in x and thus they compute either x, or $\neg x$ or $\neg x \vee x$. In the former two cases, we simply substitute g by x or $\neg x$ respectively. If g computes $\neg x \vee x$, then do for every $\wedge$-gate g' that has g as an input the following: create a clone g'' of g', i.e., a new $\wedge$-gate that has the same inputs and outputs as g'. Then substitute the input g of g' by $\neg x$ and by x for g''. Since all gates that have g' as an input are $\vee$-gates, this does not change the function computed in these values. Finally, delete g. Doing this for all g, we delete all $\wedge$-gates in $\lambda(t)$. Now setting $\lambda'(t)$ to contain the newly introduced input gates completes the construction. Obviously, the number of $\vee$-gates in $\lambda'(t^*)$ is never bigger than that in $\lambda(t^*)$.

Finally, if t_s is an unlabeled leaf, then all $\vee$-gates in $\lambda(t)$ compute constants. Substituting them by those constants and defining $\lambda'(T')$ in the obvious way, completes the the proof. ◀

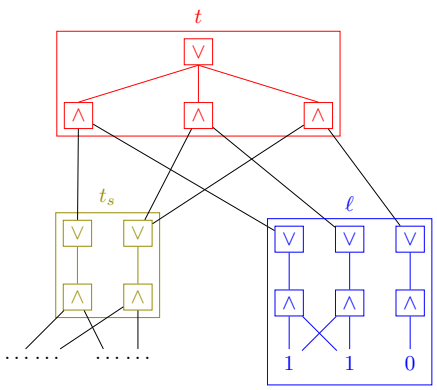
4.4 Existential quantification on bounded width d-DNNF

In this section, we give an algorithm that allows us to quantify variables in d-DNNF. The main result is the following.

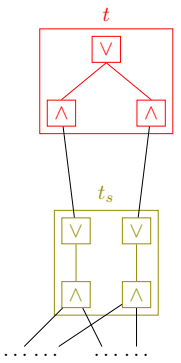
▶ **Theorem 5.** *There is an algorithm that, given a complete structured DNNF (D, T, λ) of width w and $Z \subseteq \mathsf{var}(D)$, computes in time $2^{O(w)}|D|$ a complete structured d-DNNF (D', T', λ') of width at most 2^w having a designated gate computing $\exists Z\ D$ and another designated gate computing $\neg\exists Z\ D$.*

In the rest of this section, we will prove Theorem 5. Let (D, T, λ) be a complete structured DNNF. Let $X = \mathsf{var}(D)$, the variables $Z \subseteq X$ those that we will quantify and w the width of D.

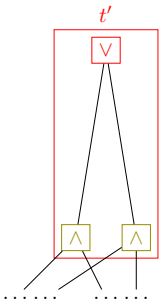
Given a node t of T, let $\mathsf{var}(t)$ be the set of variables which are at the leaves of the subtree of T rooted in t. We define $\mathsf{forgot}(t) := Z \cap \mathsf{var}(t)$ and $\mathsf{kept}(t) := \mathsf{var}(t) \setminus \mathsf{forgot}(t)$. Intuitively, $\mathsf{forgot}(t)$ contains the set of variables that are quantified away below t while $\mathsf{kept}(t)$ contains the remaining variables under t. Let D_v for a gate v denote the sub-DNNF of D rooted in v.



(a) Original circuit.



(b) Propagating constants. The rightmost part evaluates to 0 and disappears.



(c) Merging t_s and t and cleaning disconnected gates.

Figure 4 Illustration of the transformation of Lemma 4. The constants are propagated in the first step to remove gates in bag ℓ. Then the bags of t_s and t are merged without changing the computed function.

Shapes. A key notion for our algorithm will be what we call *shapes*. Let t be a node of T and let O_t be the set of $\vee$-gates of D labeling t. An assignment $\tau : \mathsf{kept}(t) \to \{0,1\}$ is of shape $S \subseteq O_t$ if and only if

$$S = \{s \in O_t \mid \exists \sigma : \mathsf{forgot}(t) \to \{0,1\}, \tau \cup \sigma \models D_s\}.$$

We denote by $\mathsf{Shape}_t \subseteq 2^{O_t}$ the set of shapes of a node t. Observe that $|\mathsf{Shape}_t| \leq 2^{|O_t|} \leq 2^w$ since $|O_t| \leq w$ by definition.

The key observation is that Shape_t can be inductively computed. Indeed, let t be a node of T with children t_1, t_2 and let $S_1 \in \mathsf{Shape}_{t_1}$, $S_2 \in \mathsf{Shape}_{t_2}$. We define $S_1 \bowtie S_2 \subseteq O_t$ to be the set of gates $s \in O_t$ that evaluate to 1 once we replace every gate in S_1 and S_2 by 1 and every gate in $O_{t_1} \setminus S_1$ and $O_{t_2} \setminus S_2$ by 0.

▶ **Lemma 6.** *Let t be node of T with children t_1, t_2. Let $\tau_1 : \mathsf{kept}(t_1) \to \{0,1\}$ be of shape S_1 and $\tau_2 : \mathsf{kept}(t_2) \to \{0,1\}$ be of shape S_2. Then $\tau = \tau_1 \cup \tau_2$ is of shape $S_1 \bowtie S_2$.*

Proof. Let S be the shape of τ. We first prove $S \subseteq S_1 \bowtie S_2$. So let $s \in S$. Since τ is of shape S, there exists $\sigma : \mathsf{forgot}(t) \to \{0,1\}$ such that $\tau \cup \sigma$ satisfies D_s. Since s is an $\vee$-gate, there must be an input gate s' of s such that $\tau \cup \sigma$ satisfies s'. By definition, s' is an $\wedge$-gate with two children $s_1 \in O_{t_1}$ and $s_2 \in O_{t_2}$. Thus D_{s_1} is satisfied by $(\tau \cup \sigma)|_{\mathsf{var}(t_1)} = \tau_1 \cup \sigma|_{\mathsf{var}(t_1)}$. Consequently, $s_1 \in S_1$ since S_1 is the shape of τ_1. Similarly $s_2 \in S_2$. Thus, in the construction of $S_1 \bowtie S_2$, both s_1 and s_2 are replaced by 1, so s evaluates to 1, that is, $s \in S_1 \bowtie S_2$.

We now show that $S_1 \bowtie S_2 \subseteq S$. So let $s \in S_1 \bowtie S_2$. Then, in the construction of $S_1 \bowtie S_2$, there must be an input s' of s that is satisfied. Then s' is an $\wedge$-gate with children $s_1 \in O_{t_1}$, $s_2 \in O_{t_2}$ evaluating to 1. It follows that s_1 and s_2 have been replaced by 1 in the construction of $S_1 \bowtie S_2$. Now by definition of S_1, there exists $\sigma_1 \colon \mathsf{forgot}(t_1) \to \{0,1\}$ such that $\tau_1 \cup \sigma_1$ satisfies D_{s_1} and $\sigma_2 \colon \mathsf{forgot}(t_2) \to \{0,1\}$ such that $\tau_2 \cup \sigma_2$ satisfies D_{s_2}. Thus, $(\tau_1 \cup \sigma_1) \cup (\tau_2 \cup \sigma_2) = \tau \cup (\sigma_1 \cup \sigma_2)$ is well-defined because σ_1 and σ_2 do not share any variables because s' is decomposable. Moreover, $\tau \cup (\sigma_1 \cup \sigma_2)$ satisfies D_s and thus we have $s \in S$. ◀

Constructing the projected d-DNNF. We now inductively construct a d-DNNF D' computing $\exists Z\ D$ and of width at most 2^w. The extended vtree T' for D' is obtained from T by removing the labels of the leaves corresponding to variables in Z. One can then apply Lemma 4 to obtain a vtree. We inductively construct for every node t of T and $S \in \mathsf{Shape}_t$, an $\vee$-gate $v_t(S)$ in D' such that $D'_{v_t(S)}$ accepts exactly the assignments of shape S and we will define $\lambda'(t) = \bigcup_{S \in \mathsf{Shape}_t} v_t(S)$.

If t is a leaf of T, then $\mathsf{kept}(t)$ has at most one variable, thus we have at most two assignments of the form $\mathsf{kept}(t) \to \{0,1\}$. We can thus try all possible assignments to compute Shape_t explicitly and $v_t(S)$ will either be a literal or a constant for each $S \in \mathsf{Shape}_t$. We put $v_t(S)$ in $\lambda'(t')$ where t' is the leaf of T' corresponding to t. It is clear that if t' is labeled with variable x then $v_t(S)$ is a literal labeled by x or by $\neg x$. If t' is unlabeled, then it corresponds to a leaf t of T labeled with a variable of Z. Thus $v_t(S)$ is a constant input so the conditions of structuredness are respected.

Now let t be a node of T with children t_1, t_2 and assume that we have constructed $v_{t_1}(S_1)$ for every $S_1 \in \mathsf{Shape}_{t_1}$ and $v_{t_2}(S_2)$ for every $S_2 \in \mathsf{Shape}_{t_2}$. We define $v_t(S)$ as:

$$\bigvee_{S_1,S_2 : S = S_1 \bowtie S_2} v_{t_1}(S_1) \wedge v_{t_2}(S_2)$$

where S_1, S_2 run over Shape_{t_1} and Shape_{t_2} respectively.

First of all, observe that the $\wedge$-gates above are decomposable since $D'_{v_{t_1}(S_1)}$ is on variables $\mathsf{kept}(t_1)$ which is disjoint from $\mathsf{kept}(t_2)$, the variables of $D'_{v_{t_2}(S_2)}$.

Moreover, observe that the disjunction is deterministic. Indeed, by induction, τ satisfies the term $v_{t_1}(S_1) \wedge v_{t_2}(S_2)$ if and only if $\tau|_{\mathsf{var}(t_1)}$ is of shape S_1 and $\tau|_{\mathsf{var}(t_2)}$ is of shape S_2. Since an assignment has exactly one shape, we know that τ cannot satisfy another term of the disjunction.

Finally, we have to show that $v_t(S)$ indeed computes the assignments of shape S. This is a consequence of Lemma 6. Indeed, if τ is of shape S then let S_1, S_2 be the shapes of $\tau|_{\mathsf{var}(t_1)}$ and $\tau|_{\mathsf{var}(t_2)}$ respectively. By Lemma 6, $S = S_1 \bowtie S_2$ and then $\tau \models v_{t_1}(S_1) \wedge v_{t_2}(S_2)$, and then, $\tau \models v_t(S)$.

Now, if $\tau \models v_{t_1}(S_1) \wedge v_{t_2}(S_2)$ for some S_1 and S_2 in the disjunction, then we have by induction that $\tau|_{\mathsf{var}(t_1)}$ and $\tau|_{\mathsf{var}(t_2)}$ are of shape S_1 and S_2 respectively. By Lemma 6, τ is of shape $S_1 \bowtie S_2 = S$.

Let t' be the node of T' corresponding to t. We put all gates needed to compute $v_t(S)$ in $\lambda'(t')$ for every S. This has the desired form: a level of $\vee$-gate, followed by a level of $\wedge$-gate connected to $\vee$-gates in $\lambda'(t'_1)$ and $\lambda'(t'_2)$. By construction, the width of the d-DNNF constructed so far is $\max_t |\mathsf{Shape}_t| \leq 2^w$.

Now assume that we have a d-DNNF D_0 with a gate $v_t(S)$ for every t and every $S \in \mathsf{Shape}_t$ computing the assignments of shape τ. Let r be the root of T. We assume w.l.o.g. that the root of D is a single $\vee$-gate r_o connected to every $\wedge$-gate labeled by r. Then $v_r(\{r_o\})$ accepts exactly $\exists Z D$ and $v_r(\emptyset)$ accepts $\neg\exists Z\ D$.

5 Algorithms for graph width measures

In this section, we will show how we can use the result of Section 4 in combination with known compilation algorithms to show tractability results for QBF with restricted underlying graph structure and bounded quantifier alternation. This generalizes the results of [8, 13, 14].

We use the following result which can be verified by careful analysis of the construction in [9, Section 3]; for the convenience of the reader we give an independent proof in the long version of this paper [7].

▶ **Theorem 7.** *There is an algorithm that, given a CNF F of primal treewidth k, computes in time $2^{O(k)}|F|$ a complete structured d-DNNF D of width $2^{O(k)}$ equivalent to F.*

We lift Theorem 7 to incidence treewidth by using the following result from [17].

▶ **Proposition 8.** *There is an algorithm that, given a CNF-formula F of incidence treewidth k, computes in time $O(2^k|F|)$ a 3CNF-formula F' of primal treewidth $O(k)$ and a subset Z of variables such that $F \equiv \exists Z F'$.*

▶ **Corollary 9.** *There is an algorithm that, given a CNF F formula of incidence treewidth k, computes in time $2^{O(k)}|F|$ a complete structured d-DNNF D of width $2^{O(k)}$ and a subset Z of variables such that $F \equiv \exists Z D$.*

Note that in [3] there is another algorithm that compiles bounded incidence treewidth into d-DNNF without introducing new variables that have to be projected away to get the original function. The disadvantage of this algorithm though is that the time to compile is quadratic in the size of F. Since we are mostly interested in QBF in which the last quantifier block is existential, adding some more existential variables does not hurt our approach, so we opted for the linear time algorithm we get from Corollary 9.

Now using Theorem 5 iteratively, we directly get the following result.

▶ **Theorem 10.** *There is an algorithm that, given a QBF F with free variables, ℓ quantifier blocks and of incidence treewidth k, computes in time $\exp^{\ell+1}(O(k))|F|$ a complete structured d-DNNF of width $\exp^{\ell+1}(O(k))$ accepting exactly the models of F.*

Proof. Let $F = Q_1 X_1 \dots \exists X_\ell G$. We use Corollary 9 to construct a structured DNNF D of width $2^{O(k)}$ such that $G \equiv \exists Z D$, that is $F \equiv Q_1 X_1 \dots \exists (X_\ell \cup Z) D$. Assume first that $Q_1 = \forall$. Then using the fact that for every formula F' we have $\forall X F' \equiv \neg\exists\neg F'$, we can rewrite this into $F \equiv \neg\exists X_1(\neg\exists X_2(\neg \dots (\neg\exists(X_\ell \cup Z)G)\dots))$. We now use Theorem 5 to iteratively from the right eliminate all blocks $\neg\exists X_i$. The result is a complete structured d-DNNF accepting exactly the models of F. If $Q_1 = \exists$, we apply essentially the same construction with the only difference that there is no negation in front of the formula which can be also dealt with using Theorem 5. Each application of Theorem 5 blows the width of the circuit by a single exponential, resulting in the stated complexity. ◀

As an application of Theorem 10, we give a result on model counting.

▶ **Corollary 11.** *There is an algorithm that, given a QBF F with free variables, ℓ quantifier blocks and of incidence treewidth k, computes in time $\exp^{\ell+1}(O(k))|F|$ the number of models of F.*

We remark that Corollary 11 generalizes several results from the literature. On the one hand, it generalizes the main result of [8] from decision to counting, from primal treewidth to incidence treewidth and gives more concrete runtime bounds[2]. On the other hand, it generalizes the counting result of [14] from projected model counting, i.e., QBF formulas with free variables and just one existential quantifier block, to any constant number of quantifier alternations. Moreover, our runtime is linear in $|F|$ in contrast to the runtime of [14] which is quadratic.

As a generalization of Theorem 10, let us remark that there are compilation algorithms for graph measures beyond treewidth. For example, it is known that CNF formulas of *bounded signed cliquewidth* [15] can be compiled efficiently [3]. More exactly, there is an algorithm

[2] We remark that the latter two points have already been made recently in [17].

that compiles a CNF formula F of signed incidence cliquewidth k in time $2^{O(k)}|F|^2$ into a structured d-DNNF of size $2^{O(k)}|F|$. We will not formally introduce signed incidence cliquewidth here but refer the reader to [15, 6]. Inspecting the proof of [3], one can observe that the algorithm construct a complete structured d-DNNF of width at most $2^{O(k)}$ which as above yields the following result.

▶ **Theorem 12.** *There is an algorithm that, given a QBF F with free variables, with ℓ quantifier blocks and of signed incidence cliquewidth k, computes in time $\exp^{\ell+1}(O(k))|F| + 2^{O(k)}|F|^2$ a complete structured d-DNNF of width $\exp^{\ell+1}(O(k))$ accepting exactly the models of F.*

With Theorem 12 it is now an easy exercise to derive generalizations of [8, 14, 15].

In the light of the above positive results one may wonder if our approach can be pushed to more general graph width measures that have been studied for propositional satisfiability like for example modular treewidth [20] or (unsigned) cliquewidth [26]. Using the results of [18], we can answer this question negatively in two different ways: on the one hand, QBF of bounded modular cliquewidth and bounded incidence cliquewidth with one quantifier alternation is NP-hard, so under standard assumptions there is no version of Theorem 10 and thus also not of Corollary 9 for cliquewidth. On the other hand, analyzing the proofs of [18], one sees that in fact there it is shown that for every CNF formula F there is a bounded modular treewidth and bounded incidence cliquewidth formula F' and a set Z of variables such that $F \equiv \exists Z F'$. Since it is known that there are CNF formulas that do not have subexponential size DNNF [4], it follows that there are such formulas F' such that every DNNF representation of $\exists Z F'$ has exponential width. This unconditionally rules out a version of Corollary 9 and Theorem 12 for modular treewidth or cliquewidth.

6 Lower Bounds

In this section, we will show that all restrictions we put onto the DNNF in Theorem 5 are necessary.

6.1 The definition of width

Width of an OBDD is usually defined on *complete* OBDD. There is however another way of defining width for OBDD by just counting the number of nodes that are labeled with the same variable which for a non-complete OBDD might be far smaller. Let us call this notion *weak width*. We will show that width in Theorem 5 cannot be substituted by weak width.

▶ **Lemma 13.** *For every n there is an OBDD D_n in $O(n)$ variables of weak width 3 and a set $Z \subseteq \mathsf{var}(D_n)$ such that $\neg\exists Z\ D_n$ does not have an DNNF of size $2^{o(n)}$.*

Proof. Let S_i for $i \in \mathbb{N}$ denote the term $\neg z_i \wedge \left(\bigwedge_{j\in[i-1]} z_j\right)$. For a CNF $F = C_1 \wedge \ldots \wedge C_m$ we then define the function

$$F' = \bigvee_{i=1}^{m} S_i \wedge C_i.$$

It is easy to see that by testing $z_1, \ldots, z_m$ successively and branching a small OBDD for C_i at each 0-output of the decision node testing z_i as depicted on Figure 5, one can construct an OBDD of size $O(|F|)$ computing F'. If every variable appears in at most three clauses of F, then this OBDD has weak width 3 since a variable x is only tested for clauses where it appears.

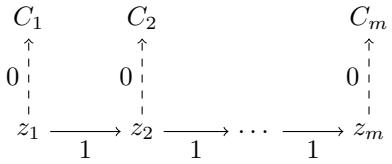


Figure 5 Structure of an OBDD for F'.

Note that $\forall Z\ F' \equiv F$. Since there are CNF formulas in which every variable appears in at most three clauses which do not have subexponential size DNNF [4]. It follows that for such F the function $\forall Z\ F'$ has exponential size. Now remarking that $\forall Z\ F' \equiv \neg\exists Z\ (\neg F')$ and that $\neg F'$ has an OBDD of weak width 3 as well, completes the proof. ◀

6.2 Structuredness

One of the properties required for Theorem 5 is that we need the input to be structured. Since structuredness is quite restrictive, see e.g. [22], it would be preferable to get rid of it to show similar results. Unfortunately, there is no such result as the following lemma shows.

To formulate our results, we need a definition of width for FBDD. This is because width as we have defined it before depends on the vtree of the DNNF which we do not have in the case without structuredness. To define width for the unstructured case, we consider layered FBDD: an FBDD F is called *layered* if the nodes of F can be partitioned into sets $L_1, \dots, L_s$ such that for every edge uv in F there is an $i \in [s]$ such that $u \in L_i$ and $v \in L_{i+1}$. The *width* of F is then defined as $\max\{|L_i| \mid i \in [s]\}$.

▶ **Lemma 14.** *For every n there is a function f_n in $O(n^2)$ variables with an FBDD representation of size $O(n^2)$ and width $O(1)$ such that there is a variable x of f_n such that every deterministic DNNF for $\exists x\ f_n$ has size $2^{\Omega(n)}$.*

Proof. We use a function introduced by Sauerhoff [25]: let $g : \{0,1\}^n \to \{0,1\}$ be the function that evaluates to 1 if and only if the sum of its inputs is divisible by 3. For a $n \times n$-matrix X with inputs $x_{ij} \in \{0,1\}$, we define

$$R_n(X) := \bigoplus_{i=1}^{n} g(x_{i1}, x_{i2}, \dots, x_{in})$$

where $\oplus$ denotes addition modulo 2 and define $C_n(X) := R_n(X^T)$ where X^T is the transpose of X. Then $S_n(X) := R_n(X) \vee C_n(X)$.

Note that, ordering the variables of X by rows, resp. columns, R_n and C_n both have OBDD of width $O(1)$ and size $O(n^2)$. Now let $S'_n = (x \wedge R_n) \vee (\neg x \wedge C_n)$. Then S'_n clearly has an FBDD of size $O(n^2)$ and width $O(1)$: decide on x first and then depending on its value follow the OBDD for R_n or C_n.

But $\exists x S'_n(X) = S_n(X)$ which completes the proof since S_n is known to require size $2^{\Omega(n)}$ for deterministic DNNF [4]. ◀

7 Conclusion

We have introduced a new notion of width of complete structured d-DNNF and shown that using it, in combination with a rather simple quantifier elimination result and known compilation results, one can show several new tractability results around QBF. In contrast

to earlier results that solved similar problems in one pass of dynamic programming [8, 14], our approach is iterative and only considers one quantifier block at the same time which in our opinion greatly simplifies the argument. Moreover, factoring out the initial compilation phase allowed us to generalize known results for treewidth to signed cliquewidth essentially for free.

We feel that the notion of width we introduced for complete structured d-DNNF is an interesting notion, independent of the results on quantifier elimination here, and deserves closer examination. In the long version of this paper [7] we initiate this by giving results on standard transformation that are generally considered in knowledge compilation. We are certain that beyond this our width notion will have more uses in the future.

It might be interesting to apply our approach to non-monotone reasoning problems from artificial intelligence. There is a wealth of such problems that have been considered under treewidth reductions, see e.g. the problems in [16], but for more general width measures far less is known, some exceptions being [12, 1]. Proving more such results might be possible by reductions to QBF as in [17] but it would be necessary to understand if those reductions maintain the considered width measure. Alternatively, one could try to mimic our approach of compiling and then refining the solution space iteratively for the individual problems which might be easier than performing direct dynamic programming.

References

1 Bernhard Bliem, Sebastian Ordyniak, and Stefan Woltran. Clique-Width and Directed Width Measures for Answer-Set Programming. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1105–1113. IOS Press, 2016. doi:10.3233/978-1-61499-672-9-1105.

2 Beate Bollig and Ingo Wegener. Asymptotically Optimal Bounds for OBDDs and the Solution of Some Basic OBDD Problems. *J. Comput. Syst. Sci.*, 61(3):558–579, 2000. doi:10.1006/jcss.2000.1733.

3 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On Compiling CNFs into Structured Deterministic DNNFs. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference*, volume 9340 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2015.

4 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge Compilation Meets Communication Complexity. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1008–1014. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/147.

5 Simone Bova and Stefan Szeider. Circuit Treewidth, Sentential Decision, and Query Compilation. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 233–246. ACM, 2017. doi:10.1145/3034786.3034787.

6 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding Model Counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science*, pages 143–156, 2015.

7 Florent Capelli and Stefan Mengel. Knowledge Compilation, Width and Quantification. *CoRR*, abs/1807.04263, 2018. arXiv:1807.04263.

8 Hubie Chen. Quantified Constraint Satisfaction and Bounded Treewidth. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 161–165, 2004.

9 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.

10 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 819–826. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-143.

11 Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.

12 Wolfgang Dvorák, Stefan Szeider, and Stefan Woltran. Reasoning in Argumentation Frameworks of Bounded Clique-Width. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo Ricardo Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010.*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 219–230. IOS Press, 2010. doi:10.3233/978-1-60750-619-5-219.

13 Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. Treewidth in Verification: Local vs. Global. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*, pages 489–503. Springer, 2005. doi:10.1007/11591191_34.

14 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Exploiting Treewidth for Projected Model Counting and Its Limits. 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 165–184. Springer, 2018. doi:10.1007/978-3-319-94144-8_11.

15 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. doi:10.1016/j.dam.2006.06.020.

16 Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010. doi:10.1016/j.artint.2009.10.003.

17 Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to courcelle's theorem. *CoRR*, abs/1805.08456, 2018. accepted for SAT'18. arXiv:1805.08456.

18 Michael Lampis and Valia Mitsou. Treewidth with a Quantifier Alternation Revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

19 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. *Algorithmica*, 76(1):168–194, 2016.

20 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. *Algorithmica*, 76(1):168–194, 2016. doi:10.1007/s00453-015-0030-x.

21 Knot Pipatsrisawat and Adnan Darwiche. New Compilation Languages Based on Structured Decomposability. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 517–522. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-082.php.

22 Thammanit Pipatsrisawat and Adnan Darwiche. A Lower Bound on the Size of Decomposable Negation Normal Form. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1856.

23 S. Hortemo Sæther, J.A. Telle, and M. Vatshelle. Solving MaxSAT and #SAT on Structured CNF Formulas. In *Theory and Applications of Satisfiability Testing*, pages 16–31, 2014.

24 M. Samer and S. Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50–64, 2010.

25 Martin Sauerhoff. Approximation of boolean functions by combinatorial rectangles. *Theor. Comput. Sci.*, 1-3(301):45–78, 2003. doi:10.1016/S0304-3975(02)00568-6.

26 Friedrich Slivovsky and Stefan Szeider. Model Counting for Formulas of Bounded Clique-Width. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 677–687. Springer, 2013. doi:10.1007/978-3-642-45030-3_63.

27 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability, 6th International Conference*, volume 2919 of *LNCS*, pages 188–202. Springer, 2004.

A Tight Extremal Bound on the Lovász Cactus Number in Planar Graphs

Parinya Chalermsook
Aalto University, Espoo, Finland
parinya.chalermsook@aalto.fi

Andreas Schmid
Max Planck Institute for Informatics, Saarbrücken, Germany
aschmid@mpi-inf.mpg.de

Sumedha Uniyal
Aalto University, Espoo, Finland
sumedha.uniyal@aalto.fi

Abstract

A *cactus* graph is a graph in which any two cycles are edge-disjoint. We present a constructive proof of the fact that any plane graph G contains a cactus subgraph C where C contains at least a $\frac{1}{6}$ fraction of the triangular faces of G. We also show that this ratio cannot be improved by showing a tight lower bound. Together with an algorithm for linear matroid parity, our bound implies two approximation algorithms for computing "dense planar structures" inside any graph: (i) A $\frac{1}{6}$ approximation algorithm for, given any graph G, finding a planar subgraph with a maximum number of triangular faces; this improves upon the previous $\frac{1}{11}$-approximation; (ii) An alternate (and arguably more illustrative) proof of the $\frac{4}{9}$ approximation algorithm for finding a planar subgraph with a maximum number of edges.

Our bound is obtained by analyzing a natural local search strategy and heavily exploiting the exchange arguments. Therefore, this suggests the power of local search in handling problems of this kind.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Graph Drawing, Matroid Matching, Maximum Planar Subgraph, Local Search Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.19

Related Version Full Version: `https://arxiv.org/abs/1804.03485`.

Funding *Parinya Chalermsook*: Part of this work was done while PC and AS were visiting the Simons Institute for the Theory of Computing. It was partially supported by the DIMACS/Simons Collaboration on Bridging Continuous and Discrete Optimization through NSF grant #CCF-1740425. Parinya has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557) and by Academy of Finland Research Fellows, under grant number 310415 and 314284.
Sumedha Uniyal: Partially supported by Academy of Finland under the grant agreement number 314284.

1 Introduction

Linear matroid parity (introduced in various equivalent forms [21, 18, 15]) is a key concept in combinatorial optimization that includes many important optimization problems as special cases; probably the most well-known example is the *maximum matching problem*. The polynomial-time computability of linear matroid parity made it a popular choice as an algorithmic tool for handling both theoretical and practical optimization problems. An

important special case of linear matroid parity, the graphic matroid parity problem, is often explained in the language of *cacti* (see e.g. [9]), a graph in which any two cycles must be edge-disjoint. In 1980, Lovász [21] initiated the study of $\beta(G)$ (sometimes referred to as the *cactus number* of G), the maximum value of the number of triangles in a cactus subgraph of G, and showed that it generalizes maximum matching and can be reduced to linear matroid parity, therefore implying that $\beta(G)$ is polynomial-time computable[1][2].

Cactus graphs arise naturally in many applications[3]; perhaps the most relevant example in the context of approximation algorithms is the Maximum Planar Subgraph (MPS) problem: Given an input graph, find a planar subgraph with a maximum number of edges. Notice that, since any planar graph with n vertices has at most $3n-6$ edges, outputting a spanning tree with $n-1$ edges immediately gives a $\frac{1}{3}$-approximation algorithm. Generalizing the idea of finding spanning trees, one would like to look for a planar graph H, denser than a spanning tree, and at the same time efficiently computable. Calinescu et al. [3] showed that a cactus subgraph with a maximum number of triangles (which is efficiently computable via matroid parity algorithms) could be used to construct a $\frac{4}{9}$-approximation for MPS.

The $\frac{4}{9}$-approximation for MPS was achieved through an extremal bound of $\beta(G)$ when G is a plane graph. In particular, it was proven that $\beta(G) \geq \frac{1}{3}(n-2-t(G))$, where $n = |V(G)|$ and $t(G) = (3n-6) - |E(G)|$ (i.e. the number of edges missing for G to be a triangulated plane graph).

1.1 Our Results

In this work, we are interested in further studying the extremal properties of $\beta(G)$ and exhibit stronger algorithmic implications. Our main result is summarized in the following theorem.

▶ **Theorem 1.** *Let G be a plane graph. Then $\beta(G) \geq \frac{1}{6} f_3(G)$ where $f_3(G)$ denotes the number of triangular faces in G. Moreover, a natural local search 2-swap algorithm achieves this bound.*

It is not hard to see that $f_3(G) \geq 2n-4-2t(G)$ where $t(G)$ denotes the number of edges missing for G to be a triangulated plane graph. Therefore, we obtain the main result of [3] immediately.

▶ **Corollary 2.** *$\beta(G) \geq \frac{1}{3}(n-2-t(G))$. Hence, the matroid parity algorithm gives a $\frac{4}{9}$-approximation for MPS.*

Besides implying the MPS result, we exhibit further implications of our bound. Recently in [7], the authors introduced *Maximum Planar Triangles (MPT)*, where the goal is to find a plane subgraph with a maximum number of triangular faces. It was shown that an approximation algorithm for MPT naturally translates into one for MPS, where a $\frac{1}{6}$ approximate MPT solution could be turned into a $\frac{4}{9}$ approximate MPS solution. However, the authors only managed to show a $\frac{1}{11}$ approximation for MPT.

Although the only change from MPS to MPT lies in the objective of maximizing the number of triangular faces instead of edges, the MPT objective seems much harder to handle, for instance, the extremal bound provided in [3] is not sufficient to derive any approximation algorithm for MPT.

---

[1] There are many efficient algorithms for matroid parity (both randomized and deterministic), e.g. [9, 22, 24, 12].

[2] When we study $\beta(G)$, notice that a cactus subgraph that achieves the maximum value of $\beta(G)$ would only need to have cycles of length three (triangles). Such cacti are called *triangular cacti*.

[3] See for instance the wikipedia page `https://en.wikipedia.org/wiki/Cactus_graph`.

Theorem 1 therefore implies the following result for MPT.

▶ **Corollary 3.** *A matroid parity algorithm gives a $\frac{1}{6}$ approximation algorithm for MPT.*

Our conceptual contributions are the following:

1. Our result further highlights the extremal role of the cactus number in finding a dense planar structure, as illustrated by the fact that our bound on $\beta(G)$ is more "robust" to the change of objectives from MPS to MPT. It allows us to reach the limit of approximation algorithms that matroid parity provides for both MPS and MPT.
2. Our work implies that local search arguments alone are sufficient to "almost" reach the best known approximation results for both MPS and MPT in the following sense: Matroid parity admits a PTAS via local search [19, 2]. Therefore, combining this with our bound implies that local search arguments are sufficient to get us to a $\frac{4}{9}+\epsilon$ approximation for MPS and $\frac{1}{6}+\epsilon$ approximation for MPT. Therefore, this suggests that local search might be a promising candidate for such problems.
3. Finally, in some ways, our work can be seen as an effort to open up all the black boxes used in MPS algorithms with the hope of learning algorithmic insights that are crucial for making progress on this kind of problems. In more detail, there are two main "black boxes" hidden in the MPS result: (i) The use of Lovász min-max cactus formula in deriving the bound $\beta(G) \geq \frac{1}{3}(n-2-t(G))$, and (ii) the use of a matroid parity algorithm as a blackbox in computing $\beta(G)$. Our bound for $\beta(G)$ is now purely combinatorial (and even constructive) and manages to by-pass (i).

Related work. On the hardness of approximation side, MPS is known to be APX-hard [3], while MPT is only known to be NP-hard [7]. In combinatorial optimization, there are a number of problems closely related to MPS and MPT. For instance, finding a maximum series-parallel subgraph [5] or a maximum outer-planar graph [3], as well as the weighted variant of these problems [4]; these are the problems whose objectives are to maximize the number of edges.

Perhaps the most famous extremal bound in the context of cactus is the min-max formula of Lovász [21] and a follow-up formula that is more illustrative in the context of cactus [25]. All these formulas generalize the Tutte-Berge formula [1, 26] that has been used extensively both in research and curriculum.

Another related set of problems has the objectives of maximizing the number of vertices, instead of edges. In particular, in the maximum induced planar subgraph (i.e. given graph G, one aims at finding a set of nodes $S \subseteq V(G)$ such that $G[S]$ is planar, while maximizing $|S|$.) This variant has been studied under a more generic name, called *maximum subgraph with hereditary property* [23, 20, 13]. This variant is unfortunately much harder to approximate: $\tilde{\Omega}(|V(G)|)$[4] hard to approximate [14, 17]; in fact, the problems in this family do not even admit any FPT approximation algorithm [6], assuming the *gap exponential time hypothesis* (Gap-ETH).

1.2 Overview of Techniques

We give a high-level overview of our techniques. The description in this section assumes certain familiarity with how standard local search analysis is often done.

[4] The term $\tilde{\Omega}$ hides asymptotically smaller factors.

Our algorithm works as follows. Let G be an input plane graph, and let $\mathcal{C}$ be a cactus subgraph of G whose triangles correspond to triangular faces of G. The local search operation, t-swap, is done as follows: As long as there is a collection $X \subseteq \mathcal{C}$ of $\ell : \ell \leq t$ edge-disjoint triangles and Y such that $(\mathcal{C} \setminus X) \cup Y$ contains more triangular faces of G than $\mathcal{C}$ and it remains a cactus, we perform such an improvement step. A cactus subgraph is called locally t-swap optimal, if it can not be improved by a t-swap operation. Remark that the triangles chosen by our local search are only those which are triangular faces in the input graph G (we assume that the drawing of G is fixed.)

Our analysis is highly technical, although the basic idea is very simple and intuitive. We give a high-level overview of the analysis. We remark that this description is overly simplified, but it sufficiently captures the crux of our arguments. Let $\mathcal{C}$ be the solution obtained by the local search 2-swap algorithm. We argue that the number of triangles in $\mathcal{C}$ is at least $f_3(G)/6$. We remark that the 2-swap is required, as we are aware of a bad example H for which the 1-swap local search only achieves a bound of $(\frac{1}{7} + o(1))f_3(H)$. For simplicity, let us assume that $\mathcal{C}$ has only one non-singleton component. Let $S \subseteq V(G)$ be the vertices in such a connected component.

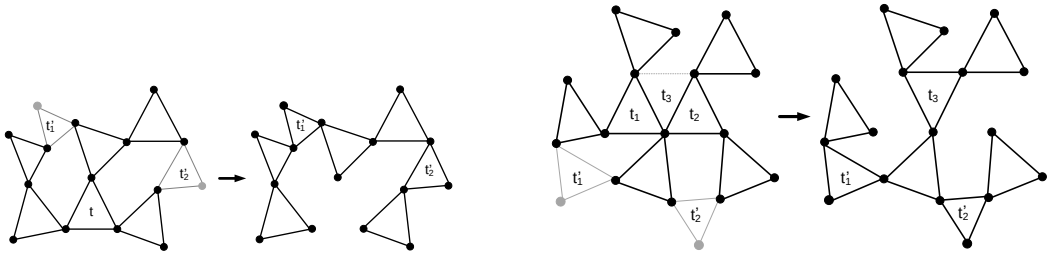
Let t be a triangle in $\mathcal{C}$. Notice that removing the three edges of t from $\mathcal{C}$ breaks the cactus into at most three components, say $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$ that are pairwise vertex-disjoint, i.e. sets $S_j = V(\mathcal{C}_j)$ are pairwise vertex-disjoint. Recall at this point that we would like to upper bound the number of triangles in G by six times Δ, where Δ is the number of triangles in the cactus $\mathcal{C}$. Notice that $f_3(G)$ is comprised of $f_3(G[S_1]) + f_3(G[S_2]) + f_3(G[S_3]) + q'$, where q' is the number of triangles in G "across" the components S_j (i.e. those triangles whose vertices intersect with at least two sets S_i, S_j, where $i \neq j$. Therefore, if we could somehow give a nice upper bound on q', e.g. if $q' \leq 6$, then we could inductively use $f_3(G[S_j]) \leq 6\Delta_j$ where Δ_j is the number of triangles in $\mathcal{C}_j$, and that therefore

$$f_3(G) \leq 6(\Delta_1 + \Delta_2 + \Delta_3) + 6 \leq 6(\Delta - 1) + 6 = 6\Delta$$

and we would be done. However, it is not possible to give a nice upper bound on q' that holds in general for all situations. We observe that such a bound can be proven for some suitable choice of t: Roughly speaking, removing such a triangle t from $\mathcal{C}$ would create a small "interaction" between components $\mathcal{C}_j$ (i.e. small q'). We say that such a triangle t is a *light* triangle; otherwise, we say that it is *heavy*. Let $\mathcal{C}'$ be the current cactus we are considering. As long as there is a light triangle left in $\mathcal{C}'$, we would remove it (thus breaking $\mathcal{C}'$ into $\mathcal{C}'_1, \mathcal{C}'_2, \mathcal{C}'_3$) and inductively use the bound for each $\mathcal{C}'_j$. Therefore, we have reduced the problem to that of analyzing the base case of a cactus in which all triangles are heavy. Handling the base case of the inductive proof is the main challenge of our result.

We sketch here the two key ideas. Let $S = V(\mathcal{C})$. The first key idea is the way we exploit the locally optimal solution in certain parts of the graph $G[S]$. We want to point out; the fact that all triangles in $\mathcal{C}$ are heavy is exploited crucially in this step. Recall that, each heavy triangle is such that its removal creates three components $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ with many "interactions" (i.e. many triangles across components) between them. This large amount of interaction is the main reason why we could not use induction before. However, intuitively, these triangles across components could serve as candidates for making local improvements. So the fact that there are many interactions would become our advantage in the local search analysis.

We briefly illustrate how we take advantage of heavy triangles. Let $\mathcal{T}$ be the set of triangular faces in G that are not contained in $\bigcup_i G[S_i]$, so each triangle in $\mathcal{T}$ has vertices in at least two subsets S_j, S_i where $j \neq i$. The local search argument would allow us to say that all triangles in $\mathcal{T}$ have one vertex in S_i, one in S_j and one outside of $S_1 \cup S_2 \cup S_3$. This idea is illustrated in Figure 1a.



(a) A 1-swap operation. If there were two triangles t'_1, t'_2 in $\mathcal{T}$ between two different pairs of components S_j, S_i (where $j \neq i$), we could remove t from $\mathcal{C}$ and add t'_1, t'_2 to get a better cactus.

(b) A 2-swap operation. Let t_1 and t_2 be two adjacent triangles in our cactus. If there was an edge between t_1 and t_2, then there would exists a local improvement by removing t_1 and t_2 from $\mathcal{C}$ and adding t'_1, t'_2 and t_3.

Figure 1 Two examples for the swap operations.

Moreover, we will argue that there are not too many triangular faces in $G[S]$, and we give a rough idea of how the exchange argument can be used in Figure 1b.

Finally, the ideas illustrated in both figures are only applied locally in a certain "region" inside the input planar graph G, so globally it is still unclear what would happen. Our final ingredient is a way to decompose the regions inside a plane graph into various "atomic" types. For each such atomic type, the local exchange argument is sufficient to argue optimally about the number of triangles in G in that region compared to that in the cactus. Combining the bounds on these atomic types gives us the desired result. This is the most technically involved part of the paper, and we present it gradually by first showing the analysis that gives $\beta(G) \geq \frac{1}{7} f_3(G)$. For this, we need to classify the regions into five atomic types. To prove the main theorem, that $\beta(G) \geq \frac{1}{6} f_3(G)$, we need a more complicated classification into thirteen atomic types.

Organization of the paper. In Section 2, we give a detailed overview for the proof of our main result. As the proof in full detail would be to long to fit in this extended abstract, we refer the interested reader to a full version on arXiv [8]. In Section 3, we present how to construct a planar graph for which the bound proven in Theorem 1 is tight. In addition we show how it implies the extremal bound provided in [3]. In Section 4, we point out possible directions for future research and extensions of our work.

2 Overview of the Proof

In this section, we give a formal overview of the structure of the proof of Theorem 1. Let our input G be a plane graph (a planar graph with a fixed drawing). Let $\mathcal{C}$ be a locally optimal triangular cactus solution for the natural local search algorithm that uses 2-swap operations, as described in the previous section. Let $\Delta(\mathcal{C})$ denote the number of triangular faces of $\mathcal{C}$ which correspond to the triangular faces of G. We will show $\Delta(\mathcal{C}) \geq f_3(G)/6$. In general, we will use the function $\Delta : G \to \mathbb{N}$ to denote the number of triangular faces in any plane graph G.

We partition the vertices in G into subsets based on the connected components of $\mathcal{C}$, *i.e.* $V(G) = \bigcup_i S_i$ where $\mathcal{C}[S_i]$ is a connected cactus subgraph of $\mathcal{C}$. For each i, where $|S_i| \geq 1$, let $q(S_i)$ denote the number of triangular faces in G with at least two nodes in S_i. The following proposition holds by the 2-swap optimality of $\mathcal{C}$ which implies $f_3(G) = \sum_i q(S_i)$.

▶ **Proposition 4.** *If* $\Delta(\mathcal{C}_i) \geq \frac{1}{6}q(S_i)$ *for all* i*, then* $\Delta(\mathcal{C}) \geq \frac{1}{6}f_3(G)$.

Therefore, it is sufficient to analyze any arbitrary component S_i where $\mathcal{C}[S_i]$ contains at least one triangle of $\mathcal{C}$ (if the component does not contain such a triangle it is just a singleton vertex) and show that $\Delta(\mathcal{C}_i) \geq \frac{1}{6}q(S_i)$. Thus, from now on, we fix such an arbitrary component S_i and denote S_i simply by S, $q(S_i)$ by $q(S)$, and $\Delta(\mathcal{C}[S_i])$ by p. We will show that $q \leq 6p$ through several steps.

Step 1: Reduction to Heavy Cactus

In the first step, we will show that the general case can be reduced to the case where all triangles in $\mathcal{C}$ are *heavy* (to be defined below). We refer to different types of vertices, edges and triangles in the graph G as follows:

- **Cactus.** All edges/vertices/triangles in the cactus $\mathcal{C}[S]$ are called *cactus edges/vertices/triangles* respectively.
- **Cross.** Edges with exactly one end-point in S are called *cross edges*. Triangles that use one vertex outside of S are *cross triangles*. Notice that each cross triangle has exactly one edge in $G[S]$, that edge is called a *supporting edge* of the cross triangle. Similarly, we say that an edge $e \in E(G[S])$ supports a cross triangle; such a cross triangle t contains exactly one vertex v in some component $S_i \neq S$. The component S_i is called the *landing component* of t. Similarly the vertex v is called the *landing vertex* of t.
- **type-$[i]$ edges.** An edge in $G[S]$ that is not a cactus edge and does not support a cross triangle is called a *type-*$[0]$ *edge*. An edge in $G[S]$ that is not a cactus edge and supports i cross triangle(s) is called a *type-*$[i]$ *edge*.

Therefore, each edge in $G[S]$ is a cactus, type-0, type-1 or type-2 edge. The introduced naming convention makes it easier to make important observations like the following (see Figure 2 for an illustration of our naming convention).

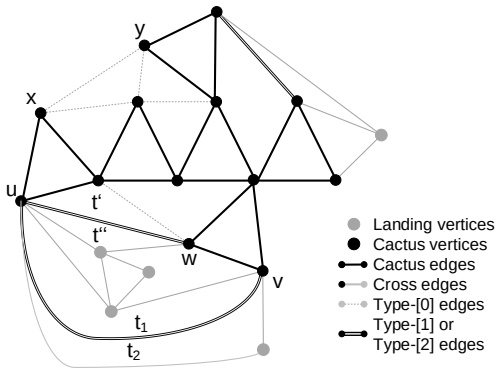


Figure 2 Various types of edges, vertices and triangles. Here the cross triangles t'' and t_1 have the same landing component.

▶ **Observation 5.** *Triangles that contribute to the value of* q *are of the following types: (i) the cactus triangles; (ii) the cross triangles; and (iii) the "remaining" triangles that connect three cactus vertices using at least one type-*0*, type-*1 *or type-*2 *edge, and do not have a cross triangle drawn inside.*

Types of cactus triangles and Split cacti. Consider a (cactus) triangle t in $\mathcal{C}$. For $i \in \{0,1,2,3\}$, we say that t is of type-i if exactly i of its edges support a cross triangle. Let p_i denote the number of type-i cactus triangles, so we have that $p_0 + p_1 + p_2 + p_3 = p$.

We denote the operation of deleting the edges of t from a connected cactus $\mathcal{C}[S]$ by *splitting* $\mathcal{C}[S]$ at t. The resulting three smaller triangular cacti (denoted by $\{\mathcal{C}_v^t\}_{v \in V(t)}$) are referred to as the *split cacti* of t. For each $v \in V(t)$, let $S_v^t := V(\mathcal{C}_v^t)$ be the *split component* containing v. Let $u, v \in V(t) : u \neq v$. Denote by B_{uv}^t the set of type-1 or type-2 edges having one endpoint in S_u^t and the other in S_v^t. Now we are ready to define the concept of heavy and light cactus triangles, which will be crucially used in our analysis.

Heavy and light cactus triangles. We say that a cactus triangle t is *heavy* if either there are at least four cross triangles supported by $E(t) \cup \bigcup_{uv \in E(t)} B_{uv}^t$ or there are at least three cross triangles supported by the edges in one set $B_{uv}^t \cup uv$ for some $uv \in E(t)$ and no cross triangle supported by the rest of the sets $B_{ww'}^t \cup ww'$for each $ww' \in E(t)$. Otherwise, the triangle is *light*. Intuitively, the notion of a light cactus triangle t captures the fact that, after removing t, there is only a small amount of "interaction" between the split components.

We will abuse the notations a bit by using S instead of $V[S]$. Recall, that we denote by $q(S)$ the total number of triangular faces in G with exactly two vertices in S. We denote by $p(S)$ the total number of triangles in the cactus $\mathcal{C}[S]$.

Function φ. Consider a set $S \subseteq V(G)$ and a drawing of $G[S]$ (since we are talking about a fixed drawing of the plane graph G, this is well-defined). Denote by $\ell(S)$ the length of the outer-face f_S of the graph $G[S]$. We define $\varphi(S)$ as the number of edges on the outer-face that do not support any cross triangle drawn on the outer-face, so we have $0 \leq \varphi(S) \leq \ell(S)$.

The main ingredients of Step 1 are encapsulated in the following theorem.

▶ **Theorem 6** (Reduction to heavy triangles). *Let $\gamma \geq 6$ be a real number, and φ be as described above. If $q(S) \leq \gamma p(S) - \varphi(S)$ for all S for which $\mathcal{C}[S]$ is a connected cactus that contains no light triangle, then $q(S) \leq \gamma p(S) - \varphi(S)$ for all S.*

Therefore, if we manage to show the bound $q(S) \leq \gamma p(S) - \varphi(S)$ for the heavy cactus, it will follow that $q \leq \gamma p$ in general (due to non-negativity of function φ). In other words, this gives a reduction from the general case to the case when all cactus triangles are heavy. We end the description of Step 1 by presenting the description of φ.

Step 2: Skeleton and Surviving Triangles

Now, we focus on the case when there are only heavy triangles in the given cactus, and we will give a formal overview of the key idea we use to derive the bound $q(S) \leq 6p(S) - \varphi(S)$, which in combination with Theorem 6, gives our main Theorem 1. For convenience, we refer to the terms $p(S)$ and $q(S)$ as simply p and q respectively.

Structures of heavy triangles. Using local search's swap operations, the light and heavy triangles behave in a very well structured manner. The following proposition summarizes these structures for heavy triangles.

▶ **Proposition 7.** *Let t be a cactus triangle in cactus $\mathcal{C}[S]$.*

- *If t is heavy, then t is either type-0 or type-1.*

- *If t is a heavy type-1 triangle and the edge $uv \in E(t)$ supports the cross triangle supported by t, then $B^t_{ww'} = \emptyset$ for all $ww' \in E(t) \setminus \{uv\}$ and the total number of cross triangles supported by edges in B^t_{uv} is greater than or equal to two.*
- *If t is a heavy type-0 triangle, then there is an edge $uv \in E(t)$ such that $B^t_{ww'} = \emptyset$ for all $ww' \in E(t) \setminus \{uv\}$ and the total number of cross triangles supported by edges in B^t_{uv} is greater than or equal to three.*

By Proposition 7 we can only have type-0 and type-1 cactus triangles in $\mathcal{C}$. Moreover, for each such heavy triangle t, the type-1 or type-2 edges in $G[S]$ only connect vertices of two split components of t.

Let a_i be the number of edges of type-i. Notice that the number of non-cactus edges in $G[S]$ is $\sum_i a_i = |E(G[S])| - 3p$.

Skeleton graph H. Let A be the set of all type-0 edges in $G[S]$ and $H := H[S] := G[S] \setminus A$. Thus $H[S]$ contains only cactus or type-1 or type-2 edges.

Each face f of H possibly contains several faces of G, so we will refer to such a face as a *super-face*. At high-level, our plan is to analyze each super-face f, providing an upper bound on the number of triangular faces of G drawn inside f, and then sum over all such f to retrieve the final result. We call H a *skeleton graph* of G, whose goal is to provide a decomposition of the faces of G into structured super-faces. Denote by $\mathcal{F}$ the set of all super-faces (except for the p faces corresponding to cactus triangles).

Let f be a super-face. Denote by $survive(f)$ the number of triangular faces of G drawn inside f that do not contain any cross triangles. Now we do a simple counting argument for q using the skeleton H as follows: (i) There are p cactus triangles in H, (ii) There are $p_1 + a_1 + 2a_2$ cross triangles supported by edges in $G[S]$, and (iii) There are $\sum_{f \in \mathcal{F}} survive(f)$ triangular faces in G that were not counted in (i) or (ii). Combining this, we obtain:

$$q \leq p + (p_1 + a_1 + 2a_2) + \sum_{f \in \mathcal{F}} survive(f) \tag{1}$$

The first and second terms are expressed nicely as functions of p's and a's, so the key is to achieve the best upper bound on the third term in terms of the same parameters. Roughly speaking, the intuition is the following: When a_2 or a_1 is high (there are many edges in $G[S]$ supporting cross triangles), the second term becomes higher. However, each cross triangle would need to be drawn inside some face in $G[S]$, therefore decreasing the value of the term $\sum_{f \in \mathcal{F}} survive(f)$. Similar arguments can be made for p_1. Therefore, the key to a tight analysis is to understand this trade-off.

The structure of super-faces. Let $f \in \mathcal{F}$ be a super-face. Recall that an edge in the boundary of f is either a type-1 or type-2 edge, or a cactus edge. We aim for a better understanding of the value of $survive(f)$. In general, this value can be as high as $|E(f)| - 2$, e.g. if $G[V(f)]$ is a triangulation of the region bounded by the super-face f using type-0 edges. However, if some edge in the boundary of f supports a cross triangle whose landing component is drawn inside of f in G, this would decrease the value of $survive(f)$, by *killing* the triangular face adjacent to it, hence the term *survive*.

The following observation is crucial in our analysis:

▶ **Observation 8.** *Consider each edge $e \in E(f)$. There are two possible cases:*

- *Edge e is a type-1 or type-2 or cactus edge and supports a cross triangle drawn in f.*
- *Edge e is a type-1 or type-2 or cactus edge and does not support any cross triangle drawn in f.*

Edges lying in the first case are called occupied *edges (the set of such edges in $E(f)$ is denoted by $Occ(f)$), while the others are called* free *edges in f (the set of free edges in $E(f)$ is denoted by $Free(f)$). The length of f can be written as $|E(f)| = |Occ(f)| + |Free(f)|$.*

A very important quantity for our analysis is $\mu(f) = \frac{1}{2} \cdot |Occ(f)| + |Free(f)|$, roughly bounding the value of $survive(f)$ (within some small constant additives terms.)

We will assume without loss of generality that $survive(f)$ is the maximum possible value of surviving triangles that can be obtained by drawing type-0 edges in f, so $\mu(f)$ is a function that depends only on the bounding edges in f. We define $gain(f) = \mu(f) - survive(f)$, which is again a function that only depends on bounding edges of f. Intuitively, the higher the term $gain(f)$, the better for us (since this would lower the value of $survive(f)$), and in fact, it will later become clear that $gain(f)$ roughly captures the "effectiveness" of a local exchange argument on the super-face f. Hence, it suffices to show that $\sum_{f\in\mathcal{F}} gain(f)$ is sufficiently large. The following proposition makes this precise:

▶ **Proposition 9.** $\sum_{f\in\mathcal{F}} survive(f) = (3p - 0.5p_1 + 1.5a_1 + a_2) - \sum_{f\in\mathcal{F}} gain(f)$

Proof. Notice that $\sum_{f\in\mathcal{F}} \mu(f)$ can be analyzed as follows:

- Each cactus triangle is counted three times (once for each of its edges), and for a type-1 triangle, one of the three edges contribute only one half. Therefore, this accounts for the term $3p - 0.5p_1$.
- Each type-1 or type-2 edge is counted two times (once per super-face containing it in its boundary). For a type-2 edge, the contribution is always half (since it always is accounted in $Occ(f)$). For a type-1 edge, the contribution is half on the occupied case, and full on the free case. Therefore, this accounts for the term $1.5a_1 + a_2$.

Overall we get, $\sum_{f\in\mathcal{F}} \mu(f) = 3p - 0.5p_1 + 1.5a_1 + a_2$, which finishes the proof. ◀

Combining this proposition with Equation 1, we get:

$$q \leq 4p + 0.5p_1 + 2.5a_1 + 3a_2 - \sum_{f\in\mathcal{F}} gain(f) \tag{2}$$

A warm-up: Using the gains to prove a weaker bound. To recap, after Step 1 and Step 2, we have reduced the analysis to the question of lower bounding $\sum_{f\in\mathcal{F}} gain(f)$. We first illustrate that we could get a weaker (but non-trivial) result compared to our main result by using a generic upper bound on the gains. In Step 3, we will show how to substantially improve this bound, achieving the ratio of our main Theorem 1 which is tight.

▶ **Lemma 10.** *For any super-face (except for the outer-face) in $\mathcal{F}$, we have $gain(f) \geq 1.5$.*

As the outer (super-)face f_0 of $H[S]$ is special, we can achieve a lower bound on the quantity $gain(f_0)$ that depends on $\varphi(S)$. This is captured by the following lemma.

▶ **Lemma 11.** *For the outer-face f_0, we have that $gain(f) \geq \varphi(S) - 1$.*

$$\sum_{f\in\mathcal{F}} gain(f) \geq \varphi(S) - 1 + 1.5(|\mathcal{F}| - 1) = \varphi(S) + 1.5|\mathcal{F}| - 0.5 \tag{3}$$

The following lemma upper bounds the number of skeleton faces (i.e. super-faces of the skeleton.)

▶ **Lemma 12.** $|\mathcal{F}| = a_1 + a_2 + 1 \leq 2p - 2$.

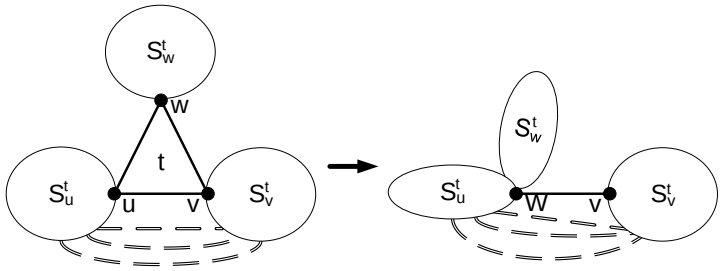


Figure 3 An example of the contraction transformation.

Proof. Proposition 7 allows us to modify the graph H into another simple planar graph $\widetilde{H}$ such that the claimed upper bound on $|\mathcal{F}|$ will follow simply from Euler's formula.

Let t be a cactus triangle where $V(t) = \{u, v, w\}$ and $uw \in E(t)$ be such that the edge set B^t_{uw} is empty, as guaranteed in Proposition 7. For every cactus triangle t we contract the edge uw into one new vertex W. Note that this operation creates two parallel edges with endpoints W and v in the resulting graph. To avoid multi-edges in the resulting graph $\widetilde{H}$ we remove one of them (see Figure 3 for an illustration of this operation). Since B^t_{uw} is empty this operation cannot create any other multi-edges in $\widetilde{H}$. In addition the contraction of an edge maintains planarity, hence after each such transformation the graph remains simple and planar. As a result of applying the above operation to all cactus triangles, the graph $\widetilde{H}$ has $p + 1$ vertices and p edges corresponding to the contracted triangles. By Euler's formula the number of edges in $\widetilde{H}$ is at most $3(p + 1) - 6 = 3p - 3$, which implies that $a_1 + a_2 \leq 2p - 3$, and as $|\mathcal{F}| = a_1 + a_2 + 1$ we get that $|\mathcal{F}| \leq 2p - 2$. ◀

Combining the trivial gains (i.e. Inequality 3) with Inequality 2, we get

$$q \leq (4p{+}0.5p_1{+}2.5a_1{+}3a_2){-}(\varphi(S){+}1.5(a_1{+}a_2{+}1){-}2.5) = 4p{+}0.5p_1{+}a_1{+}1.5a_2{-}\varphi(S){+}1$$

Now, using Lemma 12 and the trivial bound that $p_1 \leq p$, we get $q(S) \leq 4.5p + 1.5(a_1 + a_2) - \varphi(S) + 1 \leq 7.5p(S) - \varphi(S)$, therefore implying a factor 7.5 upper bound.

Step 3: Upper Bounding Gains via Super-Face Classification

In this final step, we show another crucial idea that allows us to reach a factor 6. Intuitively, the most difficult part of lower bounding the total gain is the fact that the value of $gain(f)$ is different for each type of super-face, and one cannot expect a strong "universal" upper bound that holds for all of them. For instance, Figure 4 shows a super-face with $gain(f) = 1.5$, so strictly speaking, we cannot improve the generic bound of 1.5.

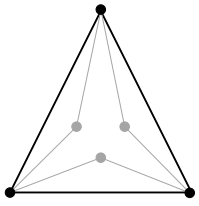


Figure 4 A super-face $f \in \mathcal{F}$ having $gain(f) = 1.5$; $\mu(f) = 1.5$ and $survive(f) = 0$.

This is where we introduce our final ingredient, that we call *classification scheme*. Roughly, we would like to "classify" the super-faces in $\mathcal{F}$ into several types, each of which has the same gain. Analyzing super-faces with similar gains together allows us to achieve a better result.

Super-face classification scheme. We are interested in coming up with a set of rules Φ that classifies $\mathcal{F}$ into several types. We say that the rule Φ is a d-type classification if the rules classifies $\mathcal{F}$ into d sets $\mathcal{F} = \bigcup_{j=1}^{d} \mathcal{F}[j]$. Let $\vec{\chi}$ be a vector such that $\vec{\chi}[i] = |\mathcal{F}[i]|$. We would like to prove a good lower bound on the gain for each such set. We define the gain vector by $\overrightarrow{gain}$ where $\overrightarrow{gain}[i] = \min_{f \in \mathcal{F}[i]} gain(f)$. The total gain can be rewritten as:

$$\sum_{f \in \mathcal{F}} gain(f) = \overrightarrow{gain} \cdot \vec{\chi}$$

Notice that, the total gain value $\overrightarrow{gain} \cdot \vec{\chi}$ would be written in terms of the $\vec{\chi}[j]$ variables, so we would need another ingredient to lower bound this in terms of variables p's and a's. Therefore, another component of the classification scheme is a set of *valid linear inequalities* Ψ of the form $\sum_{j=1}^{d} C_j \vec{\chi}[j] \leq \sum_{j \in \{0,1\}} d_j p_j + \sum_{j \in \{1,2\}} d'_j a_j$. This set of inequalities will allow us to map the formula in terms of $\vec{\chi}[j]$ into one in terms of only p's and a's.

A classification scheme is defined as a pair (Φ, Ψ). We say that such a scheme certifies the proof of factor γ if it can be used to derive $q(S) \leq \gamma p(S) - \varphi(S)$. Given a fixed classification scheme and a gain vector, we can check whether it certifies a factor γ by using an LP solver (although in our proof, we would show this derivation.)

Our main result is a scheme that certifies a factor 6. Since the proof is complicated, we also provide a simpler, more intuitive proof that certifies a factor 7 first.

▶ **Theorem 13.** *There is a* 5*-type classification scheme that gives a factor* 7*.*

We remark that the analysis of factor 7 only requires a cactus that is locally optimal for 1-swap.

▶ **Theorem 14.** *There is a* 13*-type classification scheme that gives a factor* 6*.*

Intuition. The classification scheme would intuitively set the rules to separate the super-faces that would benefit from local search's exchange argument from those that would not. Therefore, for the good cases, we would obtain a much better gain, e.g., in one of our classification type, $gain(f)$ is as high as 4.5. In the bad cases that there is no such benefit, we would still use the lower bound of 1.5 that holds in general for any super-face.

3 On the Strength of Our Result

3.1 Our Bound is Almost Tight

In this section, we show that there exists a graph G for which $\beta(G) \leq (\frac{1}{6} + o(1)) f_3(G)$. We show this indirectly using a family of graphs presented in [7], as stated in the following lemma.

▶ **Lemma 15** ([7])**.** *There is a family of* n*-vertex planar graphs* $\{H_n\}_{n \in \mathbb{Z}}$ *for which there exist a maximal cactus subgraph* C_n *of* H_n *such that* $\frac{f_3(C_n)}{f_3(H_n)} \leq \frac{1}{12} + o_n(1)$*.*

In [7], this family of graphs is used to show that a maximal cactus (not maximum) is not sufficient to improve over the best known greedy strategies when approximating MPT. In the context of this paper we use C_n to compare it to a maximum cactus for H_n to prove the following.

▶ **Theorem 16.** *Let H_n be the graph family as in Lemma 15. Then,* $\frac{\beta(H_n)}{f_3(H_n)} \leq \frac{1}{6} + o_n(1)$.

Proof. By Lemma 15, it suffices to argue that $f_3(C_n) \geq \frac{\beta(H_n)}{2}$. Let C_n^* be an optimal cactus with $\beta(H_n)$ triangles. Notice that for any triangle t in C_n, $E(t)$ intersects at most two other triangles in C_n^*. If all three edges of t were to be used by three different triangles in C_n^*, this would contradict the cactus property. Moreover, if t does not intersect any triangle in C_n^* this would imply that one of its edges would complete a cycle if added to C_n^*. By these two observations we can use a simple counting scheme to upper-bound the number of triangles in C_n^* depending on the number of triangles in C_n. We iteratively add triangles of C_n to C_n^* and count in every step how many triangles in C_n^* need to be removed to maintain the cactus property. For every triangle in C_n that intersects C_n^* in one or two edges, we have to remove at most two triangles from C_n^*. For every triangle in C_n, that does not intersect C_n^* in any edge, we have to break a cycle in the resulting C_n^* by deleting one other triangle from it. In each iteration we therefore destroy at most two triangles from the original C_n^* and therefore get $f_3(C_n^*) \leq 2f_3(C_n)$. This concludes the proof as $f_3(C_n) \geq f_3(C_n^*)/2 = \beta(H_n)/2$. ◀

3.2 Comparison to the Previous Bound

One integral part to derive the improved approximation ration for MPS in [3] was to show that for any given planar graph $G = (V, E)$ with $n = |V|$ vertices and $|E| = 3n - 6 - t(G)$ edges, we have:

▶ **Theorem 17** ([3])**.** *Let G be as above, then $\beta(G) \geq \frac{1}{3}(n - t(G) - 2)$.*

As removing one edge from a triangulated planar graph merges exactly two faces, we can easily derive a lower bound that depends on $t(G)$, for the number of triangular faces in G:

$$f_3(G) \geq 2n - 2t(G) - 4$$

By Theorem 1, we have that $\beta(G) \geq \frac{1}{6}f_3(G)$. Combining these two facts implies Theorem 17.

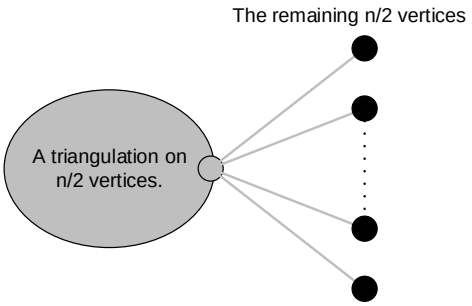


Figure 5 Bad example which shows that a extremal bound like the one in in [3] for MPS does not necessarily imply a similarly strong result to MPT.

We end this section by showing that the bound in [3] alone is not sufficient for approximating MPT. To this end we construct a graph in which $\frac{1}{3}(n - t(G) - 2) \leq 0$, even though $f_3(G) = \Theta(n)$, Let G be a planar graph with n vertices, where $\frac{n}{2}$ vertices form a triangulated planar subgraph. Let v be a vertex on the outer-face of this triangulated structure. The remaining $\frac{n}{2}$ vertices are embedded in the outer-face and are incident to exactly one edge each, with the other endpoint being v (see Figure 5 for an illustration

of this construction). Therefore by Euler's formula, the number of edges in this graph is equal to $3(\frac{n}{2}) - 6 + \frac{n}{2} = 2n - 6$ and thus $t(G) = n$, while the number of triangular faces is $f_3(G) = 2(\frac{n}{2}) - 4 - 1 = n - 5$.

4 Conclusions and Open Problems

Our work implies that a natural local search algorithm gives a $(\frac{4}{9} + \epsilon)$-approximation for MPS and a $\frac{1}{6} + \epsilon$ approximation for MPT. To be more precise, when given any graph G, we follow the t-swap local search strategy for $t = O(1/\epsilon)$: Start from any cactus subgraph H. Try to improve it by removing t triangles and adding $(t + 1)$ triangles in a way that ensures that the graph remains a cactus subgraph. A local optimal solution will always be a $(\frac{4}{9} + \epsilon)$ approximation for MPS and a $(\frac{1}{6} + \epsilon)$ approximation for MPT.

Knowing this fact, there is an obvious candidate algorithm for improving over the long-standing best approximation factor for MPS. We call a graph H a diamond-cactus if every block in H is either a diamond[5] or a triangle. Start from any diamond-cactus subgraph H of G and then try to improve it by removing t triangles from H and adding $(t + 1)$ triangles, maintaining the fact that H is a diamond-cactus subgraph. We conjectured that this algorithm gives a better than $\frac{4}{9}$-approximation for MPS, but we suspect that the analysis will require substantially new ideas.

Another interesting direction is to see whether there is a general principle that captures a denser planar structure than cactus subgraphs by going above matroid parity in the hierarchy of efficiently computable problems. For instance, are diamond-cactus subgraphs captured by matroid parity? Or can it be formulated as an even more abstract structure than matroids (e.g. commutative rank [2]) that can still be computed efficiently? We believe that studying this direction will lead to a better understanding of algebraic techniques for finding dense planar structures.

Finally, the absence of LP-based techniques in this problem domain seems rather unfortunate. There have been some experimental studies recently, but the theoretical understanding of what can be proven formally in the context of power of relaxation is certainly lacking [16, 10, 11]. Is there a convex relaxation that allows us to find a relatively dense planar subgraph (e.g. $(3 - \epsilon)$-approximation for MPS using LP-based techniques)?

References

1 Claude Berge. La theorie des graphes. *Paris, France*, 1958.

2 Markus Bläser, Gorav Jindal, and Anurag Pandey. Greedy Strikes Again: A Deterministic PTAS for Commutative Rank of Matrix Spaces. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 33:1–33:16, 2017.

3 Gruia Călinescu, Cristina G Fernandes, Ulrich Finkler, and Howard Karloff. A better approximation algorithm for finding planar subgraphs. *Journal of Algorithms*, 27(2):269–302, 1998.

4 Gruia Calinescu, Cristina G Fernandes, Howard Karloff, and Alexander Zelikovsky. A new approximation algorithm for finding heavy planar subgraphs. *Algorithmica*, 36(2):179–205, 2003.

5 Gruia Călinescu, Cristina G Fernandes, Hemanshu Kaul, and Alexander Zelikovsky. Maximum series-parallel subgraph. *Algorithmica*, 63(1-2):137–157, 2012.

[5] A diamond subgraph is a graph that is isomorphic to the graph resulting from deleting any single edge from a K_4.

6 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 743–754. IEEE, 2017.

7 Parinya Chalermsook and Andreas Schmid. Finding Triangles for Maximum Planar Subgraphs. In *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, (WALCOM'17), Proceedings.*, pages 373–384, 2017.

8 Parinya Chalermsook, Andreas Schmid, and Sumedha Uniyal. A Tight Extremal Bound on the Lovász Cactus Number in Planar Graphs. *CoRR*, abs/1804.03485, 2018. `arXiv:1804.03485`.

9 Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. Algebraic algorithms for linear matroid parity problems. *ACM Transactions on Algorithms (TALG)*, 10(3):10, 2014.

10 Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Exact Algorithms for the Maximum Planar Subgraph Problem: New Models and Experiments. In *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, pages 22:1–22:15, 2018. `doi:10.4230/LIPIcs.SEA.2018.22`.

11 Markus Chimani and Tilo Wiedera. Cycles to the Rescue! Novel Constraints to Compute Maximum Planar Subgraphs Fast. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2018.19`.

12 Harold N Gabow and Matthias Stallmann. An augmenting path algorithm for linear matroid parity. *Combinatorica*, 6(2):123–150, 1986.

13 Magnús M Halldórsson. Approximations of weighted independent set and hereditary subset problems. In *Graph Algorithms And Applications 2*, pages 3–18. World Scientific, 2004.

14 Johan Håstad. Clique is hard to approximate withinn 1- ε. *Acta Mathematica*, 182(1):105–142, 1999.

15 T.A. Jenkyns. *Matchoids : a Generalization of Matchings and Matroids.* Thesis (Ph.D.)–University of Waterloo, 1974.

16 M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, July 1996. `doi:10.1007/BF02086607`.

17 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In *International Colloquium on Automata, Languages, and Programming*, pages 226–237. Springer, 2006.

18 Eugene L Lawler. *Combinatorial optimization: networks and matroids.* Courier Corporation, 1976.

19 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Matroid matching: the power of local search. *SIAM Journal on Computing*, 42(1):357–379, 2013.

20 John M Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

21 László Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Series B*, 28(2):208–236, 1980.

22 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

23 Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. In *International Colloquium on Automata, Languages, and Programming*, pages 40–51. Springer, 1993.

24 James B Orlin. A fast, simpler algorithm for the matroid parity problem. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 240–258. Springer, 2008.

25 Zoltán Szigeti. On a min-max theorem of cacti. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 84–95. Springer, 1998.

26 William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

Average-Case Completeness in Tag Systems

Matthew Cook
University of Zürich, Switzerland
ETH Zürich, Switzerland
cook@ini.ethz.ch

Turlough Neary
University of Zürich, Switzerland
ETH Zürich, Switzerland
tneary@ini.ethz.ch

Abstract

To prove average-case NP-completeness for a problem, we must choose a known average-case complete problem and reduce it to that problem. Unfortunately, the set of options to choose from is far smaller than for standard (worst-case) NP-completeness. In an effort to help remedy this we focus on tag systems, which due to their extreme simplicity have been a target for other types of reductions for many problems including the matrix mortality problem, the Post correspondence problem, the universality of cellular automaton Rule 110, and all of the smallest universal single-tape Turing machines. Here we show that a tag system can efficiently simulate a Turing machine even when the input is provided in an extremely simple encoding which adds just $\log n$ carefully set bits to encode an arbitrary Turing machine input of length n. As a result we show that the bounded halting problem for nondeterministic tag systems is average-case NP-complete. This result is unexpected when one considers that in the current state of the art for simple universal systems it had appeared that there was a trade-off whereby simpler systems required more complicated input encodings. In other words, although simple systems can compute interesting things, they had appeared to require very carefully encoded inputs in order to do so. Our result surprisingly goes in the opposite direction by giving the first average-case completeness result for such a simple model of computation. In ongoing work we have already found applications of our result having used it to give average-case NP-completeness results for a 2D generalization of the Collatz function, a nondeterministic version of the 2D elementary functions studied by Koiran and Moore, 3D piecewise affine maps, and bounded Post correspondence problem instances that use simpler word pairs than previous results.

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

Keywords and phrases average-case NP-completeness, encoding complexity, tag system, bounded halting problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.20

Funding *Turlough Neary*: Supported by Swiss National Science Foundation grant numbers 200021-153295 and 200021-166231.

1 Introduction

Given the massive interest in worst-case NP-completeness, the literature devoted to proving the stronger result of average-case NP-completeness can be considered quite limited. This is surprising when one considers the practical importance of determining whether or not we are likely to encounter intractable instances in problems we wish to solve. One reason for the smaller number of results is that it is more difficult to prove the stricter form of reduction required to show average-case NP-completeness [11, 25]. A first step towards overcoming this difficulty is to give new average-case completeness results for problems whose simplicity allows for easier average-case reductions to other systems.

Of all the simple models in the literature where a new average-case completeness result would have applications to a wide range of problems, perhaps the most compelling case can be made for tag systems, a very simple form of rewriting system introduced by Post [18]. The simplicity of the context free rewrite rule employed by tag systems has made them a favored target of simulation by many other systems. For this reason, tag systems have been used either through individual reductions or via chains of reductions to prove many undecidability and hardness results (e.g. [4, 9, 12, 21, 22, 23, 24, 26]). Reductions to tag systems have also yielded significant improvements in lower bounds for a number of well studied problems [14, 16, 19, 20]. So proving that the bounded halting problem for tag systems is average-case NP-complete could lead to other new average-case NP-completeness results and even improved lower bounds for existing results. In fact we [5] have already used 2-tag systems to give an average-case NP-completeness result for Post correspondence problem instances that use shorter word pairs than those found in [8, 25]. In ongoing work [6] we have already begun using 2-tag systems as the starting point for chains of simulations that prove average-case NP-completeness. We have used 2-tag systems to prove the average-case NP-completeness of a bounded reachability problem for a generalized 2D[1] version of the Collatz function that is nondeterministic [6]. As a corollary of our result we find that a nondeterministic version of the 2D elementary functions of Koiran and Moore [10] also have a bounded reachability problem that is average-case NP-complete. In addition we simulate tag systems to prove an average-case NP-completeness result for bounded reachability in 3D piecewise affine maps that are nondeterministic [6].

It is worth noting that the applications of tag systems given in the references above are not where the applications end; they propagate to other results through further chains of reductions. The results in [16] are an example where binary tag systems were used to significantly improve the undecidability bounds for both the Post correspondence problem and the matrix mortality problem. The new bounds for the matrix mortality problem also give improved undecidability bounds for the problem in [2] of reaching the origin with piecewise linear systems and for the quantum measurements problem in [7]. The results in the present paper are a first step towards proving average case completeness results for bounded versions of these problems.

There are some obvious reasons to think an attempt to prove an average-case completeness result for a system as simple as a tag systems is doomed to fail. It seems natural to expect that the simplest systems require unwieldy encodings to compute, or suffer from an exponential trade-off when it comes to time efficiency, and for a long time the literature seemed to bear this out. However, in [17, 26] Neary and Woods showed that many of the simplest known models of computation [4, 12, 14, 19, 20, 22] actually simulate Turing machines in polynomial time, an exponential improvement over the previous simulations. It follows that many simple systems now have a P-complete prediction problem, which means that there is no known way to predict the long term behavior of these systems significantly faster than by explicit step by step simulation.

Despite these improvements in efficiency it remained the case that the simplest universal systems utilized complicated input encodings [4, 12, 13, 14, 16, 19, 20]. So it seemed reasonable to expect that as programs get shorter or the form of rules get simpler, extra complexity gets forced into the input encoding. This observation was expressed nicely by Yedidia and Aaronson in [27]:

[1] It is an open problem as to whether or not generalized 1D Collatz functions can simulate Turing machines in polynomial time [10] and so proving an NP-completeness result for a nondeterministic generalization of 1D Collatz functions would most likely require some radically new encoding technique.

> "the known small universal Turing machines achieve their small size only at the cost of an extremely complicated description format for the input machine. That is, most of the complexity gets "shunted" from the Turing machine itself to the input encoding format."

The complexity of the input encodings used by the simplest known universal systems means that their input encodings have a very low chance of occurring when sampled from a uniform distribution over the input alphabet. So while many of the simplest systems are now known to have P-complete prediction problems based on carefully encoded inputs, it could nonetheless be the case that the behavior of the simplest universal systems is easy to predict *on average*.

Here we give a first result indicating that this is not the case. Specifically, we show that tag systems can efficiently simulate the computation of binary Turing machines when provided with an extremely simple encoding which adds just $\log n$ carefully set bits to encode an arbitrary input of length n. As a result we find that the bounded halting problem for nondeterministic tag systems is average-case NP-complete.

2 Preliminaries

The length of a word w is denoted by $|w|$. We let ϵ denote the empty word. Given a natural number i we let $\langle i \rangle$ be its binary digit representation.

2.1 2-Tag Systems

▶ **Definition 1.** *A 2-tag system consists of a finite alphabet of symbols Σ and a finite set of rules $R : \Sigma \to \Sigma^*$.*

The computation of a 2-tag system acts on a word $w = \sigma_0\sigma_1 \dots \sigma_l$ which we call the *dataword*. The entire configuration is given by w. In a computation step, the two symbols $\sigma_0\sigma_1$ are deleted and we apply a rule for the first symbol σ_0, i.e., a rule of the form $\sigma_0 \to \sigma_{l+1} \dots \sigma_{l+c}$, by appending the word $\sigma_{l+1} \dots \sigma_{l+c}$. A dataword (configuration) w_2 is obtained from w_1 via a single computation step as follows:

$$\sigma_0\sigma_1\sigma_2 \dots \sigma_l \vdash \sigma_2 \dots \sigma_l\sigma_{l+1} \dots \sigma_{l+c}$$

where $\sigma_0 \to \sigma_{l+1} \dots \sigma_{l+c} \in R$. A 2-tag system *halts* if $|w| < 2$ or if there is no rule defined for the leftmost symbol σ_0 . A *round* is the $\lfloor \frac{|w|}{2} \rfloor$ or $\lceil \frac{|w|}{2} \rceil$ computation steps that traverse the word w exactly once. We say a symbol σ_i of w is *read* if and only if at the start of some computation step it is the leftmost symbol (i.e. i is even, so the rule $\sigma_i \to \sigma_{k+1} \dots \sigma_{k+c}$ will be applied). In this work we consider tag systems that are nondeterministic, that is they are permitted to have more than one rule for each $\sigma_i \in \Sigma$.

In [26] it was shown that 2-tag systems efficiently simulate deterministic binary Turing machines in time $O(t^4(\log t)^2)$ where t is the running time of the Turing machine. This time overhead was later improved in Chapter 5 of [15] to give Theorem 2.

▶ **Theorem 2** (Woods and Neary [26, 15])**.** *Given a single tape deterministic Turing machine M that computes in time t then there is a 2-tag system T_M that simulates the computation of M and computes in time $O(t^2 \log t)$.*

In [15] given a binary input word $w = x_1x_2 \dots x_n$ for Turing machine M it is encoded as the T_M input dataword

$$\bar{x}_1\dot{\bar{x}}_1\, x_2\dot{x}_2\, x_3\dot{x}_3 \dots\, x_n\dot{x}_n\, (aa)^{2^{\lceil \log_2 n \rceil + c}} \tag{1}$$

While the results in [26, 15] offer a polynomial time simulation of Turing machines using this input encoding, this does not allow us to prove an average-case NP-completeness result, as the probability of choosing a word that encodes some w (via Equation (1)) is exponentially smaller than the probability of choosing w. In Lemma 11 we will show how 2-tag systems can compute the encoding in Equation (1) when provided with a more compact encoding of M's input word. Our compact binary encoding requires only $n + \log n$ encoding bits to encode M's input word (the remaining $n + \log n$ bits in our length $2(n + \log n)$ encoding are arbitrary padding bits). This gives an input encoding that is only polynomially less likely to occur than the input to M and so using Lemma 11 and the results in [26] we can prove that the bounded halting problem for tag systems is average-case NP-complete.

2.2 Average-Case Complexity

When we speak of the average-case complexity of a decision problem we are considering the expected time to solve that problem with respect to some distribution over instances of the problem. This leads to the notion of a distributional problem [1, 8, 11, 25]. A distributional problem is a pair (D, μ) where D is a decision problem and μ is a distribution over instances of D. In this work instances of D are given as binary words and we let L_D be the set of instances for which the answer to the problem D is positive.

The definitions in this section are adapted from [1, 8, 11, 25]. For the distributional problems in this work we take the uniform probability distribution [8, 11] for binary words $w \in \{0,1\}^*$ which is proportional to $\mu(w) = |w|^{-2}2^{-|w|}$. The probability of choosing a word of length n is proportional to $1/n^2$. We denote the distribution over all words of length n with μ_n and so we have $\mu_n(w) = 2^{-n}$. We let $Pf(w_1, w_2, \ldots, w_m) \in \{0,1\}^*$ be the prefix free code for the binary words $w_1, w_2, \ldots, w_m$ where $|Pf(u_1, \ldots, u_m)| = 2m + \sum_{i=1}^{m} |w_i| + 2\log_2 |w_i|$ (see [3]).

▶ **Definition 3** (Average polynomial function). *A function f that maps words to natural numbers is polynomial on average with respect to a distribution μ if there exists an $\epsilon > 0$ such that for all n*

$$\sum_{|w|=n} \mu_n(w)(f(w))^\epsilon = O(n)$$

where $w \in \{0,1\}^n$.

▶ **Definition 4** (Average polynomial time). *An algorithm runs in average polynomial time with respect to a distribution μ if its running time is bounded by an average polynomial function with respect to μ.*

Average-case reductions insist that when reducing a distributional problem (D, μ) to another distributional problem (D', μ'), instances $x \in D$ should be at most polynomially more likely than the instances they reduce to. This property is enforced by condition 2 of Definition 5. Previous 2-tag system simulations of Turing machine used input encodings that do not satisfy this condition. In Lemma 11 and Corollary 12 we show that 2-tag system with a concise input encoding can simulate Turing machines efficiently, and then in Theorem 13 we show that this new input encoding satisfies condition 2 of Definition 5.

▶ **Definition 5** (Ptime reduction between distributional problems). *A distributional problem (D, μ)* Ptime reduces *to a distributional problem (D', μ') if there exists a polynomial time computable function $g(w) = y$, where $w \in D$ and $y \in D'$, and a polynomial $p(|w|)$ such that the following two conditions hold:*

1. $g(w) \in L_{D'}$ *if and only if* $w \in L_D$
2. $\sum_{g(w)=y} \mu(w) \leqslant p(|w|)\,\mu'(y)$

▶ **Definition 6** (Dilation of a distributional problem)**.** *A dilation* Δ *of a distributional problem* (D, μ) *is a distributional problem* (D_Δ, μ_Δ) *where instances of* D_Δ *include extra padding. The dilation* Δ *maps each instance* w *of* D *to a set given by* $\{Pf(w, s_w)|s_w \in S_w\}$ *where* S_w *is a finite set of binary words. The set* D_Δ *is given by the union of all sets produced by applying* Δ *to the instances of* D *(i.e.* $D_\Delta = \bigcup_{w \in D} \{Pf(w, s_w)|s_w \in S_w\}$*). For each instance* $Pf(w, s_w)$ *of* D_Δ*,* $Pf(w, s_w) \in L_{D_\Delta}$ *if and only if* $w \in L_D$*. The probability distribution is given by* $\mu_\Delta(Pf(w, s_w)) = \dfrac{\mu(w)2^{-|s_w|}}{\sum_{r' \in S_w} 2^{-|r'|}}$.

We say a dilation is a *Ptime dilation* if it is computed by a randomized algorithm $\mathcal{A}$ that runs in polynomial time, that is on input w $\mathcal{A}$ outputs an element from $\{Pf(w, s_w)|s_w \in S_w\}$ in time polynomial in $|w|$. We think of s_w as the sequence of coin flips made by $\mathcal{A}$.

▶ **Definition 7** (Nonrare dilation)**.** *A dilation* Δ *(as defined in Definition 3) is nonrare if* R_Δ *(given in Equation* (2)*) is polynomial on average with respect to the distribution* μ'*.*

$$R_\Delta(w) = \frac{1}{\sum_{s \in S_w} 2^{-|s|}} \tag{2}$$

▶ **Definition 8** (Ptime randomized reduction between distributional problems)**.** *A distributional problem* (D, μ) *randomly reduces to a distributional problem* (D', μ') *if* (D, μ) *has a nonrare Ptime dilation* (D_Δ, μ_Δ) *such that* (D_Δ, μ_Δ) *Ptime reduces to* (D', μ') *(see Definition 5).*

Let $M_1, M_2, M_3, \ldots$ be an enumeration of nondeterministic binary Turing machines and let $T_1, T_2, T_3, \ldots$ be an enumeration of nondeterministic 2-tag systems.

The problem in Definition 9 is known to be average-case NP-complete and it will be used in our reduction at the end of the next section to prove that the problem for tag systems given in Definition 10 is average-case NP-complete.

▶ **Definition 9** (Distributional bounded halting problem for nondeterministic Turing machines)**.**
Problem: Given a nondeterministic Turing machine M_i*, a binary word* w*, and a natural number* t*, determine whether or not* M *halts in* t *steps when given* w *as input.*
Instance: A binary word $Pf(\langle i \rangle, w, 1^t)$
Distribution: Proportional to $\mu(\langle i \rangle, w, t) = 2^{-(|\langle i \rangle|+|w|)}|\langle i \rangle|^{-2}|w|^{-2}t^{-2}$.

▶ **Definition 10** (Distributional bounded halting problem for nondeterministic 2-tag systems)**.**
Problem: Given a nondeterministic 2-tag system T_i*, a binary word* w*, and a natural number* t*, determine whether or not* T *halts in* t *steps when given* w *as input.*
Instance: A binary word $Pf(\langle i \rangle, w, 1^t)$
Distribution: Proportional to $\mu(\langle i \rangle, w, t) = 2^{-(|\langle i \rangle|+|w|)}|\langle i \rangle|^{-2}|w|^{-2}t^{-2}$.

3 New Concise Input Encoding for 2-Tag Systems

In Equation (3), we define an encoding function $f : \Sigma^n \times \Sigma^{n+\lceil \log_2 n \rceil + c} \to \{0,1\}^*$ where $c = 1$ if $2^{\lceil \log_2 n \rceil} < n + \lceil \log_2 n \rceil$ otherwise $c = 0$. The encoding function takes $w = x_1 x_2 \ldots x_n$ an arbitrary binary Turing machine input and $s = z_1 z_2 \ldots z_{n+\lceil \log_2 n \rceil + c}$ an arbitrary padding

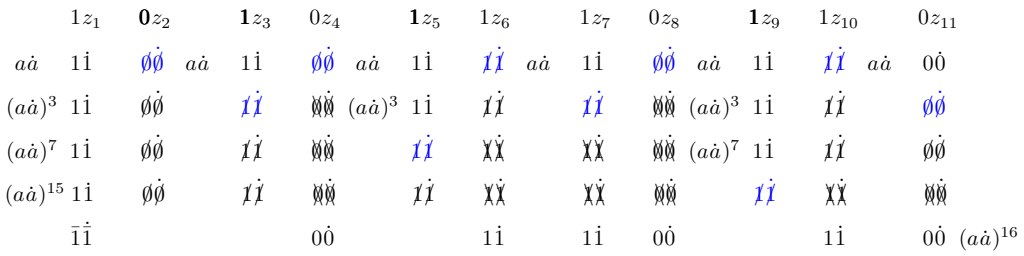


Figure 1 Six datawords giving an overview of the tag system algorithm in Lemma 11. The dataword at the top is the input to the tag system and is the encoding of the word 1011010 via Function (3). The extra parity bits inserted are highlighted in the top row in bold. The extra white space between pairs of symbols and the vertical alignment of symbols are for readability. The $z_i \in \{0,1\}$ are arbitrary padding symbols that are not read by the tag system so the first round eliminates them. The second line from top gives the dataword after one iteration (three rounds) of our algorithm on the input dataword, the third line from the top gives the dataword after two iterations of our algorithm on the input dataword and so on until the fifth dataword where one further round produces the output of the algorithm on the last line. Each iteration of our algorithm involves three rounds of the dataword. A detailed view of the 2 other rounds not shown here is given in Datawords (7) and (8).

word (where $x_i, z_j \in \{0,1\}$) and maps the pair to a 2-tag system input word. The value of $f(w,s)$ is obtained by taking $x_1x_2\ldots x_n$ and inserting either $\lceil \log_2 n \rceil$ or $\lceil \log_2 n \rceil + 1$ extra *parity bits* and then adding a padding bit (z_j) after each bit in the resulting word. An example of the application of f appears in Figure 1.

$$f(w,s) = \underset{1}{u}_1 z_1 \, \underset{1}{u}_2 z_2 \, \ldots \, \underset{1}{u}_{n+\lceil \log_2 n \rceil + c} \, z_{n+\lceil \log_2 n \rceil + c} \begin{cases} z_i \in \{0,1\} & \text{(unread bits)} \\ \underset{1}{u}_i = x_{i-\lceil \log_2 i \rceil} & \text{if } \forall_k \;\; i \neq 2^k + 1 \\ \underset{1}{u}_{2^k+1} = h(w,k) & \text{(parity bits)} \end{cases} \tag{3}$$

where $h(w,k)$ is given by Equation (4), $w_{2^k+1} = \underset{1}{u}_{3\cdot 2^k+1} \underset{1}{u}_{5\cdot 2^k+1} \underset{1}{u}_{7\cdot 2^k+1} \ldots \underset{1}{u}_{m\cdot 2^k+1}$ is a binary word of certain input bits with $m = 2y+1$, $y, k \in \mathbb{N}$, $0 \leqslant k < \lceil \log_2 n \rceil + c$, and $m(2^k) + 1 \leqslant n + \lceil \log_2 n \rceil + c < (m+2)2^k + 1$. Note that each w_{2^k+1} word (there is one for each k) is formed from a set of bits, and these sets are disjoint from each other and from the set of parity bits, but the union of these sets and the set of parity bits is the entire input word (apart from the throw-away z_i bits). The function h indicates how word w_{2^k+1} is used to set parity bit $\underset{1}{u}_{2^k+1}$.

$$h(w,k) = \begin{cases} 0 & \text{if } |w_{2^k+1}| > 0 \text{ and number of 1 symbols in } w_{2^k+1} \text{ is even} \\ 1 & \text{if } |w_{2^k+1}| > 0 \text{ and number of 1 symbols in } w_{2^k+1} \text{ is odd} \\ 1 & \text{if } |w_{2^k+1}| = 0 \end{cases} \tag{4}$$

▶ **Lemma 11.** *Let $w = x_1x_2\ldots x_n$ and $s = z_1z_2\ldots z_{n+\lceil \log_2 n \rceil + c}$ be binary words where $x_i, z_j \in \{0,1\}$. There is a 2-tag system $\mathcal{T}$ that takes a binary word of the form*

$$f(w,s) = \underset{1}{u}_1 z_1 \, \underset{1}{u}_2 z_2 \, \ldots \, \underset{1}{u}_{n+\lceil \log_2 n \rceil + c} \, z_{n+\lceil \log_2 n \rceil + c} \tag{5}$$

as input and produces a word of the form

$$\underset{4}{\bar{x}}_1\underset{4}{\dot{\bar{x}}}_1\ \underset{4}{x}_2\underset{4}{\dot{x}}_2\ \cdots\ \underset{4}{x}_n\underset{4}{\dot{x}}_n\ (\underset{4}{a}\underset{4}{\dot{a}})^{2^{\lceil \log_2 n\rceil + c}} \tag{6}$$

in time $O(n \log_2 n)$.

Proof. We prove the existence of such a tag system $\mathcal{T}$ by exhibiting one and showing why it works, which will take the next few pages.

To produce Dataword (5) from Dataword (6) there are three tasks to be carried out: (a) append $(\underset{2}{a}\underset{2}{\dot{a}})^{2^{\lceil \log_2 n\rceil + c}}$ to the right end of the word, (b) change the first pair of symbols be uniquely in the form $\underset{4}{\bar{x}}\underset{4}{\dot{\bar{x}}}$ and (c) delete the parity symbols (i.e. $u_{2^k+1}\dot{u}_{2^k+1}$ pairs in Equation (3)). We give an overview of how the algorithm achieves these tasks concurrently and then we give the rules for the tag system and explain how they implement this algorithm.

Tasks (a) and (b)

To append a subword of the form $(a\dot{a})^{2^{\lceil \log_2 n\rceil + c}}$ one could append a single $a\dot{a}$ pair at the end of the dataword and then iterate a process where on each iteration each $a\dot{a}$ pair is mapped to $a\dot{a}\,a\dot{a}$ so that after $\lceil \log_2 n\rceil + c$ iterations the initial $a\dot{a}$ pair has grown to become $(a\dot{a})^{2^{\lceil \log_2 n\rceil + c}}$. Unfortunately there is no unique pair in the initial dataword that can be used to append a single $a\dot{a}$ pair and so we must append $a\dot{a}$ pairs throughout the dataword which gives $a\dot{a}$ subwords that grow in multiple locations throughout the dataword. Half of these growing $a\dot{a}$ subwords are deleted on each of the $\lceil \log_2 n\rceil + c$ iterations so that after the last iteration only one subword of the form $(a\dot{a})^{2^{\lceil \log_2 n\rceil + c}}$ remains.

We now explain how our algorithm knows when it has carried out the $\lceil \log_2 n\rceil + c$ iterations needed to grow the $(a\dot{a})^{2^{\lceil \log_2 n\rceil + c}}$ word.

Each iteration involves 3 rounds of the dataword and marks every second unmarked $u\dot{u}$ pair by mapping it to $\not{u}\dot{\not{u}}$. See for example Figure 1 where on the first iteration pairs 2, 4, 6, 8 and 10 are marked, on the second iteration pairs 3, 7 and 11 are marked, and so on. Using the $\lceil \log_2 n\rceil + c$ parity pairs we can determine when we have completed $\lceil \log_2 n\rceil + c$ iterations of this marking process. On iteration k we mark the pairs $u_{2^k+1}\dot{u}_{2^k+1}, u_{3(2^k)+1}\dot{u}_{3(2^k)+1}, u_{5(2^k)+1}\dot{u}_{5(2^k)+1} \ldots u_{m(2^k)+1}\dot{u}_{m(2^k)+1}$ and so on each iteration exactly one parity pair $(u_{2^k+1}\dot{u}_{2^k+1})$ is marked. To see that this is the case it is sufficient to note that if at the start of iteration i the unmarked pairs are

$$u_1\dot{u}_1, \quad u_{2^i+1}\dot{u}_{2^i+1}, \quad u_{2(2^i)+1}\dot{u}_{2(2^i)+1}, \quad u_{3(2^i)+1}\dot{u}_{3(2^i)+1}, \quad \cdots \quad u_{s_1(2^i)+1}\dot{u}_{s_1(2^i)+1}$$

then at the start of iteration $i+1$ the unmarked pairs are

$$u_1\dot{u}_1,\ u_{2^{i+1}+1}\dot{u}_{2^{i+1}+1},\ u_{2(2^{i+1})+1}\dot{u}_{2(2^{i+1})+1},\ u_{3(2^{i+1})+1}\dot{u}_{3(2^{i+1})+1},\ \ldots\ u_{s_2(2^{k+1})+1}\dot{u}_{s_2(2^{k+1})+1}$$

and so pairs of the form $u_{2^k+1}\dot{u}_{2^k+1}, u_{3(2^k)+1}\dot{u}_{3(2^k)+1}, u_{5(2^k)+1}\dot{u}_{5(2^k)+1} \ldots u_{m(2^k)+1}\dot{u}_{m(2^k)+1}$ are marked on iteration k for $k = i$ and $k = i+1$. The value of each $u_{2^k+1}\dot{u}_{2^k+1}$ pair is set via Equation (4) so that an even number of $1\dot{1}$ pairs are marked during each of the

first $\lceil \log_2 n \rceil + c - 1$ iterations, and on iteration number $\lceil \log_2 n \rceil + c$ an odd number of $1\dot{1}$ pairs are marked. So by checking whether the number of $u\dot{u} = 1\dot{1}$ pairs marked during each iteration is odd or even our algorithm can determine when exactly $\lceil \log_2 n \rceil + c$ iterations have been completed.

Now we can explain how we append $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ during the $\lceil \log_2 n \rceil + c$ iterations described above. On the first iteration a single $a\dot{a}$ pair is appended to the left of each $u\dot{u}$ pair that remains unmarked by applying a rule of the form $u \to a\dot{a}\ u\dot{u}$. On each subsequent iteration each $a\dot{a}$ pair is replaced with two $a\dot{a}$ pairs if the $u\dot{u}$ pair immediately to the right remains unmarked and each $a\dot{a}$ pair is deleted by mapping it to the empty word if the $u\dot{u}$ pair immediately to the right is marked on that iteration. In addition on each iteration each unmarked $u\dot{u}$ pair adds another $a\dot{a}$ pair. So immediately to the left of each unmarked $u\dot{u}$ pair we have a single $a\dot{a}$ pair after the first iteration, 3 $a\dot{a}$ pairs after the second iteration, 7 $a\dot{a}$ pairs after the third iteration, and $2^k - 1$ $a\dot{a}$ pairs after the k^{th} iteration (see Figure 1). Thus after $\lceil \log_2 n \rceil + c$ iterations we have $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c} - 1}$ to the left of the only unmarked pair $(u_1 u_1)$. Then in one final round each $a\dot{a}$ in $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c} - 1}$ is mapped to $a\dot{a}$, and this lone remaining $u_1 u_1$ pair appends one further $a\dot{a}$ to give $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ at the right end of the dataword, while also producing the $\bar{u}\dot{\bar{u}}$ to simultaneously achieve Tasks (a) and (b).

Task (c)

From Equation (3) the parity pairs that appear in the dataword have the form $u_{2^k+1}\dot{u}_{2^k+1}$ and before we can delete these pairs we must first distinguish them from all other $u_i\dot{u}_i$ pairs in the dataword. Recall from Task (a) that on iteration k we mark the pairs $u_{2^k+1}\dot{u}_{2^k+1}$, $u_{3(2^k)+1}\dot{u}_{3(2^k)+1}$, $u_{5(2^k)+1}\dot{u}_{5(2^k)+1}$, $\ldots$ $u_{m(2^k)+1}\dot{u}_{m(2^k)+1}$. Note that $u_{2^k+1}\dot{u}_{2^k+1}$ is the left-most pair marked during iteration k, and since every second unmarked $u\dot{u}$ pair is marked during iteration k there must be a single unmarked pair to the left of $u_{2^k+1}\dot{u}_{2^k+1}$, and this unmarked pair must be $u_1\dot{u}_1$ since $u_1\dot{u}_1$ is never marked which also means that in all iterations following iteration k there is exactly one unmarked pair to the left of $\not{u}_{2^k+1}\dot{\not{u}}_{2^k+1}$. It is also the case that immediately following iteration k there must be at least $l \geqslant 2$ unmarked pairs to left of each pair of the form $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ where $j \geqslant 3$. It follows that after a further r iterations (iteration $k + r$) when we have continued marking every second unmarked pair there will be $\frac{\lceil l \rceil}{2^r}$ unmarked pairs to the left of $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$. Since $l \geqslant 2$ and our algorithm iterates until there is only one unmarked pair there is an iteration $k + r$ where $\frac{\lceil l \rceil}{2^r} = 1$ and $\frac{\lceil l \rceil}{2^{r-1}} = 2$. It follows that for all pairs of the form $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ where $j \geqslant 3$ there is at least one iteration where the number of unmarked pairs to the left of $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ is even at the beginning of the iteration. If at the beginning of an iteration the number of unmarked pairs to the left of a $\not{u}\dot{\not{u}}$ pair is even then we apply the rule $\not{u} \to \not{\not{u}}\dot{\not{u}}$ (see for example Figure 1). It follows that pairs of the form $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ where $j \geqslant 3$ will be changed

Table 1 Tag system rules where $u \in \{0, 1\}$. The left column specifies when the rules are used during the algorithm: during round 1, 2 or 3 of each iteration, or during the final round on the dataword.

round 1	$\underset{1}{u} \to \underset{2\,2\,2}{u\dot{u}\ddot{u}}$, $\underset{1}{a} \to \underset{2\,2}{a\dot{a}}$, $\not{\underset{1}{u}} \to \not{\underset{2\,2}{u\dot{u}}}$, $\not{\underset{1}{a}} \to \not{\underset{2\,2}{a\dot{a}}}$,
round 2	$\underset{2}{u} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3\,3}{u\dot{u}\ddot{u}}$, $\underset{2}{\dot{1}} \to \underset{3\,3}{1\dot{1}\ddot{1}}$, $\underset{2}{\dot{0}} \to \underset{3\,3\,3}{\emptyset\dot{\emptyset}\ddot{\emptyset}}$, $\underset{2}{\ddot{u}} \to \epsilon$, $\underset{2}{a} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3}{a\dot{a}}$, $\underset{2}{\dot{a}} \to \epsilon$, $\not{\underset{2}{u}} \to \not{\underset{3\,3}{u\dot{u}}}$, $\underset{2}{\dot{u}} \to \not{\underset{3\,3}{u\dot{u}}}$, $\not{\underset{2}{a}} \to \not{\underset{3\,3}{a\dot{a}}}$, $\not{\underset{2}{\dot{a}}} \to \not{\underset{3\,3}{a\dot{a}}}$,
round 3	$\underset{3}{u} \to \underset{1\,1}{u\dot{u}}$, $\underset{3}{\dot{u}} \to \underset{1\,1}{u\dot{u}}$, $\underset{3}{\ddot{u}} \to \epsilon$, $\underset{3}{a} \to \underset{1\,1}{a\dot{a}}$, $\underset{3}{\dot{a}} \to \underset{1\,1}{a\dot{a}}$, $\not{\underset{3}{u}} \to \not{\underset{1\,1}{u\dot{u}}}$, $\not{\underset{3}{\dot{u}}} \to \not{\underset{1\,1}{u\dot{u}}}$, $\not{\underset{3}{a}} \to \not{\underset{1\,1}{a\dot{a}}}$, $\not{\underset{3}{\dot{a}}} \to \not{\underset{1\,1}{a\dot{a}}}$
Final round	$\underset{1}{\dot{u}} \to \underset{*\,*}{a\dot{a}}\,\underset{4\,4}{d\,\bar{u}\dot{\bar{u}}}$, $\underset{1}{\dot{a}} \to \underset{*\,*}{a\dot{a}}$, $\not{\underset{1}{\dot{u}}} \to \epsilon$, $\not{\underset{1}{\dot{a}}} \to \underset{4\,4}{u\dot{u}}$, $d \to \epsilon$, $\underset{*}{\dot{a}} \to \underset{4\,4}{a\dot{a}}$

to $\not{\underset{j(2^k)+1}{\dot{u}}}\not{\underset{j(2^k)+1}{\ddot{u}}}$ and pairs of the form $\not{\underset{2^k+1}{u}}\not{\underset{2^k+1}{\dot{u}}}$ (parity pairs) will not be changed. So applying a rule to delete $\not{u}\not{\dot{u}}$ pairs after iteration $\lceil \log_2 n \rceil + c$ deletes all parity pairs from the dataword without deleting the $u_i\dot{u}_i = x_i\dot{x}_i$ pairs that appear in Equation (6).

Algorithm Details

From the lemma statement we begin with a word of the form

$$\underset{1}{u_1}z_1\ \underset{1}{u_2}z_2\ \underset{1}{u_3}z_3\ \underset{1}{u_4}z_4\ \cdots\ \underset{1}{u_{n+\lceil \log_2 n \rceil + c}}\,z_{n+\lceil \log_2 n \rceil + c} \tag{7}$$

Rules of the form $\underset{1}{u} \to \underset{2\,2\,2}{u\dot{u}\ddot{u}}$ are applied to Dataword (7) which after one round gives

$$\underset{2}{u_1}\underset{2}{\dot{u}_1}\underset{2}{\ddot{u}_1}\ \underset{2}{u_2}\underset{2}{\dot{u}_2}\underset{2}{\ddot{u}_2}\ \underset{2}{u_3}\underset{2}{\dot{u}_3}\underset{2}{\ddot{u}_3}\ \underset{2}{u_4}\underset{2}{\dot{u}_4}\underset{2}{\ddot{u}_4}\ \cdots\ \underset{2}{u_{n+\lceil \log_2 n \rceil + c}}\,\underset{2}{\dot{u}_{n+\lceil \log_2 n \rceil + c}}\,\underset{2}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}} \tag{8}$$

The next round on the dataword uses the rules $\{\underset{2}{u} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3\,3}{u\dot{u}\ddot{u}}, \underset{2}{\dot{0}} \to \underset{3\,3\,3}{\emptyset\dot{\emptyset}\ddot{\emptyset}}, \underset{2}{\dot{1}} \to \underset{3\,3}{1\dot{1}\ddot{1}}, \underset{2}{\ddot{u}} \to \epsilon\}$ to mark every second $\underset{2\,2\,2}{u\dot{u}\ddot{u}}$ triple in Dataword (8). Because 2-tag systems only read every second symbol, when $i = 1 \bmod 2$, symbols $\underset{2}{u_i}$ and $\underset{2}{\ddot{u}_i}$ are read applying the rules $\underset{2}{u} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3\,3}{u\dot{u}\ddot{u}}$ and $\underset{2}{\ddot{u}} \to \epsilon$ to append $\underset{3\,3}{a\dot{a}}\,\underset{3}{u_i}\underset{3}{\dot{u}_i}\underset{3}{\ddot{u}_i}$ (ϵ is the empty word), and when $i = 0 \bmod 2$, symbol $\underset{2}{\dot{u}_i}$ is read applying either the rule $\underset{2}{\dot{0}} \to \underset{3\,3\,3}{\emptyset\dot{\emptyset}\ddot{\emptyset}}$ or the rule $\underset{2}{\dot{1}} \to \underset{3\,3}{1\dot{1}\ddot{1}}$ to append $\not{\underset{3}{u_i}}\not{\underset{3}{\dot{u}_i}}(\not{\underset{3}{\ddot{u}_i}})$ (the $\not{\underset{3}{\ddot{u}_i}}$ symbol in brackets is present only if $u_i = 0$). So the above rules mark every second $u\dot{u}\ddot{u}$ triple to give a dataword of one of the following two forms

$$\underset{3\,3}{a\dot{a}}\ \underset{3}{u_1}\underset{3}{\dot{u}_1}\underset{3}{\ddot{u}_1}\ \not{\underset{3}{u_2}}\not{\underset{3}{\dot{u}_2}}(\not{\underset{3}{\ddot{u}_2}})\ \underset{3\,3}{a\dot{a}}\ \underset{3}{u_3}\underset{3}{\dot{u}_3}\underset{3}{\ddot{u}_3}\ \not{\underset{3}{u_4}}\not{\underset{3}{\dot{u}_4}}(\not{\underset{3}{\ddot{u}_4}})\ \cdots\ \not{\underset{3}{u_{n+\lceil \log_2 n \rceil + c}}}\,\not{\underset{3}{\dot{u}_{n+\lceil \log_2 n \rceil + c}}}\,(\not{\underset{3}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}}}) \tag{9}$$

$$\underset{3}{\dot{a}}\ \underset{3}{u_1}\underset{3}{\dot{u}_1}\underset{3}{\ddot{u}_1}\ \not{\underset{3}{u_2}}\not{\underset{3}{\dot{u}_2}}(\not{\underset{3}{\ddot{u}_2}})\ \underset{3\,3}{a\dot{a}}\ \underset{3}{u_3}\underset{3}{\dot{u}_3}\underset{3}{\ddot{u}_3}\ \not{\underset{3}{u_4}}\not{\underset{3}{\dot{u}_4}}(\not{\underset{3}{\ddot{u}_4}})\ \cdots\ \underset{3\,3}{a\dot{a}}\ \underset{3}{u_{n+\lceil \log_2 n \rceil + c}}\,\underset{3}{\dot{u}_{n+\lceil \log_2 n \rceil + c}}\,\underset{3}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}} \tag{10}$$

We get a dataword of the form given in (9) if $n + \lceil \log_2 n \rceil + c$ is even and we get a dataword of the form given in (10) if $n + \lceil \log_2 n \rceil + c$ is odd. To see this recall from the previous paragraph that if $n + \lceil \log_2 n \rceil + c$ is odd we read $\underset{0}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}}$, and when it is read it is deleted along with $\underset{3}{a_1}$ which is why $\underset{3}{a_1}$ does not appear at the left end of Dataword (10).

For the next round the rules $\{\underset{3}{u} \to \underset{1\,1}{u}\dot{\underset{}{u}}, \underset{3}{\dot{u}} \to \underset{1\,1}{u}\dot{\underset{}{u}}, \underset{3}{\ddot{u}} \to \epsilon, \underset{3}{\not{u}} \to \not{\underset{1\,1}{u}}\dot{\not{u}}, \underset{3}{\dot{\not{u}}} \to \underset{1\,1}{u}\dot{\underset{}{u}}, \underset{3}{\ddot{\not{u}}} \to \epsilon, \underset{3}{a} \to \underset{1\,1}{a}\dot{\underset{}{a}}, \underset{3}{\dot{a}} \to \underset{1\,1}{a}\dot{\underset{}{a}},\}$ are applied to Datawords (9) and (10) to give Datawords (11) and (12), respectively. Note that to produce datawords of the form given in (11) and (12), where the leftmost symbol is $\underset{1}{a}$ instead of $\underset{1}{\dot{a}}$, Datawords (9) and (10) must have even length. The parity of Datawords (9) and (10) depends on the number of $1\dot{1}\ddot{1}$ triples that are marked when reading Dataword (8), as a $1\dot{1}\ddot{1}$ triple is marked by replacing with a $\not{1}\dot{\not{1}}$ pair. So if we mark an even number of $1\dot{1}\ddot{1}$ triples, Dataword (9) will have the same parity as Dataword (8), and Dataword (10) will have a different parity from Dataword (8) (since Dataword (10) is missing the leftmost $\underset{3}{a}_1$). So because Dataword (9) is only produced if Dataword (8) is even and Dataword (10) is only produced if Dataword (8) is odd, Dataword (9) and (10) are both even. From the description of Task (a) we know that the number of $\underset{2\,2\,2}{1\dot{1}\ddot{1}}$ triples marked is even for the first $\lceil \log_2 n \rceil + c$ iterations of our algorithm and so it follows that the length of Datawords (9) and (10) are even. For this reason the leftmost symbol in Datawords (11) and (12) is $\underset{1}{a}$ and so undotted symbols are read during the next round.

$$\underset{1\,1}{a}\dot{\underset{}{a}}\ \underset{1\,1}{u_1}\dot{\underset{1}{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \underset{1\,1}{a}\dot{\underset{}{a}}\ \underset{1\,1}{u_3}\dot{\underset{1}{u}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ \cdots\ \not{\underset{1}{u}}_{n+\lceil \log_2 n \rceil + c}\dot{\not{\underset{1}{u}}}_{n+\lceil \log_2 n \rceil + c} \qquad \text{if } n + \lceil \log_2 n \rceil + c \text{ is even} \tag{11}$$

$$\underset{1\,1}{a}\dot{\underset{}{a}}\ \underset{1\,1}{u_1}\dot{\underset{1}{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \underset{1\,1}{a}\dot{\underset{}{a}}\ \underset{1\,1}{u_3}\dot{\underset{1}{u}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ \cdots\ \underset{1\,1}{a}\dot{\underset{}{a}}\ \underset{1}{u}_{n+\lceil \log_2 n \rceil + c}\dot{\underset{1}{u}}_{n+\lceil \log_2 n \rceil + c} \qquad \text{if } n + \lceil \log_2 n \rceil + c \text{ is odd} \tag{12}$$

In Datawords (11) and (12) the 2-tag system is ready to repeat the process of marking every second unmarked symbol. The cases for Datawords (11) and (12) proceed in a similar manner so we will continue only with the case for Dataword (11). Applying the rules $\underset{1}{u} \to \underset{2\,2\,2}{u}\dot{\underset{}{u}}\ddot{\underset{}{u}}$, $\not{\underset{1}{u}} \to \not{\underset{2\,2}{u}}\dot{\not{u}}$ and $\underset{1}{a} \to \underset{2\,2}{a}\dot{\underset{}{a}}$ to Dataword (11) gives

$$\underset{2\,2}{a}\dot{\underset{}{a}}\ \underset{2\,2}{u_1}\dot{\underset{2}{u}}_1\ddot{\underset{2}{u}}_1\ \not{\underset{2}{u}}_2\dot{\not{\underset{2}{u}}}_2\ \underset{2\,2}{a}\dot{\underset{}{a}}\ \underset{2\,2}{u_3}\dot{\underset{2}{u}}_3\ddot{\underset{2}{u}}_3\ \not{\underset{2}{u}}_4\dot{\not{\underset{2}{u}}}_4\ \underset{2\,2}{a}\dot{\underset{}{a}}\ \underset{2\,2}{u_5}\dot{\underset{2}{u}}_5\ddot{\underset{2}{u}}_5\ \cdots\ \not{\underset{2}{u}}_{n+\lceil \log_2 n \rceil + c}\dot{\not{\underset{2}{u}}}_{n+\lceil \log_2 n \rceil + c} \tag{13}$$

Continuing the computation, the rules used in the first iteration are used again here to mark every second $\underset{2\,2\,2}{u}\dot{\underset{}{u}}\ddot{\underset{}{u}}$ triple but on this iteration we also apply the rules $\{\not{\underset{2}{u}} \to \not{\underset{3\,3}{\not{u}}}\dot{\not{u}}, \dot{\not{\underset{2}{u}}} \to \not{\underset{3\,3}{u}}\dot{\not{u}}, \underset{2}{a} \to \underset{3\,3}{a}\dot{\underset{}{a}}\ \underset{3\,3}{a}\dot{\underset{}{a}}, \underset{2}{\dot{a}} \to \epsilon\}$ to Dataword (13) to produce Dataword (14). When reading Dataword (13) each triple $\underset{2\,2\,2}{u}\dot{\underset{}{u}}\ddot{\underset{}{u}}$ causes a shift in the reading frame where if we read the symbols with a single dot before a triple we will read the symbols with no dots after that triple. This means that if there is a even number of $\underset{2\,2\,2}{u}\dot{\underset{}{u}}\ddot{\underset{}{u}}$ triples to the left of a $\not{\underset{2\,2}{u}}\dot{\not{u}}$ pair we read $\not{\underset{2}{u}}$ and apply the rule $\not{\underset{2}{u}} \to \not{\underset{3\,3}{\not{u}}}\dot{\not{u}}$, and if there is an odd number we read $\dot{\not{\underset{2}{u}}}$ and apply the rule $\dot{\not{\underset{2}{u}}} \to \not{\underset{3\,3}{u}}\dot{\not{u}}$. This allows us to distinguish the parity pairs from all other pairs by changing as described in the paragraph on Task (c), since only the marked parity pairs have the form $\not{u}\dot{\not{u}}$ after the final iteration with all other marked pairs having the from $\not{\not{u}}\dot{\not{\not{u}}}$. Next we consider the change in the number of $a\dot{a}$ pairs when Dataword (13) is read to produce Dataword (14). If we read $\underset{2}{\dot{u}}$ in a $\underset{2\,i\,2}{u_i}\dot{\underset{2}{u}}_i\ddot{\underset{2}{u}}_i$ triple

it follows that we read $\underset{2}{\dot{a}}$ symbols in the run of $\underset{2}{a}\underset{2}{\dot{a}}$ pairs immediately to the left of that triple, and so when we mark a triple by applying the rule $\underset{2}{\dot{u}} \to \not{\underset{3}{u}}_2\dot{\not{\underset{3}{u}}}_2(\ddot{\not{\underset{3}{u}}}_2)$ the $\underset{2}{a}\underset{2}{\dot{a}}$ pairs are deleted because we apply the rule $\underset{2}{\dot{a}} \to \epsilon$ giving the behavior described in the third paragraph of Task (a). Alternatively, when we read $\underset{2}{u}$ the $\underset{3}{u}\underset{3}{\dot{u}}\underset{3}{\ddot{u}}$ triple remains unmarked and we read $\underset{2}{a}$ symbols in the run of $\underset{2}{a}\underset{2}{\dot{a}}$ pairs immediately to the left which applies the rule $\underset{2}{a} \to \underset{3}{a}\underset{3}{\dot{a}}\,\underset{3}{a}\underset{3}{\dot{a}}$ giving the described in the third paragraph of Task (a). Since we have covered the difference between reading even and odd numbers of unmarked symbols during the previous iteration, here we cover only the case where there is an even number of unmarked symbols in Dataword (13). So following a round on Dataword (13) we get a Dataword of the form given in (14).

$$(\underset{3}{a}\underset{3}{\dot{a}})^3\ \underset{3}{u}_1\underset{3}{\dot{u}}_1\underset{3}{\ddot{u}}_1\ \not{\underset{3}{u}}_2\dot{\not{\underset{3}{u}}}_2\ \not{\underset{3}{u}}_3\dot{\not{\underset{3}{u}}}_3(\ddot{\not{\underset{3}{u}}}_3)\ \not{\underset{3}{u}}_4\dot{\not{\underset{3}{u}}}_4\ (\underset{3}{a}\underset{3}{\dot{a}})^3\ \underset{3}{u}_5\underset{3}{\dot{u}}_5\underset{3}{\ddot{u}}_5\ \cdots\ \not{\underset{3}{u}}_{n+\lceil\log_2 n\rceil+c}\dot{\not{\underset{3}{u}}}_{n+\lceil\log_2 n\rceil+c} \tag{14}$$

From our explanation at the end of the previous iteration we know that a single round on Dataword (14) gives a dataword of the form

$$(\underset{1}{a}\underset{1}{\dot{a}})^3\ \underset{1}{u}_1\underset{1}{\dot{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \not{\underset{1}{u}}_3\dot{\not{\underset{1}{u}}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ (\underset{1}{a}\underset{1}{\dot{a}})^3\ \underset{1}{u}_5\underset{1}{\dot{u}}_5\ \cdots\ \not{\underset{1}{u}}_{n+\lceil\log_2 n\rceil+c}\dot{\not{\underset{1}{u}}}_{n+\lceil\log_2 n\rceil+c} \tag{15}$$

Each subsequent iteration carries on as described above until we come to iteration $\lceil\log_2 n\rceil + c$. From Task (a) we know that during iteration $\lceil\log_2 n\rceil + c$ we mark an odd number of $\underset{2}{1}\dot{\underset{2}{1}}\ddot{\underset{2}{1}}$ triples and so from the paragraph preceding Dataword (9) this means that the leftmost $\underset{1}{a}$ gets deleted at the end of iteration $\lceil\log_2 n\rceil + c$ to give a Dataword of the form (16). From the description of Task (a) we know that after $\lceil\log_2 n\rceil + c$ iterations $\underset{1}{u}_1\underset{1}{\dot{u}}_1$ is the only unmarked pair in the dataword and that the only $a\dot{a}$ pairs to be found are immediately to the left of $\underset{1}{u}_1\underset{1}{\dot{u}}_1$ in a word of the form $(\underset{1}{a}\underset{1}{\dot{a}})^{2^{n+\lceil\log_2 n\rceil+c}-1}$ as shown in Dataword (16).

$$\underset{1}{\dot{a}}(\underset{1}{a}\underset{1}{\dot{a}})^{2^{n+\lceil\log_2 n\rceil+c}-2}\ \underset{1}{u}_1\underset{1}{\dot{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \not{\underset{1}{u}}_3\dot{\not{\underset{1}{u}}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ \not{\underset{1}{u}}_5\dot{\not{\underset{1}{u}}}_5\ \cdots\ \not{\underset{1}{u}}_{n+\lceil\log_2 n\rceil+c}\dot{\not{\underset{1}{u}}}_{n+\lceil\log_2 n\rceil+c} \tag{16}$$

In Dataword (16) following $\lceil\log_2 n\rceil + c$ iterations we read the dotted symbol for pairs with underscript 1 for the first time and we apply the rules $\{\underset{1}{\dot{u}} \to \underset{*}{a}\underset{*}{\dot{a}}\,d\,\underset{4}{\bar{u}}\underset{4}{\dot{\bar{u}}},\ \underset{1}{\dot{a}} \to \underset{*}{a}\underset{*}{\dot{a}},\ \dot{\not{\underset{3}{u}}} \to \epsilon,\ \dot{\not{\underset{3}{u}}} \to \underset{4}{u}\underset{4}{\dot{u}},\ d \to \epsilon\}$ to give Dataword (17). From the description of Task (c) we know that rule $\dot{\not{\underset{3}{u}}} \to \epsilon$ deletes all pairs of the form $\not{\underset{4}{u}}_{2^k+1}\dot{\not{\underset{4}{u}}}_{2^k+1}$ and so from Equation (3) Datawords (17) and (6) are identical.

$$\underset{4}{\bar{u}}_1\underset{4}{\dot{\bar{u}}}_1\ \underset{4}{u}_4\underset{4}{\dot{u}}_4\ \underset{4}{u}_6\underset{4}{\dot{u}}_6\ \underset{4}{u}_7\underset{4}{\dot{u}}_7\ \underset{4}{u}_8\underset{4}{\dot{u}}_8\ \underset{4}{u}_{10}\underset{4}{\dot{u}}_{10}\ \cdots\ \underset{4}{u}_{n+\lceil\log_2 n\rceil+c}\underset{4}{\dot{u}}_{n+\lceil\log_2 n\rceil+c}(\underset{4}{a}\underset{4}{\dot{a}})^{2^{n+\lceil\log_2 n\rceil+c}} \tag{17}$$

◀

We can now use Lemma 11 to prove Corollary 12 which will be used in the reduction in our main theorem. In this reduction our tag systems simulate a particular type of Turing machine where the halting time is bounded by what is known as a longevity guard. A longevity guard [8] for a Turing machine M is a function $l : \Sigma^* \to \mathbb{N}$ where on input w either M halts in $\leqslant l(w)$ steps or it runs forever.

▶ **Corollary 12.** *Let M be an arbitrary nondeterministic Turing machine with a single binary tape and a longevity guard l, let f be the function given by Equation (3), and let s be an arbitrary binary word of length $|w| + \lceil\log_2 |w|\rceil + c$ where $c = 1$ if $2^{\lceil\log_2 |w|\rceil} < n + \lceil\log_2 |w|\rceil$ otherwise $c = 0$. There is a nondeterministic 2-tag system that takes words of the form $f(w, s)$, and halts in time $O(l(w)^2 \log l(w))$ if and only if M halts on input w in time $l(w)$.*

Proof. The deterministic tag system algorithm in [15] simulates a Turing machine transition rule q_y, x_1, x_2, d, q_z (with current state q_y, read symbol x_1, write symbol x_2, move direction $d \in \{L, R\}$, and next state q_z) using a rule of the form $\underset{y}{x_1}\dot{\underset{y}{x}}_1 \to \not{\underset{0}{x}}_2\dot{\not{\underset{0}{x}}}_2 s^{2^z}$ if $d = R$, and a rule of the form $\underset{y}{x_1}\dot{\underset{y}{x}}_1 \to s^{2^z}\underset{0}{x_2}\dot{\underset{0}{x}}_2$ if $d = L$. In our nondeterministic 2-tag system there is a rule of one of the two forms given above for each transition rule in the nondeterministic Turing machine M. Such a nondeterministic 2-tag system uses the same algorithm as the tag system in [15] and so simulates t steps of M in time $O(l(w)^2 \log l(w))$ (see Theorem 2) halting if and only if M halts. Adding the 2-tag system from Lemma 11 as a subroutine to such a nondeterministic 2-tag system allows it to simulate M in time $O(l(w)^2 \log l(w))$ when given $f(w, s)$ as input. ◀

To see that the longevity guard is necessary in Corollary 12 consider what happens if we replace $l(w)$ with an arbitrary running time t: Our 2-tag system still halts in time $O(t^2 \log t)$ if M halts in time t, however it is also possible that our tag system halts in time $O(t^2 \log t)$ when M does not halt in time t but does halt at some later time $> t$. This is because the asymptotic bound $O(t^2 \log t)$ does not bound precisely the number of simulated Turing machine steps at t. So without the longevity guard we get the "if" but not the "only if".

▶ **Theorem 13.** *The distributional bounded halting problem for nondeterministic 2-tag systems is complete for average-case NP under Ptime randomized reductions.*

Proof. We show that for every distributional problem (D, μ) in NP, there is a Ptime randomized reduction from (D, μ) to the distributional bounded halting problem for nondeterministic 2-tag systems that satisfies Definition 8.

In [8] Gurevich shows that for each NP distributional problem (D, μ) there exists a nondeterministic binary Turing machine M_i that has a polynomial longevity guard l such that (D, μ) Ptime reduces to the halting problem for M_i. Given a binary input word w that encodes an instance v of D, M_i halts in $\leqslant l(w)$ steps if and only if $v \in L_D$. From Definition 9 a Ptime function g that reduces instances of D to instances of the bounded halting problem for M_i is given by

$$g(v) = Pf(\langle i \rangle, w, 1^{l(w)}) \tag{18}$$

Let Δ be a dilation of (D, μ) to (D_Δ, μ_Δ) where each instance $v \in D$ is mapped to a set of the form $\{Pf(v, s_v) | s_v \in \{0,1\}^{|w|+\lceil \log_2 |w| \rceil + c}\}$, with $c = 1$ if $2^{\lceil \log_2 |w| \rceil} < |w| + \lceil \log_2 |w| \rceil$ otherwise $c = 0$. Since g is Ptime computable we know from Equation (18) that given v the value $|w| + \lceil \log_2 |w| \rceil + c$ can be computed in polynomial time and so there is a randomized algorithm that computes Δ giving a Ptime dilation. In addition Δ is also nonrare as $R_\Delta = 1$ (see Definition 7). So from Definition 8 we can show that (D, μ) Ptime randomly reduces to the bounded halting problem for 2-tag systems by showing that its nonrare Ptime dilation (D_Δ, μ_Δ) Ptime reduces to the bounded halting problem for 2-tag systems.

From Definition 6 and the paragraph before Equation (18) we see that for all $Pf(v, s_w) \in D_\Delta$ we have $Pf(v, s_w) \in L_{D_\Delta}$ if and only if M_i halts in time $l(w)$ on input w. It follows from Corollary 12 that there is a tag system T_j such that for all $Pf(v, s_w) \in D_\Delta$, T_j halts in time $O(l(w)^2 \log l(w))$ on input $f(v, s_w)$ if and only if $Pf(v, s_w) \in D_\Delta$. So there is a reduction g' that reduces instance of D_Δ to instances of the bounded halting problem for T_j (Definition 10). The reduction g' which is given in Equation (19) satisfies condition 1 of Definition 5.

$$g'(Pf(v, s_w)) = Pf(\langle j \rangle, f(w, s_w), 1^{O(l(w)^2 \log l(w))}) \tag{19}$$

Since $g(v)$ in Equation (18) is Ptime computable so too is $g'(Pf(v, s_w))$. Now to complete our proof it only remains to show that the reduction g' given in Equation (19) satisfies condition 2 of Definition 5.

We already know that the reduction g given in Equation (18) Ptime reduces the distributional problem (D, μ) to the bounded halting problem for M_i, and so it follows from condition 2 of Definition 5 and the probability distribution in Definition 9 that there is a polynomial p_1, such that

$$\mu(v) \;<\; p_1(|v|)2^{-(|\langle i\rangle|+|w|)}|\langle i\rangle|^{-2}|w|^{-2}l(w)^{-2} \tag{20}$$

For simplicity we rewrite Equation (21) as

$$\mu(v) \;<\; p_1(v)2^{-|w|} \tag{21}$$

From Equation (3) we get $|f(w, s_w)| = 2|w| + 2\log_2|w|$ and so from Definition 10 the probability of getting an instance $Pf(\langle j\rangle, f(w, s_w), 1^{O(l(w)^2 \log l(w))})$ is proportional to

$$\mu(\langle j\rangle, f(w, s_w), O(l(w)^2 \log l(w))) = \\ 2^{-(|\langle j\rangle|+2|w|+2\log_2|w|)}|\langle j\rangle|^{-2}(2|w| + 2\log_2|w|)^{-2}(l(w)^2 \log l(w))^{-2} \tag{22}$$

The value $|\langle j\rangle|$ is a constant independent of $|v|$ and the values $|w|$ and $l(w)$ are polynomial in $|v|$ and so there is a polynomial p_2 such that $p_2(|v|)^{-1} < 2^{(-(|\langle j\rangle|+2\log_2|w|)}|\langle j\rangle|^{-2}(2|w| + 2\log_2|w|)^{-2}(l(w)^2 \log l(w))^{-2}$ which substitutes into Equation (22) to give

$$\mu(\langle j\rangle, f(w, s_w), O(l(w)^2 \log l(w))) \;>\; p_2(v)^{-1}2^{-2|w|} \tag{23}$$

Recall that $|w_s| = |w|+\log_2|w|$ and so from Definition 6 an instance of D_Δ given by $Pf(v, w_s)$ has a probability proportional to

$$\mu_\Delta(Pf(v, w_s)) \;=\; \mu(v)2^{-(|w|+\log_2|w|)} \tag{24}$$

From Equations (21), (23) and (24) we get Equation (25) which shows that the polynomial $p_1(|v|)p_2(|v|)$ satisfies condition 2 of Definition 5 for the reduction g' in Equation (19).

$$\mu_\Delta(Pf(v, w_s)) \;<\; p_1(|v|)p_2(v)\mu(\langle j\rangle, f(w, s_w), O(l(w)^2 \log l(w))) \tag{25}$$

◀

4 An example of the decoding process, with paired notation

In this section we show a diagram of an example of the decoding process, using paired notation.

Each letter X in paired notation represents a pair of letters X_0X_1 in the tag system, of which only one will be read on the next round. Which one? That depends on the parity of the reading frame when it gets read. Often this parity is not known when the pair X_0X_1 is written.

Sometimes a third symbol λ is written to the tape, just for the purpose of changing the parity. If read, its rule appends an empty appendant. So it doesn't matter if it gets read or not.

In our paired notation, such a parity-changer is written as $\big|$ as in Figure 2.

The parity with which a pair is read is indicated by a thick line either over or under the symbol, as in Figure 2. So the vertical parts of the thick line are written by the previous line, while the horizontal parts simply alternate between the top and the bottom whenever there is a vertical part. They continue at the top (or bottom) from one line to the next.

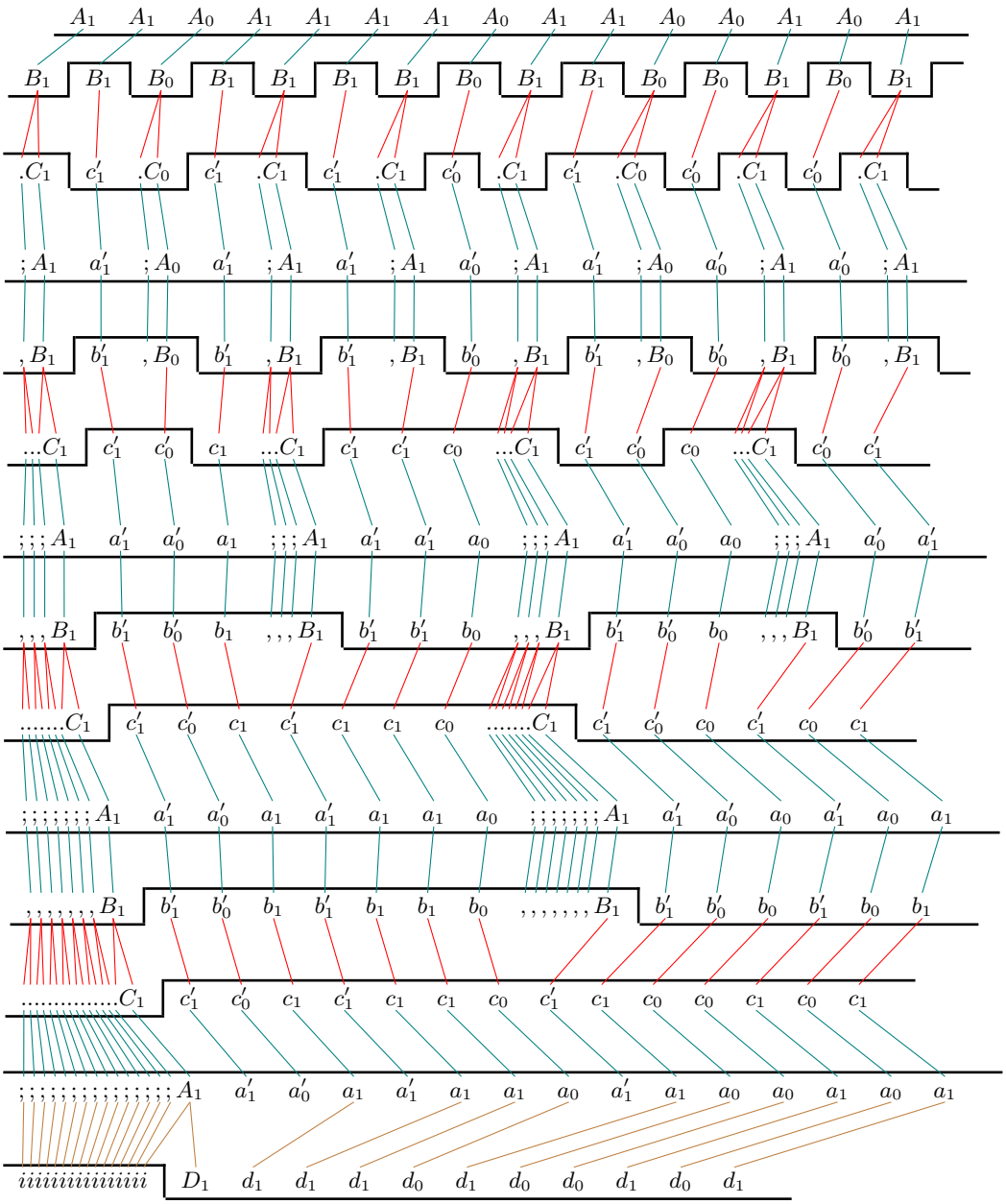


Figure 2 Diagram of an example of the decoding process. Each symbol here represents two symbols on the tag system tape, and the line passing over or under the symbols indicates the reading frame parity. The colored lines indicate how the symbols are transformed on each round. The bits marked with primes and eliminated in the final round are (if you trace them back to the initial row) exactly the extra bits used for the encoding. An additional partial round (not shown) over just the i symbols will yield the form guaranteed by Lemma 11: A row of bits, the first of which is uniquely marked, followed by a "counter" whose length is a power of two and which is at least as large as the row of bits, all being read in "plain" parity.

Table 2 The complete rules for the tag system. Paired notation is shown on the left; traditional notation is shown on the right. The initial tape would use either $\underset{1}{0}$ or $\underset{1}{1}$ as the second symbol in each A_i pair (not necessarily even matching the first symbol of the pair), but ever after the first round the second symbols of the A_i pairs will be as shown here. Since these input symbols will not be read, they make no difference. At the end, after a final partial round over just the $\underset{* *}{a\dot{a}}$ symbols, the correspondence of symbols in the final tape is $D_0 = \underset{4\,4}{\bar{0}\dot{\bar{0}}}$, $D_1 = \underset{4\,4}{\bar{1}\dot{\bar{1}}}$, $d_0 = \underset{4\,4}{0\dot{0}}$, $d_1 = \underset{4\,4}{1\dot{1}}$, and $e = \underset{4\,4}{a\dot{a}}$, and the tape is exactly the target dataword specified by Lemma 11.

symbol in diagram	*rule for plain parity*	*rule for flipped parity*	*corresponding original symbols and rules*		
A_0	$\underline{A_0} \longrightarrow B_0\vert$	$\overline{A_0} \longrightarrow i\vert D_0$	$\underset{1\,1}{0\dot{0}}$	$\underset{1}{0} \to \underset{2\,2\,2}{0\dot{0}\ddot{0}}$	$\underset{1}{\dot{0}} \to \underset{* *}{a\dot{a}}\, d\, \underset{4\,4}{\bar{0}\dot{\bar{0}}}$
A_1	$\underline{A_1} \longrightarrow B_1\vert$	$\overline{A_1} \longrightarrow i\vert D_1$	$\underset{1\,1}{1\dot{1}}$	$\underset{1}{1} \to \underset{2\,2\,2}{1\dot{1}\ddot{1}}$	$\underset{1}{\dot{1}} \to \underset{* *}{a\dot{a}}\, d\, \underset{4\,4}{\bar{1}\dot{\bar{1}}}$
a'_0	$\underline{a'_0} \longrightarrow b'_0$	$\overline{a'_0} \longrightarrow$	$\underset{1\,1}{\not{0}\dot{\not{0}}}$	$\underset{1}{\not{0}} \to \underset{2\,2}{\not{0}\dot{\not{0}}}$	$\underset{1}{\dot{\not{0}}} \to$
a'_1	$\underline{a'_1} \longrightarrow b'_1$	$\overline{a'_1} \longrightarrow$	$\underset{1\,1}{\not{1}\dot{\not{1}}}$	$\underset{1}{\not{1}} \to \underset{2\,2}{\not{1}\dot{\not{1}}}$	$\underset{1}{\dot{\not{1}}} \to$
a_0	$\underline{a_0} \longrightarrow b_0$	$\overline{a_0} \longrightarrow d_0$	$\underset{1\,1}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{1}{\not{\not{0}}} \to \underset{2\,2}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{1}{\dot{\not{\not{0}}}} \to \underset{4\,4}{0\dot{0}}$
a_1	$\underline{a_1} \longrightarrow b_1$	$\overline{a_1} \longrightarrow d_1$	$\underset{1\,1}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{1}{\not{\not{1}}} \to \underset{2\,2}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{1}{\dot{\not{\not{1}}}} \to \underset{4\,4}{1\dot{1}}$
;	$\underline{;} \longrightarrow ,$	$\overline{;} \longrightarrow i$	$\underset{1\,1}{a\dot{a}}$	$\underset{1}{a} \to \underset{* *}{a\dot{a}}$	$\underset{1}{\dot{a}} \to \underset{* *}{a\dot{a}}$
B_0	$\underline{B_0} \longrightarrow .C_0\vert$	$\overline{B_0} \longrightarrow c'_0\vert$	$\underset{2\,2}{0\dot{0}}$	$\underset{2}{0} \to \underset{3\,3\,3\,3\,3}{a\dot{a}0\dot{0}\ddot{0}}$	$\underset{2}{\dot{0}} \to \underset{3\,3\,3}{\not{0}\dot{\not{0}}\ddot{\not{0}}}$
B_1	$\underline{B_1} \longrightarrow .C_1\vert$	$\overline{B_1} \longrightarrow c'_1$	$\underset{2\,2}{1\dot{1}}$	$\underset{2}{1} \to \underset{3\,3\,3\,3\,3}{a\dot{a}1\dot{1}\ddot{1}}$	$\underset{2}{\dot{1}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$
b'_0	$\underline{b'_0} \longrightarrow c_0$	$\overline{b'_0} \longrightarrow c'_0$	$\underset{2\,2}{\not{0}\dot{\not{0}}}$	$\underset{2}{\not{0}} \to \underset{3\,3}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{2}{\dot{\not{0}}} \to \underset{3\,3}{\not{0}\dot{\not{0}}}$
b'_1	$\underline{b'_1} \longrightarrow c_1$	$\overline{b'_1} \longrightarrow c'_1$	$\underset{2\,2}{\not{1}\dot{\not{1}}}$	$\underset{2}{\not{1}} \to \underset{3\,3}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{2}{\dot{\not{1}}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$
b_0	$\underline{b_0} \longrightarrow c_0$	$\overline{b_0} \longrightarrow c_0$	$\underset{2\,2}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{2}{\not{\not{0}}} \to \underset{3\,3}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{2}{\dot{\not{\not{0}}}} \to \underset{3\,3}{\not{\not{0}}\dot{\not{\not{0}}}}$
b_1	$\underline{b_1} \longrightarrow c_1$	$\overline{b_1} \longrightarrow c_1$	$\underset{2\,2}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{2}{\not{\not{1}}} \to \underset{3\,3}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{2}{\dot{\not{\not{1}}}} \to \underset{3\,3}{\not{\not{1}}\dot{\not{\not{1}}}}$
,	$\underline{,} \longrightarrow ..$	$\overline{,} \longrightarrow$	$\underset{2\,2}{a\dot{a}}$	$\underset{2}{a} \to \underset{3\,3\,3\,3}{a\dot{a}a\dot{a}}$	$\underset{2}{\dot{a}} \to$
C_0	$\underline{C_0} \longrightarrow A_0$	$\overline{C_0} \longrightarrow A_0$	$\underset{3\,3}{0\dot{0}}$	$\underset{3}{0} \to \underset{1\,1}{0\dot{0}}$	$\underset{3}{\dot{0}} \to \underset{1\,1}{0\dot{0}}$
C_1	$\underline{C_1} \longrightarrow A_1$	$\overline{C_1} \longrightarrow A_1$	$\underset{3\,3}{1\dot{1}}$	$\underset{3}{1} \to \underset{1\,1}{1\dot{1}}$	$\underset{3}{\dot{1}} \to \underset{1\,1}{1\dot{1}}$
c'_0	$\underline{c'_0} \longrightarrow a'_0$	$\overline{c'_0} \longrightarrow a'_0$	$\underset{3\,3}{\not{0}\dot{\not{0}}}$	$\underset{3}{\not{0}} \to \underset{1\,1}{\not{0}\dot{\not{0}}}$	$\underset{3}{\dot{\not{0}}} \to \underset{1\,1}{\not{0}\dot{\not{0}}}$
c'_1	$\underline{c'_1} \longrightarrow a'_1$	$\overline{c'_1} \longrightarrow a'_1$	$\underset{3\,3}{\not{1}\dot{\not{1}}}$	$\underset{3}{\not{1}} \to \underset{1\,1}{\not{1}\dot{\not{1}}}$	$\underset{3}{\dot{\not{1}}} \to \underset{1\,1}{\not{1}\dot{\not{1}}}$
c_0	$\underline{c_0} \longrightarrow a_0$	$\overline{c_0} \longrightarrow a_0$	$\underset{3\,3}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{3}{\not{\not{0}}} \to \underset{1\,1}{\not{\not{0}}\dot{\not{\not{0}}}}$	$\underset{3}{\dot{\not{\not{0}}}} \to \underset{1\,1}{\not{\not{0}}\dot{\not{\not{0}}}}$
c_1	$\underline{c_1} \longrightarrow a_1$	$\overline{c_1} \longrightarrow a_1$	$\underset{3\,3}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{3}{\not{\not{1}}} \to \underset{1\,1}{\not{\not{1}}\dot{\not{\not{1}}}}$	$\underset{3}{\dot{\not{\not{1}}}} \to \underset{1\,1}{\not{\not{1}}\dot{\not{\not{1}}}}$
.	$\underline{.} \longrightarrow ;$	$\overline{.} \longrightarrow ;$	$\underset{3\,3}{a\dot{a}}$	$\underset{3}{a} \to \underset{1\,1}{a\dot{a}}$	$\underset{3}{\dot{a}} \to \underset{1\,1}{a\dot{a}}$
i	$\underline{i} \longrightarrow e$	$\overline{i} \longrightarrow e$	$\underset{* *}{a\dot{a}}$	$\underset{*}{a} \to \underset{4\,4}{a\dot{a}}$	$\underset{*}{\dot{a}} \to \underset{4\,4}{a\dot{a}}$

The production rules in this notation therefore need to include the thick line either over or under the symbol on the left hand side, but on the right hand side only vertical thick lines can appear, no horizontal ones.

The correspondence between the symbols used in Figure 2 (which are compressed due to space constraints in the figure) and the symbols used in the paper is given in Table 2.

References

1 Andreas Blass and Yuri Gurevich. Matrix Transformation Is Complete for the Average Case. *SIAM Journal on Computing*, 24(1):3–29, 1995. doi:10.1137/S0097539792232070.

2 Vincent D Blondel and John N Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36:1249–1274, 2000. doi:10.1016/S0005-1098(00)00050-9.

3 Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. *CoRR*, abs/cs/0606037, 2006. arXiv:cs/0606037.

4 Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004. URL: http://www.complex-systems.com/abstracts/v15_i01_a01.html.

5 Matthew Cook and Turlough Neary. A New Proof of Average Case NP-Completeness for the Post Correspondence Problem. In preparation.

6 Matthew Cook and Turlough Neary. Universality and Average-Case NP-Completeness in 2D Collatz Functions and Piecewise Affine Functions. In preparation.

7 Jens Eisert, Markus P Müller, and Christian Gogolin. Quantum measurement occurrence is undecidable. *Physical Review Letters*, 108(26):260501, 2012. doi:10.1103/PhysRevLett.108.260501.

8 Yuri Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42:346–398, 1991. doi:10.1016/0022-0000(91)90007-R.

9 Tero Harju and Maurice Margenstern. Splicing systems for universal Turing machines. In *DNA Computing, 10th International Workshop on DNA Computing(2004)*, volume 3384 of *Lecture Notes in Computer Science*, pages 149–158. Springer, 2005. doi:10.1007/11493785_13.

10 Pascal Koiran and Cristopher Moore. Closed-form analytic maps in one and two Dimensions can simulate universal Turing machines. *Theoretical Computer Science*, 210(1):217–223, 1999. doi:10.1016/S0304-3975(98)00117-0.

11 Leonid Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–1286, 1986. doi:10.1137/0215020.

12 Kristian Lindgren and Mats G. Nordahl. Universal Computation in Simple One-Dimensional Cellular Automata. *Complex Systems*, 4(3):299–318, 1990. URL: http://www.complex-systems.com/abstracts/v04_i03_a04.html.

13 Yuri Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. *Theoretical Computer Science*, 330(2):145–169, 2005. doi:10.1016/j.tcs.2004.09.016.

14 Marvin Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory, Symposium in Pure Mathematics*, volume 5, pages 229–238, Provelence, 1962. AMS.

15 Turlough Neary. *Small universal Turing machines*. PhD thesis, Department of Computer Science, National University of Ireland, Maynooth, 2008.

16 Turlough Neary. Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 30 of *LIPIcs*, pages 649–661, 2015. doi:10.4230/LIPIcs.STACS.2015.649.

17 Turlough Neary and Damien Woods. ¶-completeness of cellular automaton Rule 110. In *International Colloquium on Automata, Languages and Programming 2006, (ICALP) Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2006. doi:10.1007/11786986_13.

18 Emil Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943. doi:10.2307/2371809.

19 Raphael Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971. doi:10.1007/BF01418780.

20 Yurii Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996. doi:10.1016/S0304-3975(96)00077-1.

21 Yurii Rogozhin and Sergey Verlan. On the rule complexity of universal tissue P systems. In *Sixth international Workshop on Membrane Computing(2005)*, volume 3850 of *Lecture Notes in Computer Science*, pages 356–362. Springer, 2006. doi:10.1007/11603047_24.

22 Paul Rothemund. A DNA and restriction enzyme implementation of Turing Machines. In *DNA Based Computers: Proceeding of a DIMACS Workshop*, volume 2055, pages 75–119. AMS, 1996. URL: https://authors.library.caltech.edu/27384/.

23 Hava Siegelmann and Maurice Margenstern. Nine switch-affine neurons suffice for Turing universality. *Neural Networks*, 12:593–600, 1999. doi:10.1016/S0893-6080(99)00025-8.

24 Hava Siegelmann and Eduardo Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995. doi:10.1006/jcss.1995.1013.

25 Jie Wang. Average-case computational complexity theory. In Lane A Hemaspaandra and Alan L Selman, editors, *Complexity theory retrospective II*, pages 295–328. Springer-Verlag, 1998.

26 Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *In 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 132–143, Berkeley, California, October 2006. IEEE. doi:10.1109/FOCS.2006.58.

27 Adam Yedidia and Scott Aaronson. A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory. *Complex Systems*, 25(4):297–237, 2016. URL: http://www.complex-systems.com/abstracts/v25_i04_a04.html.

Pairwise Preferences in the Stable Marriage Problem

Ágnes Cseh
Institute of Economics, Centre for Economic and Regional Studies, Hungarian Academy of Sciences, 1097 Budapest, Tóth Kálmán u. 4., Hungary
cseh.agnes@krtk.mta.hu

Attila Juhos
Department of Computer Science and Information Theory, Budapest University of Technology and Economics, 1117 Budapest, Magyar Tudósok krt. 2., Hungary
juhosattila@cs.bme.hu

Abstract

We study the classical, two-sided stable marriage problem under pairwise preferences. In the most general setting, agents are allowed to express their preferences as comparisons of any two of their edges and they also have the right to declare a draw or even withdraw from such a comparison. This freedom is then gradually restricted as we specify six stages of orderedness in the preferences, ending with the classical case of strictly ordered lists. We study all cases occurring when combining the three known notions of stability – weak, strong and super-stability – under the assumption that each side of the bipartite market obtains one of the six degrees of orderedness. By designing three polynomial algorithms and two NP-completeness proofs we determine the complexity of all cases not yet known, and thus give an exact boundary in terms of preference structure between tractable and intractable cases.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases stable marriage, intransitivity, acyclic preferences, poset, weakly stable matching, strongly stable matching, super stable matching

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.21

Related Version All missing proofs can be found in the full version of the paper [9], `https://arxiv.org/abs/1810.00392`.

Funding This work was supported by the Cooperation of Excellences Grant (KEP-6/2018), by the Ministry of Human Resources under its New National Excellence Programme (ÚNKP-18-4-BME-331 and ÚNKP-18-1-I-BME-309), the Hungarian Academy of Sciences under its Momentum Programme (LP2016-3/2016), its János Bolyai Research Fellowship, and OTKA grant K128611.

Acknowledgements The authors thank Tamás Fleiner, David Manlove, and Dávid Szeszlér for fruitful discussions on the topic.

1 Introduction

In the 2016 USA Presidential Elections, polls unequivocally reported Democratic presidential nominee Bernie Sanders to be more popular than Republican candidate Donald Trump [33, 34]. However, Sanders was beaten by Clinton in their own party's primary election cycle, thus the 2016 Democratic National Convention endorsed Hillary Clinton to be the Democrat's candidate. In the Presidential Elections, Trump defeated Clinton. This recent example demonstrates well how inconsistent pairwise preferences can be.

Preferences play an essential role in the stable marriage problem and its extensions. In the classical setting [13], each man and woman expresses their preferences on the members of the opposite gender by providing a strictly ordered list. A set of marriages is stable if no

pair of agents blocks it. A man and a woman form a blocking pair if they mutually prefer one another to their respective spouses.

Requiring strict preference orders in the stable marriage problem is a strong assumption, which rarely suits real world scenarios [4]. The study of less restrictive preference structures has been flourishing [3, 10, 18, 22, 24, 27] for decades. As soon as one allows for ties in preference lists, the definition of a blocking edge needs to be revisited. In the literature, three intuitive definitions are used, each of which defines weakly, strongly and super stable matchings. According to weak stability, a matching is blocked by an edge uw if agents u and w both strictly prefer one another to their partners in the matching. A strongly blocking edge is preferred strictly by one end vertex, whereas it is not strictly worse than the matching edge at the other end vertex. A blocking edge is at least as good as the matching edge for both end vertices in the super stable case. Super stable matchings are strongly stable and strongly stable matchings are weakly stable by definition.

Weak stability is an intuitive notion that is most aligned with the classical blocking edge definition in the model defined by Gale and Shapley [13]. However, reaching strong stability is the goal to achieve in many applications, such as college admission programs. In most countries, students need to submit a strict ordering in the application procedure, but colleges are not able to rank all applicants strictly, hence large ties occur in their lists. According to the equal treatment policy used in Chile and Hungary for example, it may not occur that a student is rejected from a college preferred by him, even though other students with the same score are admitted [5, 30]. Other countries, such as Ireland [7], break ties with lottery, which gives way to a weakly stable solution. Super stable matchings are admittedly less relevant in applications, however, they represent worst-case scenarios if uncertain information is given about the agents' preferences. If two edges are incomparable to each other due to incomplete information derived from the agent, then it is exactly the notion of a super stable matching that guarantees stability, no matter what the agent's true preferences are.

The goal of our present work is to investigate the three cases of stability in the presence of more general preference structures than ties.

1.1 Related work

It is an empirical fact that cyclic and intransitive preferences often emerge in the broad topic of voting and representation, if the set of voters differs for some pairwise comparisons [2], such as in our earlier example with the polls on the Clinton–Sanders–Trump battle. Preference aggregation is another field that often yields intransitive group preferences, as the famous Condorcet-paradox [8] also states.

It might be less known that nontrivial preference structures naturally emerge in the preferences of individuals as well. The study of cyclic and intransitive preferences of a person has been triggering scientists from a wide range of fields for decades. Blavatsky [6] demonstrated that in choice situations under risk, the overwhelming majority of individuals expresses intransitive choice and violation of standard consistency requirements. Humphrey [16] found that cyclic preferences persist even when the choice triple is repeated for the second time. Using MRI scanners, neuroscientists identified brain regions encoding 'local desirability', which led to clear, systematic and predictable intransitive choices of the participants of the experiment [23]. Cyclic and intransitive preferences occur naturally in multi-attribute comparisons [11, 29]. May [29] studied the choice on a prospective partner and found that a significant portion of the participants expressed the same cyclic preference relations if candidates lacking exactly one of the three properties intelligence, looks, and wealth were offered at pairwise comparisons. In this paper, we investigate the stable marriage problem

equipped with the ubiquitous and well-studied preference structures of pairwise preferences that might be intransitive or cyclic.

Regarding the stable marriage problem, all three notions of stability have been thoroughly investigated if preferences are given in the form of a partially ordered set, a list with ties or a strict list [13, 18, 22, 24, 27, 28]. Weakly stable matchings always exist and can be found in polynomial time [27], and a super stable matching or a proof for its non-existence can also be produced in polynomial time [18, 28]. The most sophisticated ideas are needed in the case of strong stability, which turned out to be solvable in polynomial time if both sides have tied preferences [18]. Irving [18] remarked that "Algorithms that we have described can easily be extended to the more general problem in which each person's preferences are expressed as a partial order. This merely involves interpreting the 'head' of each person's (current) poset as the set of source nodes, and the 'tail' as the set of sink nodes, in the corresponding directed acyclic graph." Together with his coauthors, he refuted this statement for strongly stable matchings and shows that exchanging ties for posets actually makes the strongly stable marriage problem NP-complete [22]. We show it in this paper that the intermediate case, namely when one side has ties preferences, while the other side has posets, is solvable in polynomial time.

Beyond posets, studies on the stable marriage problem with general preferences occur sporadically. These we include in Table 1 to give a structured overview on them. Intransitive, acyclic preference lists were permitted by Abraham [1], who connects the stable roommates problem with the maximum size weakly stable marriage problem with intransitive, acyclic preference lists in order to derive a structural perspective. Aziz et al. [3] discussed the stable marriage problem under uncertain pairwise preferences. They also considered the case of certain, but cyclic preferences and show that deciding whether a weakly stable matching exists is NP-complete if both sides can have cycles in their preferences. Strongly and super stable matchings were discussed by Farczadi et al. [10]. Throughout their paper they assumed that one side has strict preferences, and show that finding a strongly or a super stable matching (or proving that none exists) can be done in polynomial time if the other side has cyclic lists, where cycles of length at least 3 are permitted to occur, but the problems become NP-complete as soon as cycles of length 2 are also allowed.

1.2 Our contribution

This paper aims to provide a coherent framework for the complexity of the stable marriage problem under various preference structures. We consider the three known notions of stability: weak, strong and super. In our analysis we distinguish six stages of entropy in the preference lists; strict lists, lists with ties, posets, acyclic pairwise preferences, asymmetric pairwise preferences and arbitrary pairwise preferences. All of these have been defined in earlier papers, along with some results on them. Here we collect and organize these known results in all three notions of stability, considering six cases of orderedness for each side of the bipartite graph. Table 1 summarizes these results. Rows and columns distinguish between preference relations considered on the two sides of the graph. The cell itself shows the complexity class of determining whether the specified problem admits a stable matching. All of our positive results also deliver a stable matching or a proof for its nonexistence. For sake of conciseness, NP-completeness is shortened to NP.

Each of the three tables contained empty cells, i.e. cases with unknown complexity so far. These are denoted by color in Table 1. We fill all gaps, providing two NP-completeness proofs and three polynomial time algorithms. Interestingly, the three tables have the border between polynomial time and NP-complete cases at very different places.

Table 1 The complexity tables for weak, strong and super-stability.

WEAK	strict	ties	poset	acyclic	asymmetric or arbitrary
strict	$\mathcal{O}(m)$ [13]	$\mathcal{O}(m)$ [18]	$\mathcal{O}(m)$ [27]	$\mathcal{O}(m)$	**NP**
ties		$\mathcal{O}(m)$ [18]	$\mathcal{O}(m)$ [27]	$\mathcal{O}(m)$	**NP**
poset			$\mathcal{O}(m)$ [27]	$\mathcal{O}(m)$	**NP**
acyclic				$\mathcal{O}(m)$	**NP**
asymmetric or arbitrary					NP [3]

STRONG	strict	ties	poset	acyclic	asymmetric	arbitrary
strict	$\mathcal{O}(m)$ [13]	$\mathcal{O}(nm)$ [18, 24]	pol [10]	pol [10]	pol [10]	NP [10]
ties		$\mathcal{O}(nm)$ [18, 24]	$\mathcal{O}\left(mn^2+m^2\right)$	$\mathcal{O}\left(mn^2+m^2\right)$	$\mathcal{O}\left(mn^2+m^2\right)$	NP [10]
poset			NP [22]	NP [22]	NP [22]	NP [22]
acyclic				NP [22]	NP [22]	NP [22]
asymmetric					NP [22]	NP [22]
arbitrary						NP [22]

SUPER	strict	ties	poset	acyclic	asymm.	arbitrary
strict	$\mathcal{O}(m)$ [13]	$\mathcal{O}(m)$ [18]	$\mathcal{O}(m)$ [18, 28]	$\mathcal{O}(m)$ [10]	$\mathcal{O}(m)$ [10]	NP [10]
ties		$\mathcal{O}(m)$ [18]	$\mathcal{O}(m)$ [18, 28]	$\boldsymbol{\mathcal{O}\left(n^2m\right)}$	$\boldsymbol{\mathcal{O}\left(n^2m\right)}$	NP [10]
poset			$\mathcal{O}(m)$ [18, 28]	$\boldsymbol{\mathcal{O}\left(n^2m\right)}$	$\boldsymbol{\mathcal{O}\left(n^2m\right)}$	NP [10]
acyclic				NP	NP	NP [10]
asymmetric					**NP**	NP [10]
arbitrary						NP [10]

Structure of the paper. We define the problem variants formally in Section 2. Weak, strong and super stable matchings are then discussed in Sections 3, 4 and 5, respectively.

2 Preliminaries

In the stable marriage problem, we are given a not necessarily complete bipartite graph $G = (U \cup W, E)$, where vertices in U represent men, vertices in W represent women, and edges mark the acceptable relationships between them. Each person $v \in U \cup W$ specifies a set $\mathcal{R}_v$ of pairwise comparisons on the vertices adjacent to them. These comparisons as ordered pairs define four possible relations between two vertices a and b in the neighborhood of v.

- a is preferred to b, while b is not preferred to a by v: $a \prec_v b$;
- a is not preferred to b, while b is preferred to a by v: $a \succ_v b$;
- a is not preferred to b, neither is b preferred to a by v: $a \sim_v b$;
- a is preferred to b, so is b preferred to a by v: $a||_v b$.

In words, the first two relationships express that an agent v *prefers* one agent *strictly* to the other. The third option is interpreted as *incomparability*, or a not yet known relation between the two agents. The last relation tells that v knows for sure that the two options are *equally good*. For example, if v is a sports sponsor considering to offer a contract to exactly one of players a and b, then v's preferences are described by these four relations in the following scenarios: a beats b, b beats a, a and b have not played against each other yet, and finally, a and b played a draw.

We say that edge va *dominates* edge vb if $a \prec_v b$. If $a \prec_v b$ or $a \sim_v b$, then b is *not preferred to* a. Sticking to our previous example with players a and b, this relation delivers

the information that either a has beaten b or they have not played yet. With this amount of somewhat uncertain information, the sports sponsor has no reason to choose b, and choosing a is also risky, because it might be the case that the two players have not played against each other yet. For two out of the three notions of stability, we will define blocking based on this risk. Another choice would be to replace $a \sim_v b$ by $a||_v b$ in the definition above. While it would lead to an equally correct model, we chose incomparability consciously. Some early papers [18, 19] do not distinguish between two agents being incomparable and equally good, while some others in the more recent literature [3, 10] motivate strong and super-stability with uncertain information. Our definition fits the more recent framework.

The partner of vertex v in matching M is denoted by $M(v)$. The neighborhood of v in graph G is denoted by $\mathcal{N}_G(v)$ and it consists of all vertices that are adjacent to v in G. To ease notation, we introduce the empty set as a possible partner to each vertex, symbolizing the vertex remaining unmatched in a matching M ($M(v) = \emptyset$). As usual, being matched to any acceptable vertex is preferred to not being matched at all: $a \prec_v \emptyset$ for every $a \in \mathcal{N}(v)$. Edges to unacceptable partners do not exist, thus these are not in any pairwise relation to each other or to edges incident to v.

We differentiate six degrees of preference orderedness in our study.

1. The strictest, classical two-sided model [13] requires each vertex to rank all of its neighbors in a *strict* order of preference. For each vertex, this translates to a transitive and complete set of pairwise relations on all adjacent vertices.
2. This model has been relaxed very early to lists admitting *ties* [18]. The pairwise preferences of vertex v form a preference list with ties if the neighbors of v can be clustered into some sets $N_1, N_2, \ldots, N_k$ so that vertices in the same set are incomparable, while for any two vertices in different sets, the vertex in the set with the lower index is strictly preferred to the other one.
3. Following the traditions [12, 19, 22, 27], the third degree of orderedness we define is when preferences are expressed as *posets*. Any set of antisymmetric and transitive pairwise preferences by definition forms a partially ordered set.
4. By dropping transitivity but still keeping the structure cycle-free, we arrive to *acyclic* preferences [1]. This category allows for example $a \sim_v c$, if $a \prec_v b \prec_v c$, but it excludes $a||_v c$ and $a \succ_v c$.
5. *Asymmetric* preferences [10] may contain cycles of length at least 3. This is equivalent to dropping acyclicity from the previous cluster, but still prohibiting the indifference relation $a||_v b$, which is essentially a 2-cycle in the form a is preferred to b, and b is preferred to a.
6. Finally, an *arbitrary* set of pairwise preferences can also be allowed [3, 10].

A matching is *stable* if it admits no blocking edge. For strict preferences, a blocking edge was defined in the seminal paper of Gale and Shapley [13]: an edge $uv \notin M$ blocks matching M if both u and v prefer each other to their partner in M or they are unmatched. Already when extending this notion to preference lists with ties, one needs to specify how to deal with incomparability. Irving [18] defined three notions of stability. We extend them to pairwise preferences in the coming three sections. We omit the adjectives weakly, strongly, and super wherever there is no ambiguity about the type of stability in question. All missing proofs can be found in the full version of the paper [9].

3 Weak stability

In weak stability, an edge outside of M blocks M if it is *strictly preferred* to the matching edge by *both* of its end vertices. From this definition follows that $w||_u w'$ and $w \sim_u w'$

are exchangeable in weak stability, because blocking occurs only if the non-matching edge dominates the matching edges at both end vertices. Therefore, an instance with arbitrary pairwise preferences can be assumed to be asymmetric.

▶ **Definition 1** (blocking edge for weak stability). *Edge uw blocks M, if*

1. $uw \notin M$;
2. $w \prec_u M(u)$;
3. $u \prec_w M(w)$.

For weak stability, preference structures up to posets have been investigated, see Table 1. A stable solution is guaranteed to exist in these cases [18, 27]. Here we extend this result to acyclic lists, and complement it with a hardness proof for all cases where asymmetric lists appear, even if they do so on one side only.

▶ **Theorem 2.** *Any instance of the stable marriage problem with acyclic pairwise preferences for all vertices admits a weakly stable matching, and there is a polynomial time algorithm to determine such a matching.*

Proof. We utilize a widely used argument [18] to show this. For acyclic relations $\mathcal{R}_v$, a linear extension $\mathcal{R}'_v$ of $\mathcal{R}_v$ exists. The extended instance with linear preferences is guaranteed to admit a stable matching [13]. Compared to $\mathcal{R}_v$, relations in $\mathcal{R}'_v$ impose more constraints on stability, therefore, they can only restrict the original set of weakly stable solutions. If both sides have acyclic lists, a stable matching is thus guaranteed to exist and a single run of the Gale-Shapley algorithm on the extended instance delivers one. ◀

Stable matchings are not guaranteed to exist as soon as a cycle appears in the preferences, as Example 3 demonstrates. Theorem 4 shows that the decision problem is in fact hard from that point on.

▶ **Example 3.** No stable matching can be found in the following instance with strict lists on one side and asymmetric lists on the other side. There are three men u_1, u_2, u_3 adjacent to one woman w. The woman's pairwise preferences are cyclic: $u_1 \prec u_2, u_2 \prec u_3, u_3 \prec u_1$. Any stable matching M must consist of a single edge. Since the men's preferences are identical, we can assume that $u_1 w \in M$ without loss of generality. Then $u_3 w$ blocks M.

▶ **Theorem 4.** *If one side has strict lists, while the other side has asymmetric pairwise preferences, then determining whether a weakly stable matching exists is* NP*-complete, even if each agent finds at most four other agents acceptable.*

4 Strong stability

In strong stability, an edge outside of M blocks M if it is *strictly preferred* to the matching edge by *one* of its end vertices, while the other end vertex *does not prefer* its matching edge to it.

▶ **Definition 5** (blocking edge for strong stability). *Edge uw blocks M, if*

1. $uw \notin M$;
2. $w \prec_u M(u)$ *or* $w \sim_u M(u)$;
3. $u \prec_w M(w)$,

or

1. $uw \notin M$;
2. $w \prec_u M(u)$;
3. $u \prec_w M(w)$ *or* $u \sim_w M(w)$.

The largest set of relevant publications has appeared on strong stability, yet gaps were present in the complexity table, see Table 1. In this section we present a polynomial algorithm that is valid in all cases not solved yet. We assume men to have preference lists with ties, and women to have asymmetric relations. Our algorithm returns a strongly stable matching or a proof for its nonexistence. It can be seen as an extended version of Irving's algorithm for strongly stable matchings in instances with ties on both sides [18]. Our contribution is a sophisticated rejection routine, which is necessary here, because of the intransitivity of preferences on the women's side. The algorithm in [10] solves the problem for strict lists on the men's side, and it is much simpler than ours. It was designed for super stable matchings, but strong and super stability do not differ if one side has strict lists. For this reason, that algorithm is not suitable for an extension in strong stability.

Roughly speaking, our algorithm alternates between two phases, both of which iteratively eliminate edges that cannot occur in a strongly stable matching. In the first phase, Gale-Shapley proposals and rejections happen, while the second phase focuses on finding a vertex set violating the Hall condition in a specified subgraph. Finally, if no edge can be eliminated any more, then we show that an arbitrary maximum matching is either stable or it is a proof for the non-existence of stable matchings. Algorithms 1 and 2 below provide a pseudocode. The time complexity analysis has been shifted to the full version of the paper [9].

The second phase of the algorithm relies on the notion of the *critical set* in a bipartite graph, also utilized in [18], which we sketch here. For an exhaustive description we refer the reader to [26]. The well-known Hall-condition [15] states that there is a matching covering the entire vertex set U if and only if for each $X \subseteq U$, $|\mathcal{N}(X)| \geq |X|$. Informally speaking, the reason for no matching being able to cover all the vertices in U is that a subset X of them has too few neighbors in W to cover their needs. The difference $\delta(X) = |X| - |\mathcal{N}(X)|$ is called the *deficiency of* X. It is straightforward that for any $X \subseteq U$, at least $\delta(X)$ vertices in X cannot be covered by any matching in G, if $\delta(X) > 0$. Let $\delta(G)$ denote the maximum deficiency over all subsets of U. Since $\delta(\emptyset) = 0$, we know that $\delta(G) \geq 0$. Moreover, it can be shown the size of maximum matching is $\nu(G) = |U| - \delta(G)$. If we let Z_1, Z_2 be two arbitrary subsets of U realizing the maximum deficiency, then $Z_1 \cap Z_2$ has maximum deficiency as well. Therefore, the intersection of all maximum-deficiency subsets of U is the unique set with maximum deficiency with the following properties: it has the lowest number of elements and it is contained in all other subsets with maximum deficiency. This set is called the *critical set* of G. Last but not least, it is computationally easy to determine the critical set, since for any maximum matching M in G, the critical set consists of vertices in U not covered by M and vertices in U reachable from the uncovered ones via an alternating path.

▶ **Theorem 6.** *If one side has tied preferences, while the other side has asymmetric pairwise preferences, then deciding whether the instance admits a strongly stable matching can be done in* $\mathcal{O}(mn^2 + m^2)$ *time.*

Initialization. For the clarity of our proofs, we add a dummy partner w_u to the bottom of the list of each man u, where w_u is not acceptable to any other man (line 1). We call the modified instance $\mathcal{I}'$. This standard technical modification is to ensure that all men are matched in all stable matchings. At start, all edges are *inactive* (line 2). The possible states of an edge and the transitions between them are illustrated in Figure 1.

First phase. The first phase of our algorithm (lines 3-9) imitates the classical Gale-Shapley deferred acceptance procedure. In the first round, each unmatched man simultaneously

Algorithm 1 Strongly stable matching with ties and asymmetric relations.

Input: $\mathcal{I} = (U, W, E, \mathcal{R}_U, \mathcal{R}_W)$; $\mathcal{R}_U$: lists with ties, $\mathcal{R}_W$: asymmetric.

INITIALIZATION

for each $u \in U$ add an extra woman w_u at the end of his list; w_u is only acceptable for u
set all edges to be inactive

PHASE 1

while there exists a man with no active edge **do**
    propose along all edges of each such man u in the next tie on his list
    for each new proposal edge uw **do**
        reject all edges $u'w$ such that $u \prec_w u'$
    end for
    STRONG_REJECT()
end while

PHASE 2

let G_A be the graph of active edges with $V(G_A) = U \cup W$
let $U' \subseteq U$ be the critical set of men with respect to G_A
if $U' \neq \emptyset$ **then**
    all active edges of each $u \in U'$ are rejected
    STRONG_REJECT()
    goto PHASE 1
end if

OUTPUT

let M be a maximum matching in G_A
if M covers all women who have ever had an active edge **then**
    STOP, OUTPUT $M \cap E$ and "There is a strongly stable matching."
else
    STOP, OUTPUT "There is no strongly stable matching."
end if

Algorithm 2 STRONG_REJECT().

let $R = U$
while $R \neq \emptyset$ **do**
    let u be an element of R
    if u has exactly one active edge uw **then**
        reject all $u'w$ such that $u' \sim_w u$
        if $u'w$ was active, then let $R := R \cup \{u'\}$
    else if u has no active edge **then**
        reject all $u'w$ such that w is in the proposal tie of u and $u' \sim_w u$
        if $u'w$ was active, then let $R := R \cup \{u'\}$
    end if
    let $R := R \setminus \{u\}$
end while

proposes to all women in his top tie (line 4). The so far inactive edges that now carry a proposal are called *active proposal edges*, or just *active edges*. Active edges stay active if they are accepted by the woman, and they become *rejected proposal edges* as soon as they are rejected by the woman they run to. The tie that a man has just proposed along is called the man's *proposal tie*. If all edges in the proposal tie are rejected (or more precisely, they become rejected proposal edges), then the man steps down on his list and proposes along all edges in the next tie (lines 3-4).

Proposals cause two types of rejections in the graph (lines 5-8), based on the rules below. Notice that these rules are more sophisticated than in the Gale-Shapley or Irving algorithms [13, 18]. The most striking difference may be that rejected edges are not deleted from the graph, since they can very well carry a proposal later. To be fully accurate, inactive edges that are rejected become *rejected inactive edges* (see Figure 1). Upon carrying a proposal later, they convert to a *rejected proposal edge*. This latter is the same state an edge ends up in if it is first proposed along and then rejected.

Edges that carry a proposal in this round, but have not carried a proposal in earlier rounds, i.e. edges in the proposal tie of men, are called *new proposal edges* (for instance, see line 5). Once again notice that these edges might or might not be active, depending on whether they have been rejected earlier.

- For each new proposal edge uw, w rejects all edges to which uw is strictly preferred (lines 5-7). Note again that uw might have been rejected earlier than being proposed along, in which case uw is a proposal edge without being active.
- The second kind of rejections are detailed in Algorithm 2. We search for a man in the set R of men to be investigated, whose set of active edges has cardinality at most 1 (lines 23-25). If any such man has exactly one active edge uw (line 26), then all other edges that are incident to w and incomparable to uw are rejected (line 27). If man u' has lost an active edge in the previous operation, then u' is added back to the set R of men to be investigated in later rounds (line 28). The other case is when a man u has no active edge at all (line 29). In this case, all edges that are incident to any neighbor w of u in his – now fully rejected – proposal tie and incomparable to uw at w are rejected (line 30). The set R is again supplemented by those men who lost active edges during the previous operation (line 31). Finally, the man u chosen at the beginning of this rejection round is excluded from R.

As mentioned earlier, men without any active edge proceed to propose along the next tie in their list. These operations are executed until there is no more edge to propose along or to reject, which marks the end of the first phase.

Second phase. In the second phase, the set of active edges induce the graph G_A, on which we examine the critical set U' (lines 10-11). If U' is not empty, then all active edges of each $u \in U'$ are rejected (line 13). These rejections might trigger more rejections, which are handled by calling Algorithm 2 as a subroutine (line 14). The mass rejections in line 13 generate a new proposal tie for at least one man, returning to the first phase (line 15). Note that an empty critical set leads to producing the output, which is described just below.

Output. In the final set of active edges, an arbitrary maximum matching M is calculated (line 17). If M covers all women who have ever had an active edge, then we send it to the output (lines 18-19), otherwise we report that no stable matching exists (lines 20-21).

We prove Theorem 6 via a number of claims, building up the proof as follows. The first three claims provide the technical footing for the last two claims. Claim 7 is a rather

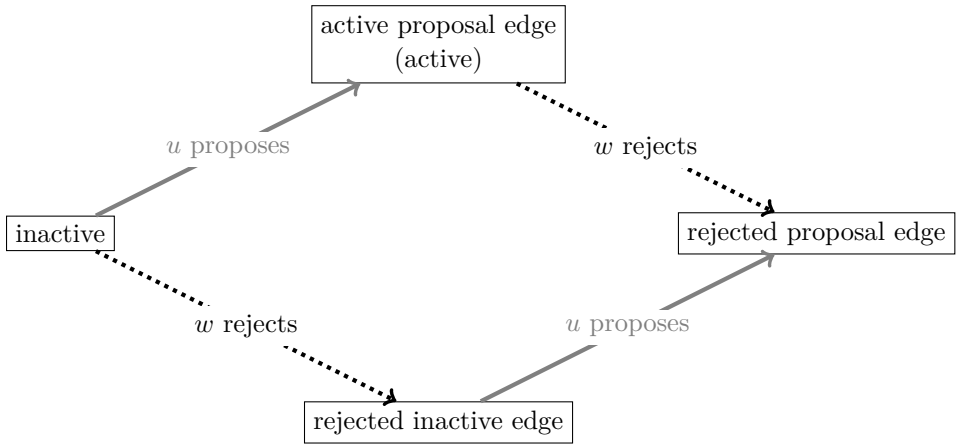


Figure 1 The possible states of an edge uw in Algorithm 1. The solid gray edges between the states symbolize proposals, while the dotted black edges mark the rejections of vertex w.

technical observation about the righteousness of the input initialization. An edge appearing in any stable matching is called a *stable edge*. Claim 8 shows that no stable edge is ever rejected. Claim 9 proves that all stable matchings must cover all women who have ever received an offer. Then, Claim 10 proves that if the algorithm outputs a matching, then it must be stable, and Claim 11 along with Corollary 12 conclude the opposite direction: if stable matchings exist, then one is outputted by our algorithm.

▷ Claim 7. A matching in $\mathcal{I}'$ is stable if and only if its restriction to $\mathcal{I}$ is stable and it covers all men in $\mathcal{I}'$.

Proof. If a matching in $\mathcal{I}'$ leaves a man u unmatched, then uw_u blocks the matching. Thus all stable matchings in $\mathcal{I}'$ cover all men. Furthermore, the restriction to $\mathcal{I}$ of a stable matching in $\mathcal{I}'$ cannot be blocked by any edge in $\mathcal{I}$, because this blocking edge also exists in $\mathcal{I}'$.

A stable matching in $\mathcal{I}$, supplemented by the dummy edges for all unmatched men cannot be blocked by any edge in $\mathcal{I}'$, because dummy edges are last-choice edges and regular edges block in both instances simultaneously. ◁

▷ Claim 8. No stable edge is ever rejected in the algorithm.

Proof. Let us suppose that uw is the first rejected stable edge and the corresponding stable matching is M. There are four rejection calls, in lines 6, 13, 27, and 30. In all cases we derive a contradiction. Our arguments are illustrated in Figure 2.

- Line 6: uw was rejected because w received a proposal from a man u' such that $u' \prec_w u$. Since M is stable, u' must have a partner w' in M such that $w' \prec_{u'} w$. We also know that u' has reached w with its proposal ties, thus, due to the monotonicity of proposals, $u'w' \in M$ must have been rejected before uw was rejected. This contradicts our assumption that uw was the first rejected stable edge.
- Lines 27 and 30: rejection was caused by a man u' such that $u' \sim_w u$.
  Either the whole proposal tie of u' was rejected or $u'w$ was the only active edge within this tie. Since M is stable, u' must have a partner w' in M. Since $u'w'$ is a stable edge, it cannot have been rejected previously. Consequently, $w \prec_{u'} w'$. Thus, $u'w$ blocks M, which contradicts its stability.

- Line 13: uw was rejected as an active edge incident to the critical set U' in G_A.
  Let $W' = \mathcal{N}_{G_A}(U')$, $U'' = \{u \in U' : M(u) \in W'\}$, and $W'' = \{w \in W' : M(w) \in U'\}$. In words, W' is the neighborhood of U', while U'' and W'' represent the men and women in U' and W' who are paired up in M. Due to our assumption, $u \in U''$ and $w \in W''$.
  We claim that $|U' \setminus U''| < |U'|$ and $\delta(U' \setminus U'') \geq \delta(U')$, which contradicts the fact that U' is critical. Since $U'' \neq \emptyset$, the first part holds. Note that $|U''| = |W''|$, so it suffices to show that $\mathcal{N}_{G_A}(U' \setminus U'') \subseteq W' \setminus W''$, because in that case

$$\begin{aligned}\delta(U' \setminus U'') = |U' \setminus U''| - |\mathcal{N}_{G_A}(U' \setminus U'')| \geq |U' \setminus U''| - |W' \setminus W''| = \\ = (|U'| - |W'|) - (|U''| - |W''|) = \\ = |U'| - |W'| = \delta(U'),\end{aligned}$$

  which would prove the second part of our claim.
  What remains to show is that $\mathcal{N}_{G_A}(U' \setminus U'') \subseteq W' \setminus W''$. Suppose the contrary, i.e. that there exists an edge ab in G_A from $U' \setminus U''$ to W''. See the third graph in Figure 2. We know that $b \in W''$ by our indirect assumption, hence $a' = M(b) \in U''$ by the definition of U'', and $a' \neq a$, because $a \notin U''$. Moreover, ab and $a'b$ are edges in G_A, thus both of them are active. Therefore, $a \sim_b a'$, for otherwise b would have rejected one of them. In order to keep M stable, a must be paired up in M with some woman b'. Since no stable edge has been rejected so far and ab does not block M, we know that $b' \sim_a b$, thus b' is in a's proposal tie. Edge ab' is stable and no stable edge has been rejected yet, thus ab' is active along with ab. Therefore, $ab' \in E(G_A)$ and $b' \in W'$. Moreover, $ab' \in M$, hence $a \in U''$ and $b' \in W''$ by the definition of U'' and W'', which contradicts the assumption that $a \notin U''$. ◁

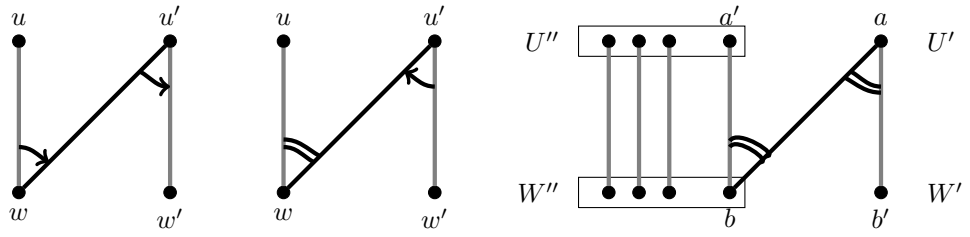


Figure 2 The three cases in Claim 8. Gray edges are in M. The arrows point to the strictly preferred edges.

▷ Claim 9. Women who have ever had an active edge must be matched in all stable matchings.

Proof. Claim 8 shows that stable matchings allocate each man u a partner not better than his final proposal tie. If a man u proposed to woman w and yet w is unmatched in the stable matching M, then uw blocks M, which contradicts the stability of M. ◁

▷ Claim 10. If our algorithm outputs a matching, then it is stable.

Proof. We need to show that any maximum matching M in G_A is stable, if it covers all women who have ever held a proposal. Let M be such a matching. Due to the exit criteria of the second phase (lines 11 and 12), M covers all men. By contradiction, let us assume that M is blocked by an edge uw. This can occur in three cases.

- While w is unmatched, u does not prefer $M(u)$ to w.
  Since uw carried a proposal at the same time or before $uM(u) \in E(G_A)$ was activated, w is a woman who has held an offer during the course of the algorithm. We assumed that all these women are matched in M.
- While $w \prec_u M(u)$, w does not prefer $M(w)$ to u.
  The full tie at u containing uw must have been rejected in the algorithm, otherwise $uM(u)$ would not be an active edge. We know that either $u \prec_w M(w)$ or $u \sim_w M(w)$ holds. If $u \prec_w M(w)$, then $wM(w)$ had to be rejected when u proposed to w, which contradicts our assumption that $wM(w) \in E(G_A)$. Hence, $u \sim_w M(w)$. Thus, when uw and its full tie was rejected at u, $M(w)w$ also should have been rejected in a STRONG_REJECT procedure, which leads to the same contradiction with $wM(w) \in E(G_A)$.
- While $u \prec_w M(w)$, u does not prefer $M(u)$ to w.
  Since $uM(u)$ is an active edge, uw has carried a proposal, because $M(u)$ is not preferred to w by u. When uw was proposed along, w should have rejected $M(w)w$, to which uw is strictly preferred. This contradicts our assumption that $wM(w) \in E(G_A)$. ◁

▷ Claim 11. If $\mathcal{I}'$ admits a stable matching M', then any maximum matching M in the final G_A covers all women who have ever held a proposal.

Proof. From Claims 7 and 9 we know that M' covers all women who have ever held a proposal and all men. It is also obvious that matching M found in line 17 covers all men, for otherwise U' could not have been the empty set in line 12 and the execution would have returned to the first phase. This means that $|M| = |M'|$. On the other hand, all women covered by $M \subseteq E(G_A)$ are fit with active edges in G_A. Therefore, women covered by M represent only a subset of women who have ever had an active edge, i.e. the women covered by M'. In order to M and M' have the same cardinality, they must cover exactly the same women. Thus, M covers all women who have ever received a proposal. ◁

▶ **Corollary 12.** *If $\mathcal{I}$ admits a stable matching then our algorithm outputs one.*

Proof. Since the edges between men and their dummy partners cannot be rejected, the algorithm will proceed to line 17. Courtesy of Claim 11, the output M covers all women who have ever received a proposal. According to Claim 10, this matching is stable, and thus we output a stable matching of $\mathcal{I}$. ◀

5 Super-stability

In super-stability, an edge outside of M blocks M if *neither* of its end vertices *prefer* their matching edge to it.

▶ **Definition 13** (blocking edge for super-stability). *Edge uw blocks M, if*
1. $uw \notin M$;
2. $w \prec_u M(u)$ *or* $w \sim_u M(u)$;
3. $u \prec_w M(w)$ *or* $u \sim_w M(w)$.

The set of already investigated problems is remarkable for super-stability, see Table 1. Up to posets on both sides, a polynomial algorithm is known to decide whether a stable solution exists [18, 28]. Even though it is not explicitly written there, a blocking edge in the super stable sense is identical to the definition of a blocking edge given in [10]. It is shown there that if one vertex class has strictly ordered preference lists and the other vertex class

has arbitrary relations, then determining whether a stable solution exists is NP-complete, but if the second class has asymmetric lists, then the problem becomes tractable.

We first show that a polynomial algorithm exists up to partially ordered relations on one side and asymmetric relations on the other side. Our algorithm can be seen as an extension of the one in [10]. Our added contributions are a more sophisticated proposal routine and the condition on stability in the output. These are necessary as men are allowed to have acyclic preferences instead of strictly ordered lists, as in [10]. Finally, we prove that acyclic relations on both sides make the problem hard.

▶ **Theorem 14.** *If one side has posets as preferences, while the other side has asymmetric pairwise preferences, then deciding whether the instance admits a super stable matching can be done in* $\mathcal{O}\left(n^2 m\right)$ *time.*

We prove this theorem by designing an algorithm that produces a stable matching or a proof for its nonexistence, see Algorithm 3. We assume men to have posets as preferences and women to have asymmetric relations. We remark that non-empty posets always have a non-empty set of *maximal elements*: these are the ones that are not dominated by any other element. Women in the set of maximal elements are called *maximal* women.

At start, an arbitrary man proposes to one of his maximal women. An offer from u is temporarily accepted by w if and only if $u \prec_w u'$ for every man $u' \neq u$ who has ever proposed to w. This rule forces each woman to reproof her current match every time a new proposal arrives. Accepted offers are called *engagements*. The proposal edges or engagements not meeting the above requirement are immediately deleted from the graph. Each man then reexamines the poset of women still on his list. If any of the maximal women is not holding an offer from him, then he proposes to her. The process terminates and the output is generated when no man has maximal women he has not proposed to. Notice that while women hold at most one proposal at a time, men might have several engagements in the output.

Algorithm 3 Super stable matching with posets and asymmetric relations.

Input: $\mathcal{I} = (U, W, E, \mathcal{R}_U, \mathcal{R}_W)$; $\mathcal{R}_U$: posets, $\mathcal{R}_W$: asymmetric.

while there is a man u who has not proposed to a maximal woman w **do**
    u proposes to w
    if $u \prec_w u'$ for all $u' \in U$ who has ever proposed to w **then**
        w accepts the proposal of u, uw becomes an engagement
    else
        w rejects the proposal and deletes uw
    end if
    if w had a previous engagement to u' and $u \prec_w u'$ or $u \sim_w u'$ **then**
        w breaks the engagement to u' and deletes $u'w$
    end if
end while

let M be the set of engagements
if M is a matching that covers all women who have ever received a proposal **then**
    STOP, OUTPUT M and "M is a super stable matching."
else
    STOP, OUTPUT "There is no super stable matching."
end if

The correctness and time complexity of our algorithm is shown in the full version of the paper [9], where we prove that the set of engagements M is a matching that covers all women who ever received a proposal if and only if the instance admits a stable matching.

▶ **Theorem 15.** *If both sides have acyclic pairwise preferences, then determining whether a super stable matching exists is* NP*-complete, even if each agent finds at most four other agents acceptable.*

6 Conclusion and open questions

We completed the complexity study of the stable marriage problem with pairwise preferences. Despite of the integrity of this work, our approach opens the way to new research problems.

The six degrees of orderedness can be interpreted in the non-bipartite stable roommates problem as well. For strictly ordered preferences, all three notions of stability reduce to the classical stable roommates problem, which can be solved in $\mathcal{O}(m)$ time [17]. The weakly stable variant becomes NP-complete already if ties are present [31], while the strongly stable version can be solved with ties in polynomial time, but it is NP-complete for posets. The complexity analysis of these cases is thus complete. Not so for super-stability, for which there is an $\mathcal{O}(m)$ time algorithm for preferences ordered as posets [19], while the case with asymmetric preferences was shown here to be NP-complete for bipartite instances as well. We conjecture that the intermediate case of acyclic preferences is also polynomially solvable and the algorithm of Irving and Manlove can be extended to it.

The Rural Hospitals Theorem [14] states that the set of matched vertices is identical in all stable matchings. It has been shown to hold for strongly and super stable matchings [20, 27] and fail for weak stability, if preferences contain ties – even for non-bipartite instances. We remark that these results carry over even to the most general pairwise preference setting. To see this, one only needs to sketch the usual alternating path argument: assume that there is a vertex v that is covered by a stable matching M_1, but left uncovered by another stable matching M_2. Then, $M_1(v)$ must strictly prefer its partner in M_2 to v, otherwise edge $vM_1(v)$ blocks M_2. Iterating this argument, we derive that such a v cannot exist. The Rural Hospitals Theorem might indicate a rich underlying structure of the set of stable matchings. Such results were shown in the case of preferences with ties. Strongly stable matchings are known to form a distributive lattice [27], and there is a partial order with $\mathcal{O}(m)$ elements representing all strongly stable matchings [25]. However, once posets are allowed in the preferences, the lattice structure falls apart [27]. The set of super stable matchings has been shown to form a distributive lattice if preferences are expressed in the form of posets [27, 32]. The questions arise naturally: does this distributive lattice structure carry over to more advanced preference structures in the super stable case? Also, even if no distributive lattice exists on the set of strongly stable matchings, is there any other structure and if so, how far does it extend in terms of orderedness of preferences?

References

1 D. J. Abraham. Algorithmics of two-sided matching problems. Master's thesis, University of Glasgow, Department of Computing Science, 2003.

2 Alberto Alesina and Howard Rosenthal. A Theory of Divided Government. *Econometrica*, 64(6):1311–1341, 1996. URL: `http://www.jstor.org/stable/2171833`.

3 Haris Aziz, Péter Biró, Tamás Fleiner, Serge Gaspers, Ronald de Haan, Nicholas Mattei, and Baharak Rastegari. Stable Matching with Uncertain Pairwise Preferences. In *Proceedings of the*

16th Conference on Autonomous Agents and MultiAgent Systems, pages 344–352. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

4 Péter Biró. Applications of Matching Models under Preferences. *Trends in Computational Social Choice*, page 345, 2017.

5 Péter Biró and Sofya Kiselgof. College admissions with stable score-limits. *Central European Journal of Operations Research*, 23(4):727–741, 2015.

6 P Blavatsky. Content-dependent preferences in choice under risk: heuristic of relative probability comparisons. *IIASA Interim Report*, IR-03-031, 2003.

7 Li Chen. University admission practices – Ireland, MiP Country Profile 8. `http://www.matching-in-practice.eu/higher-education-in-ireland/`, 2012.

8 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'Imprimerie Royale, 1785.

9 Ágnes Cseh and Attila Juhos. Pairwise preferences in the stable marriage problem. *CoRR*, 2018. `arXiv:1810.00392`.

10 Linda Farczadi, Konstantinos Georgiou, and Jochen Könemann. Stable marriage with general preferences. *Theory of Computing Systems*, 59(4):683–699, 2016.

11 P. Fishburn. Preference structures and their numerical representations. *Theoretical Computer Science*, 217:359–383, 1999.

12 T. Fleiner, R. W. Irving, and D. F. Manlove. Efficient Algorithms for Generalised Stable Marriage and Roommates Problems. *Theoretical Computer Science*, 381:162–176, 2007.

13 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

14 D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

15 P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935.

16 Steven Humphrey. Non-transitive Choice: Event-Splitting Effects or Framing Effects? *Economica*, 68(269):77–96, 2001.

17 R. W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6:577–595, 1985.

18 R. W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.

19 R. W. Irving and D. F. Manlove. The stable roommates problem with ties. *Journal of Algorithms*, 43:85–105, 2002.

20 R. W. Irving, D. F. Manlove, and S. Scott. The Hospitals / Residents problem with ties. In Magnús M. Halldórsson, editor, *Proceedings of SWAT '00: the 7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2000.

21 R. W. Irving, D. F. Manlove, and S. Scott. Strong stability in the Hospitals / Residents problem. Technical Report TR-2002-123, University of Glasgow, Department of Computing Science, 2002. Revised May 2005.

22 R. W. Irving, D. F. Manlove, and S. Scott. Strong stability in the Hospitals / Residents problem. In *Proceedings of STACS '03: the 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 2003. Full version available as [21].

23 Tobias Kalenscher, Philippe N Tobler, Willem Huijbers, Sander M Daselaar, and Cyriel MA Pennartz. Neural signatures of intransitive preferences. *Frontiers in Human Neuroscience*, 4:49, 2010.

24 Telikepalli Kavitha, Kurt Mehlhorn, Dimitrios Michail, and Katarzyna E Paluch. Strongly stable matchings in time $O(nm)$ and extension to the hospitals-residents problem. *ACM Transactions on Algorithms*, 3(2):15, 2007.

25 Adam Kunysz, Katarzyna Paluch, and Pratik Ghosal. Characterisation of strongly stable matchings. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 107–119. Society for Industrial and Applied Mathematics, 2016.

26 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

27 D. F. Manlove. The structure of stable marriage with indifference. *Discrete Applied Mathematics*, 122(1-3):167–181, 2002.

28 D. F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.

29 Kenneth O. May. Intransitivity, Utility, and the Aggregation of Preference Patterns. *Econometrica*, 22(1):1–13, 1954.

30 Ignacio Ríos, Tomás Larroucau, Giorgiogiulio Parra, and Roberto Cominetti. College Admissions Problem with Ties and Flexible Quotas. Technical report, SSRN, 2014.

31 E. Ronn. *On the complexity of stable matchings with and without ties*. PhD thesis, Yale University, 1986.

32 B. Spieker. The set of super-stable marriages forms a distributive lattice. *Discrete Applied Mathematics*, 58:79–84, 1995.

33 Huffington Post 2016 General Election: Trump vs. Sanders. `https://elections.huffingtonpost.com/pollster/2016-general-election-trump-vs-sanders`. Accessed 3 September 2018.

34 RealClearPolitics website. General Election: Trump vs. Sanders. `https://www.realclearpolitics.com/epolls/2016/president/us/general_election_trump_vs_sanders-5565.html`. Accessed 3 September 2018.

Closure Properties of Synchronized Relations

María Emilia Descotte
LaBRI, Université de Bordeaux, France

Diego Figueira
CNRS & LaBRI, Université de Bordeaux, France

Santiago Figueira
CONICET & Universidad de Buenos Aires, Argentina

Abstract

A standard approach to define k-ary word relations over a finite alphabet $\mathbb{A}$ is through k-tape finite state automata that recognize regular languages L over $\{1,\dots,k\} \times \mathbb{A}$, where (i,a) is interpreted as reading letter a from tape i. Accordingly, a word $w \in L$ denotes the tuple $(u_1,\dots,u_k) \in (\mathbb{A}^*)^k$ in which u_i is the projection of w onto i-labelled letters. While this formalism defines the well-studied class of rational relations, enforcing restrictions on the reading regime from the tapes, which we call *synchronization*, yields various sub-classes of relations. Such synchronization restrictions are imposed through regular properties on the projection of the language L onto $\{1,\dots,k\}$. In this way, for each regular language $C \subseteq \{1,\dots,k\}^*$, one obtains a class $\textsc{Rel}(C)$ of relations. Synchronous, Recognizable, and Length-preserving rational relations are all examples of classes that can be defined in this way.

We study basic properties of these classes of relations, in terms of closure under intersection, complement, concatenation, Kleene star and projection. We characterize the classes with each closure property. For the binary case ($k = 2$) this yields effective procedures.

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

Keywords and phrases synchronized word relations, rational, closure, characterization, intersection, complement, Kleene star, concatenation

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.22

Acknowledgements Work supported by ANR project DELTA, grant ANR-16-CE40-0007, grant PICT-2016-0215, and LIA INFINIS.

1 Introduction

We study relations of finite words, that is, sets $R \subseteq (\mathbb{A}^*)^k$ for a finite alphabet $\mathbb{A}$ and $k \in \mathbb{N}$, where $(\mathbb{A}^*)^k$ is the cartesian product of k copies of $\mathbb{A}^*$. The study of these relations dates back to the works of Büchi, Elgot, Mezei, and Nivat in the 1960s [11, 15, 24], with much subsequent work done later (*e.g.*, [7, 13]). Most of the investigations focused on extending the standard notion of regularity from languages to relations. This effort has followed the long-standing tradition of using equational, operational, and descriptive formalisms – that is, finite monoids, automata, and regular expressions – for describing relations, and gave rise to three different classes of relations: *Recognizable*, *Automatic* (*a.k.a. Regular* [7] or *Synchronous* [20, 13]), and *Rational*.

The above classes of relations can be seen as three particular examples of a much larger (in fact infinite) range of possibilities, where relations are described by special languages over extended alphabets, called *synchronizing languages* [18]. Intuitively, the idea is to describe a k-ary relation by means of a k-tape automaton with k heads, one for each tape, which can move independently of one another. In the basic framework of synchronized relations, one lets each head of the automaton either move right or stay in the same position. In addition, one can constrain the possible sequences of head motions by a suitable

regular language $C \subseteq \{1, \ldots, k\}^*$. In this way, each regular language $C \subseteq \{1, \ldots, k\}^*$ induces a class of k-ary relations, denoted $\textsc{Rel}(C)$, which is contained in the class *Rational* (due to Nivat's Theorem [24]). For example, on binary relations, the classes *Recognizable*, *Automatic*, and *Rational* are captured, respectively, by the languages $C_{\text{Rec}} = \{1\}^* \cdot \{2\}^*$, $C_{\text{Aut}} = \{12\}^* \cdot \{1\}^* \cup \{12\}^* \cdot \{2\}^*$, and $C_{\text{Rat}} = \{1, 2\}^*$. Roughly speaking, any other class that can be defined through the 'tape behavior' of a multi-tape automaton will be also captured by this framework. Other examples include length-preserving, or α-synchronous relations [12]. However, it should be noted that other well-known subclasses of rational relations, such as deterministic or functional relations, are not captured by the notion of synchronization. In general, the correspondence between a language $C \subseteq \{1, \ldots, k\}^*$ and the induced class $\textsc{Rel}(C)$ of synchronized relations is not one-to-one: it may happen that different languages C, D induce the same class of synchronized relations. The problem of when two classes of synchronized relations coincide, and when one is contained in the other has been only recently solved for the case of binary relations [14], while the case for arbitrary k-ary relations remains open. In this work we identify, among the infinitely many synchronized classes of relations, which are those with good closure properties, in terms of paradigmatic operations such as intersection, complement, concatenation, projection, or Kleene star.

Motivation

The motivation for identifying and studying well-behaved classes of word relations, besides its intrinsic interest within formal language theory, stems from various areas. One motivation comes from verification of safety and liveness properties of parameterized systems, where relations describe transitions [1, 10, 22, 26]. Another one arises from the study of Automatic Structures [8], where word languages and relations are used to describe infinite structures, and good closure properties are necessary to obtain effective model checking of logics. Another example is the study of formal models underlying IBM's tools for text extraction into a relational model [16]; where several classes of relations emerge (some outside *Rational*) with differing closure properties. Yet another comes from graph databases, which are actively studied as a suitable model for RDF data, social networks data, and others [2]. Paths in graph databases are described by their labels and hence they are abstracted as finite words. These paths need to be compared, for instance, for their degree of similarity, edit distance, or other relations [3, 5, 23]. As a concrete link with the present work we consider CRPQs – a basic query language for graph-structured data. As it was shown in [4], allowing rational relations in CRPQs turns the query evaluation problem undecidable. There have therefore been efforts towards finding subclasses of *Rational* relations that preserve decidability for CRPQs (*e.g.* [5, 17, 6]), often exploiting an effective closure under intersection on the underlying subclass of relations. Part of our motivation for studying closure under intersection stems from our ambition, as future work, to characterize all synchronized classes of relations that can be added to CRPQs while preserving decidability.

Contribution

Our main contribution is a characterization for each of the studied closure properties, the main results can be summarized as follows.

▶ **Theorem.** *For every regular $C \subseteq \mathbb{2}^*$, it is decidable whether $\textsc{Rel}(C)$ is closed under intersection, complement, concatenation, Kleene star and projection.*

While some of the characterizations we give are for arbitrary arity relations, we were only able to show decidability for binary arity. Indeed, the decidability of these characterizations relies, crucially, on the decidability of testing for inclusion between synchronized classes, which has only been shown for binary relations [14].

We do not include closure under union since it can be easily seen that all classes defined in this way are closed under union. The most involved result is closure under intersection. The main property we will prove is that $\textsc{Rel}(C)$ is closed under intersection if, and only if, $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$ for some D whose Parikh-image is *injective* (*i.e.*, there are no two distinct words of D with the same Parikh-image). Further, we show that this can be tested, and such a language D can be effectively constructed, whenever possible. In the same vein, we obtain that $\textsc{Rel}(C)$ is closed under complement if, and only if, $\textsc{Rel}(C) = \textsc{Rel}(D)$ for some D with a *bijective* Parikh-image. (Observe that closure under complement implies closure under intersection in view of the fact that all classes are closed under union.)

Related work

The formalization of the framework to describe synchronized classes of relations has been introduced only recently [18]. As mentioned, the problem of containment between classes of relations has been addressed in [14] for the binary case. The formalism of synchronizations has been also extended beyond rational relations by means of semi-linear constraints [17] in the context of querying graph databases.

The paper [9] studies relations with origin information, as induced by non-deterministic (one-way) finite state transducers. Origin information can be seen as a way to describe a synchronization between input and output words – somehow in the same spirit of our synchronization languages – and was exploited to recover decidability of the equivalence problem for transducers. The paper [19] pursues further this principle by studying "distortions" of the origin information, called resynchronizations. The paper [27] studies the *uniformization problem* for synchronized relations.

Organization

After a preliminary Section 2, we show the main result characterizing closure under intersection in Section 3. In Section 4 we study closure under complement and another variant that we call "relativized complement". In Section 5 we give characterizations for closure under concatenation, Kleene star and projection. We conclude with Section 6.

2 Preliminaries

We denote by $\mathbb{N}$ the set of non-negative integers. $\mathbb{A}, \mathbb{B}$ denote arbitrary finite alphabets and for $k \in \mathbb{N}$, $k \geq 1$, $\mathbb{k}$ denotes the k-letter alphabet $\{1, \ldots, k\}$. For a word $w \in \mathbb{A}^*$, $|w|$ is its length, and $|w|_a$ is the number of occurrences of symbol a in w.

Regular languages

We use standard notation for regular expressions without complement, namely, for expressions built up from the empty set, the empty word ε and the symbols $a \in \mathbb{A}$, using the operations $\cdot$, $\cup$, and $(\,)^*$. For economy of space and clarity we use the abbreviated notation $(\,)^n$, $(\,)^{<n}$, $(\,)^{\geq n}$, $(\,)^{n*}$, and $(\,)^{*n}$ – the last two being shorthands for $((\,)^n)^*$ and $((\,)^*)^n$ respectively. We abuse the notation $(\,)^k$ to also denote the cartesian product of k copies of the same set (typically $(\mathbb{A}^*)^k$) when there is no risk of confusion. We also identify regular expressions with

the defined languages; for example, we may write $abbc \in a \cdot b^{\geq 2} \cdot (c \cup d)^*$, $b(ab)^* = (ba)^*b$ and $\{a,b\}^* \cdot c = (a \cup b)^* \cdot c$. The *star-height* of a regular expression is the maximum number of nested Kleene stars $(\,)^*$. Given $u = a_1 \cdots a_n \in \mathbb{A}^*$ and $v = b_1 \cdots b_n \in \mathbb{B}^*$, we write $u \otimes v$ for the word $(a_1, b_1) \cdots (a_n, b_n) \in (\mathbb{A} \times \mathbb{B})^*$. Similarly, given $U \subseteq \mathbb{A}^*$, $V \subseteq \mathbb{B}^*$, we write $U \otimes V \subseteq (\mathbb{A} \times \mathbb{B})^*$ for the set $\{u \otimes v \colon u \in U, v \in V, |u| = |v|\}$. Given two languages L, L' over $\mathbb{A}$, we write $L \subseteq_{reg} L'$ to denote that L is a regular subset of L'.

A regular expression $C \subseteq \mathbb{2}^*$ is *concat-star*, if it is of the form

$$C = C_1^* u_1 C_2^* u_2 \cdots C_n^* u_n, \tag{$\star$}$$

for $n \in \mathbb{N}$, words $u_1, \ldots, u_n$, and regular expressions $C_1, \ldots, C_n$ where none of the C_i's describes the empty language. The C_i^*'s from $(\star)$ are called *components* of the concat-star. A concat-star expression like $(\star)$ is *smooth* if either $n \leq 2$ or there are no $\ell, s \in \mathbb{2}$ and $1 \leq i < n$ such that $C_i \subseteq \ell^*, C_{i+1} \subseteq s^*$. We say that a regular language L is concat-star (resp. smooth) if it admits a concat-star (resp. smooth) expression.

Parikh-images and linear sets

The *Parikh-image* of $w \in \mathbb{2}^*$ is the pair associating each symbol of $\mathbb{2}$ to its number of occurrences in w, *i.e.* $\pi(w) = (|w|_1, |w|_2)$. We naturally extend this to languages $L \subseteq \mathbb{2}^*$ by letting $\pi(L) \stackrel{\text{def}}{=} \{\pi(w) \colon w \in L\}$ $(\subseteq \mathbb{N}^2)$. A language $C \subseteq \mathbb{2}^*$ is *Parikh-injective* if for every $u, v \in C$, if $\pi(u) = \pi(v)$ then $u = v$; it is *Parikh-surjective* if $\pi(C) = \mathbb{N}^2$; and it is *Parikh-bijective* if it is both Parikh-injective and -surjective. We will use the product order $(\leq, \mathbb{N}^2)$, defined by $(n, m) \leq (n', m')$ iff $n \leq n'$ and $m \leq m'$. Given a vector $\bar{x} \in \mathbb{N}^2$ and a set $X = \{\bar{x}_1, \ldots, \bar{x}_n\} \subseteq \mathbb{N}^2$ (in our case, the Parikh-image of words from $\mathbb{2}^*$), we define the *linear set generated by X and $\bar{x}$* as $\langle \bar{x}, X \rangle = \{\bar{x} + \alpha_1 \cdot \bar{x}_1 + \cdots + \alpha_n \bar{x}_n \colon \alpha_i \in \mathbb{N}\}$. For economy of space we write $\langle X \rangle$ as short for $\langle \bar{0}, X \rangle$, where $\bar{0} = (0, 0)$. Note that, in particular, $\langle \emptyset \rangle = \{\bar{0}\}$. A *semi-linear set* is a finite union of linear sets. The following fact will be useful in the next section.

▶ **Lemma 1.** *For every semi-linear set $V \subseteq \mathbb{N}^2$ there exists a Parikh-injective language $C \subseteq_{reg} \mathbb{2}^*$ such that $\pi(C) = V$.*

Two sets of vectors $X, Y \subseteq \mathbb{N}^2$ are *independent* if $\bar{0} \notin X \cup Y$ and $\langle X \rangle \cap \langle Y \rangle = \{\bar{0}\}$; otherwise they are *dependent*. We say that two languages over $\mathbb{2}$ are *Parikh-independent* (resp. Parikh-dependent) if their Parikh-images are. We abuse notation and say that $\bar{x}$ and $\bar{y}$ are (in)dependent whenever $\{\bar{x}\}$ and $\{\bar{y}\}$ are (in)dependent, and likewise for words. We will need the following simple observation later.

▶ **Observation 2.** *If u and v are Parikh-independent, for every $s, t, s', t' \in \mathbb{N}$, if $\pi(u^s v^{s'}) = \pi(v^t u^{t'})$, then $s' = t$ and $t' = s$.*

Indeed, we have that $s \cdot \pi(u) + s' \cdot \pi(v) = t' \cdot \pi(u) + t \cdot \pi(v)$. Let us assume that $s' \leq t$ (the case in which is $\geq$ is similar). Then $t' \leq s$ and so we have $(s - t') \cdot \pi(u) = (t - s') \cdot \pi(v)$ which implies $s - t' = 0 = t - s'$ since u and v are Parikh-independent. Then $s' = t$ and $t' = s$.

2.1 Synchronized relations

A *synchronization* of a tuple $(w_1, \ldots, w_k)$ of words over $\mathbb{A}$ is a word over $\mathbb{k} \times \mathbb{A}$ such that the projection onto $\mathbb{A}$ of positions labeled by i is exactly w_i, for $i = 1, \ldots, k$. For example, the words $(1,a)(1,b)(2,a)$ and $(1,a)(2,a)(1,b)$ are two possible synchronizations of the same pair (ab, a). Every word $w \in (\mathbb{k} \times \mathbb{A})^*$ is a synchronization of a unique tuple $(w_1, \ldots, w_k)$

of words over $\mathbb{A}$, where for all $i \in \{1,\dots,k\}$, $i^{|w_i|} \otimes w_i$ is the projection of w onto the alphabet $\{i\} \times \mathbb{A}$. We denote such tuple $(w_1,\dots,w_k)$ by $[\![w]\!]_k$ and extend the notation to languages $L \subseteq (\mathbb{k} \times \mathbb{A})^*$ by denoting the unique k-ary relation synchronized by L as $[\![L]\!]_k \stackrel{\text{def}}{=} \{[\![w]\!]_k : w \in L\}$. In our previous example, $[\![(1,a)(1,b)(2,a)]\!]_2 = [\![(1,a)(2,a)(1,b)]\!]_2 = (ab,a)$, and $[\![\{(1,a)(2,a),(1,a)(2,b),(1,b)(2,a),(1,b)(2,b)\}^*]\!]_2$ is the equal-length relation on the alphabet $\{a,b\}$.

In this setup, we define classes of relations by restricting the set of admitted synchronizations. One way of doing so is to fix a language $C \subseteq_{reg} \mathbb{k}^*$, called *control language*, and let L vary over all regular languages over $\mathbb{k} \times \mathbb{A}$ whose projections onto $\mathbb{k}$ are in C. Thus, given $k \in \mathbb{N}$ and $C \subseteq_{reg} \mathbb{k}^*$, we define the class of k-ary *C-controlled relations* as

$$\textsc{Rel}_k(C) \stackrel{\text{def}}{=} \big\{([\![L]\!]_k, \mathbb{A}) : L \subseteq_{reg} C \otimes \mathbb{A}^*, \mathbb{A} \text{ is a finite alphabet}\big\}.$$

Whenever k is clear from the context, we write $[\![w]\!]$, $[\![L]\!]$ and $\textsc{Rel}(C)$. For economy of space, we write $C =_{\textsc{Rel}} D$ as short for $\textsc{Rel}(C) = \textsc{Rel}(D)$, and we say that C is $\textsc{Rel}$-equivalent to D. Similarly, we write $C \subseteq_{\textsc{Rel}} D$ as short for $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$ and we say that C is $\textsc{Rel}$-contained in D. The definition makes explicit the alphabet used for each relation, in contrast to previous definitions of synchronized classes [18, 14]. The reason for this is that in particular we study closure under complement, which requires the alphabet to be specified. However, we observe that synchronized classes are closed under taking super-alphabets, and thus the alphabet can be often disregarded. We then write $R \in \textsc{Rel}(C)$ to denote $(R,\mathbb{A}) \in \textsc{Rel}(C)$ for some $\mathbb{A}$.

▶ **Observation 3.** *If $(R,\mathbb{A}) \in \textsc{Rel}(C)$ then $(R,\mathbb{A}') \in \textsc{Rel}(C)$ for every $\mathbb{A} \subseteq \mathbb{A}'$. If $(R,\mathbb{A}) \in \textsc{Rel}(C)$ then $(R,\mathbb{A}_R) \in \textsc{Rel}(C)$, where $\mathbb{A}_R \subseteq \mathbb{A}$ is the set of symbols present in R.*

Clearly, $C \subseteq_{reg} D \subseteq_{reg} \mathbb{k}^*$ implies $C \subseteq_{\textsc{Rel}} D$, but the converse does not hold: $\textsc{Rel}_2(C_{\text{Rec}})$ $=$ *Recognizable* $\subsetneq$ *Automatic* $= \textsc{Rel}_2(C_{\text{Aut}})$, but $C_{\text{Rec}} \not\subseteq C_{\text{Aut}}$. Moreover, different control languages may induce the same class of synchronized relations. For any two regular $C, D \subseteq_{reg}$ $\mathbb{k}^*$ it is decidable to test whether $C \subseteq_{\textsc{Rel}} D$ in the case $k = 2$ [14], but for arbitrary k-ary relations the decidability of the class containment problem is open. Henceforward, *Rational* will denote the class $\textsc{Rel}(\mathbb{2}^*)$ of rational relations.

We restate some properties from [14] that we will use throughout (the proofs in [14] are for the case $k = 2$ but they can be easily generalized to arbitrary k). We will use the notation $R \cdot S$ to denote the usual concatenation of relations, more specifically, given $R, S \subseteq (\mathbb{A}^*)^k$, $R \cdot S = \{(u \cdot u', v \cdot v') : (u,v) \in R \text{ and } (u',v') \in S\}$.

▶ **Lemma 4** (Lemma 2 of [14]). *For every $C, D, C', D' \subseteq_{reg} \mathbb{k}^*$,*

1. *if $R \in \textsc{Rel}(C \cdot D)$, there are $R_1,\dots,R_n \in \textsc{Rel}(C)$, $R'_1,\dots,R'_n \in \textsc{Rel}(D)$ such that $R = \bigcup_i R_i \cdot R'_i$;*
2. *if $R \in \textsc{Rel}(C^*)$, there are $R_1,\dots,R_n \in \textsc{Rel}(C)$ and $I \subseteq_{reg} \{1,\dots,n\}^*$ such that $R = \bigcup_{w \in I} R_{w[1]} \cdots R_{w[|w|]}$;*
3. *For every $R \in \textsc{Rel}(C \cup D)$, there are $R_1 \in \textsc{Rel}(C)$, $R_2 \in \textsc{Rel}(D)$ such that $R = R_1 \cup R_2$.*
4. *if $C \subseteq D$, then $C \subseteq_{\textsc{Rel}} D$;*
5. *if $C \subseteq_{\textsc{Rel}} D$ and $C' \subseteq_{\textsc{Rel}} D'$, then $C \cdot C' \subseteq_{\textsc{Rel}} D \cdot D'$ and $C \cup C' \subseteq_{\textsc{Rel}} D \cup D'$;*
6. *if $C \subseteq_{\textsc{Rel}} D$, then $C^* \subseteq_{\textsc{Rel}} D^*$;*
7. *for every partition I, J of $\{1,\dots,k\}$ such that $C \subseteq I^*$ and $D \subseteq J^*$, we have $C \cdot D =_{\textsc{Rel}} D \cdot C$;*
8. *if C is finite, then $C \cdot D =_{\textsc{Rel}} D \cdot C$;*
9. *if $C \subseteq_{\textsc{Rel}} D$ then $\pi(C) \subseteq \pi(D)$; moreover, if C is finite, the converse also holds.*

The following decomposition lemma, which is an immediate consequence of [14, Proposition 3 plus Lemma 2 P7] and basic properties from Lemma 4, will be used throughout.

▶ **Lemma 5.** *Every $C \subseteq_{reg} \mathbb{2}^*$ is effectively* Rel*-equivalent to a finite union of smooth languages, i.e. given $C \subseteq_{reg} \mathbb{2}^*$, one can compute a finite set of smooth languages such that C is* Rel*-equivalent to their union.*

In addition to these, our characterization results make use of the following easy properties of relations controlled by Parikh-injective and Parikh-bijective languages.

▶ **Lemma 6.** *For any $C \subseteq_{reg} \mathbb{k}^*$ and $L, M \subseteq_{reg} C \otimes \mathbb{A}^*$,*

1. *if C is Parikh-injective, and $w, w' \in C \otimes \mathbb{A}^*$, then $[\![w]\!] = [\![w']\!]$ implies $w = w'$;*
2. *$[\![L]\!] \cup [\![M]\!] = [\![L \cup M]\!]$;*
3. *if C is Parikh-injective then $[\![L]\!] \cap [\![M]\!] = [\![L \cap M]\!]$ and $[\![L]\!] \setminus [\![M]\!] = [\![L \setminus M]\!]$;*
4. *if C is Parikh-bijective then $(\mathbb{A}^*)^k \setminus [\![L]\!] = [\![(C \otimes \mathbb{A}^*) \setminus L]\!]$;*
5. *if C is Parikh-surjective then $1^* \cdots k^* \subseteq_{\text{Rel}} C$.*

Proof. The first two items follow immediately from definitions.

3. $[\![L \cap M]\!] \subseteq [\![L]\!] \cap [\![M]\!]$ is always true. For the other containment, let $(w_1, \ldots, w_k) \in [\![L]\!] \cap [\![M]\!]$, then there exist $w \in L, w' \in M$ such that $[\![w]\!] = [\![w']\!] = (w_1, \ldots, w_k)$. Since C is Parikh-injective, by item 1, $w = w' \in L \cap M$ synchronizes $(w_1, \ldots, w_k)$ which concludes the proof.
   $[\![L]\!] \setminus [\![M]\!] \subseteq [\![L \setminus M]\!]$ is always true. For the other containment, let $w \in L \setminus M$. Then $[\![w]\!] \in [\![L]\!]$. By way of contradiction, suppose that $[\![w]\!] \in [\![M]\!]$. In this case, there exists $w' \in M$ such that $[\![w]\!] = [\![w']\!]$. Since C is Parikh-injective, by item 1, $M \not\ni w = w' \in M$ which is a contradiction.
4. For $\subseteq$, note that, since C is Parikh-surjective, $(\mathbb{A}^*)^k = [\![C \otimes \mathbb{A}^*]\!]$, and so the result follows from the previous item.
5. We make use of closure under componentwise letter-to-letter relations (*cf.* Lemma 8 of Section 2.2). Suppose $C \subseteq_{reg} \mathbb{k}^*$ is Parikh-surjective, and let $R \in \text{Rel}(1^* \cdots k^*)$. As an immediate consequence of Mezei's theorem, we have the following:

   ▷ Claim 7. For every k, $\text{Rel}(1^* \cdots k^*) = \{\bigcup_{i \in I} L_{i,1} \times \cdots \times L_{i,k} \colon I \text{ is finite and } L_{i,j} \subseteq_{reg} \mathbb{A}^* \text{ for some finite alphabet } \mathbb{A}\}$.

   Then $R = \bigcup_{i \in I} L_{i,1} \times \cdots \times L_{i,k}$ for a finite I and regular languages $L_{i,j}$. For any $i \in I$ and $j \in \mathbb{k}$ consider $T_{i,j}$ as the letter-to-letter relation $T_{i,j} = \{(u, v) : |u| = |v| \text{ and } v \in L_{i,j}\} \in \text{Rel}((12)^*)$. Note that, by Parikh-surjectivity, $U = (\mathbb{A}^*)^k = [\![C \otimes \mathbb{A}^*]\!] \in \text{Rel}(C)$ and therefore $U \circ (T_{i,1}, \ldots, T_{i,k}) = L_{i,1} \times \cdots \times L_{i,k}$. Then, by closure under union and componentwise letter-to-letter relations (Lemma 8), it follows that $R = \bigcup_{i \in I} U \circ (T_{i,1}, \ldots, T_{i,k}) \in \text{Rel}(C)$. ◀

2.2 Universal closure properties

There are some closure properties which are shared by *all* classes of synchronized relations, that is, by every $\text{Rel}(C)$ with $C \subseteq_{reg} \mathbb{k}^*$. We highlight the most salient ones.

An *alphabetic morphism* between two finite alphabets $\mathbb{A}, \mathbb{B}$ is a morphism $h : \mathbb{A}^* \to \mathbb{B}^*$ between the free monoids such that $h(a) \in \mathbb{B}$ for every $a \in \mathbb{A}$. Its application is extended to any relation $R \subseteq (\mathbb{A}^*)^k$ as follows $h(R) = \{(h(u_1), \ldots, h(u_k)) \colon (u_1, \ldots, u_k) \in R\} \subseteq (\mathbb{B}^*)^k$; and its *inverse* is applied to $S \subseteq (\mathbb{B}^*)^k$ as $h^{-1}(S) = \{(u_1, \ldots, u_k) \colon (h(u_1), \ldots, h(u_k)) \in S\} \subseteq (\mathbb{A}^*)^k$. A *letter-to-letter* relation is one from $\text{Rel}((12)^*)$.

We define the following closure properties over classes $\mathcal{C}$ of k-ary relations.

- $\mathcal{C}$ is closed under union if for all $(R, \mathbb{A}), (S, \mathbb{A}) \in \mathcal{C}$, $(R \cup S, \mathbb{A}) \in \mathcal{C}$;
- $\mathcal{C}$ is closed under (inverse) alphabetic morphisms if for all $(R, \mathbb{A}) \in \mathcal{C}$ and $h : \mathbb{A}^* \to \mathbb{B}^*$ (resp. $g : \mathbb{B}^* \to \mathbb{A}^*$) an alphabetic morphism, $(h(R), \mathbb{B}) \in \mathcal{C}$ (resp. $(g^{-1}(R), \mathbb{B}) \in \mathcal{C}$);
- $\mathcal{C}$ is closed under componentwise letter-to-letter relations if for every $(R, \mathbb{A}) \in \mathcal{C}$ and $(T_1, \mathbb{A}), \ldots, (T_k, \mathbb{A}) \in \textsc{Rel}((12)^*)$ the following relation over the alphabet $\mathbb{A}$ is also in $\mathcal{C}$: $R \circ (T_1, \ldots, T_k) \stackrel{\text{def}}{=} \{(u_1, \ldots, u_k) \colon \text{there is } (v_1, \ldots, v_k) \in R \text{ s.t. } (v_i, u_i) \in T_i \text{ for every } i\}$.
- $\mathcal{C}$ is closed under recognizable projections if for all $(R, \mathbb{A}) \in \mathcal{C}$ and $(S, \mathbb{A}) \in \textsc{Rel}(1^* \cdots k^*)$, $(R \cap S, \mathbb{A}) \in \mathcal{C}$.

▶ **Lemma 8.** *For every $k \in \mathbb{N}$ and $C \subseteq_{reg} \mathbb{k}^*$, $\textsc{Rel}(C)$ is closed under union, alphabetic morphisms, inverse alphabetic morphisms, componentwise letter-to-letter relations, and recognizable projections .*

Proof. Closure under union follows from the fact that if $L, L' \subseteq_{reg} C \otimes \mathbb{A}^*$, then $L \cup L' \subseteq_{reg} C \otimes \mathbb{A}^*$ and $[\![L]\!] \cup [\![L']\!] = [\![L \cup L']\!]$ (Lemma 6). Closure under letter-to-letter relations follows from the fact that, given $L \subseteq_{reg} C \otimes \mathbb{A}^*$ and k letter-to-letter relations $T_1, \ldots, T_k$ over $\mathbb{A}$, there exists $L' \subseteq_{reg} C \otimes \mathbb{A}^*$ such that $[\![L']\!] = [\![L]\!] \circ (T_1, \ldots, T_k)$ (one can build an automaton recognizing such language from the automata for $L, T_1, \ldots, T_k$). Since any (inverse) alphabetic morphism can be implemented as a letter-to-letter relation, it follows that $\textsc{Rel}(C)$ is closed under (inverse) alphabetic morphisms. Finally, closure under recognizable projections follows from closure under letter-to-letter relations and closure under union, since for every $R \in \textsc{Rel}(C)$ and $S = \bigcup_{i \in I} L_{i,1} \times \cdots \times L_{i,k} \in \textsc{Rel}(1^* \cdots k^*)$ (recall Claim 7) we have that $R \cap S = \bigcup_{i \in I} R \circ (T_{i,1}, \ldots, T_{i,k})$ for $T_{i,j} = \{(w, w) \colon w \in L_{i,j}\}$. ◀

3 Closure under intersection

We say that a class $\mathcal{C}$ of k-ary relations is *closed under intersection* if for all $(R, \mathbb{A}), (S, \mathbb{A}) \in \mathcal{C}$, $(R \cap S, \mathbb{A}) \in \mathcal{C}$. In this section we show a decidable characterization of the languages $C \subseteq_{reg} \mathbb{2}^*$ for which $\textsc{Rel}(C)$ is closed under intersection. Further, for $C \subseteq_{reg} \mathbb{2}^*$, if $\textsc{Rel}(C)$ is closed under intersection, it is *effectively* closed, that is, for every $R, S \in \textsc{Rel}(C)$ over an alphabet $\mathbb{A}$, one can compute $R \cap S$ as a synchronized relation, that is, as some $L \subseteq_{reg} (\mathbb{2} \times \mathbb{A})^*$ so that $[\![L]\!] = R \cap S$. The main result is the following.

▶ **Theorem 9.** *For every $C \subseteq_{reg} \mathbb{2}^*$, $\textsc{Rel}(C)$ is closed under intersection if, and only if, $C \subseteq_{\textsc{Rel}} D$ for some Parikh-injective $D \subseteq_{reg} \mathbb{2}^*$.*

At the end of this section we give an effective procedure to decide, given $C \subseteq_{reg} \mathbb{2}^*$, whether $\textsc{Rel}(C)$ is closed under intersection. Decidability can be seen as the fact that the set of languages $C \subseteq_{reg} \mathbb{2}^*$ for which there is a Parikh-injective language $D \subseteq_{reg} \mathbb{2}^*$ such that $C \subseteq_{\textsc{Rel}} D$ is both computably enumerable and co-computably enumerable. While showing that it is c.e. is straightforward, proving co-c.e. involves all the developments of this section. Concretely, we define some *bad conditions* that characterize all languages C such that $\textsc{Rel}(C)$ is *not* closed under intersection, and in this way we obtain that the set of languages $C \subseteq_{reg} \mathbb{2}^*$ which satisfy any of the bad conditions is c.e.

We will start by giving a sufficient condition for $\textsc{Rel}(C)$ to be closed under intersection. The following simple lemma (which was already proved in [18]) follows from Lemma 6.

▶ **Lemma 10.** *If $C \subseteq_{reg} \mathbb{2}^*$ is Parikh-injective, then $\textsc{Rel}(C)$ is closed under intersection.*

This lemma implies that any language which is REL-equivalent to a Parikh-injective one gives rise to a closed under intersection class. A natural question is whether the converse holds but it doesn't seem to. For instance, if $C = 1^*2^* \cup (12)^*$, $\text{REL}(C)$ is closed under intersection but it seems unlikely that C is REL-equivalent to a Parikh-injective language.

Another sufficient condition for $\text{REL}(C)$ to be closed under intersection is that $C =_{\text{REL}} D \cup X$ for some Parikh-injective $D, X \subseteq_{reg} \mathbb{2}^*$ such that $X \subseteq_{\text{REL}} 1^*2^*$ (in fact, it can be seen that injectivity of X is not really necessary). We will prove that this condition is also necessary, and thus we will have another characterization of closure under intersection. This is not obvious and we will prove a stronger statement, which we present below (Theorem 12). Also, in particular, we will show that if $\text{REL}(C)$ is closed under intersection, we can compute a Parikh-injective $D \subseteq_{reg} \mathbb{2}^*$ such that $C \subseteq_{\text{REL}} D$, which allows us in turn to compute the intersection of two relations in $\text{REL}(C)$ as a synchronized relation.

For $C \subseteq_{reg} \mathbb{2}^*$, we denote by $\text{REL}(C)^\cap$ the closure under intersection of $\text{REL}(C)$, *i.e.*, the smallest class of relations containing $\text{REL}(C)$ and being closed under intersection. We present three properties on $C \subseteq_{reg} \mathbb{2}^*$ that we call the *bad conditions*, which will characterize the languages such that $\text{REL}(C)$ is *not* closed under intersection.

Bad conditions

For $C \subseteq_{reg} \mathbb{2}^*$, consider the following properties:

(A) There exist $u_1, u_2, v, z \in \mathbb{2}^*$ such that

1. u_i and v are Parikh-independent for $i = 1, 2$,
2. $\pi(u_i) \geq (1, 1)$ for some i,
3. $\{u_1, u_2\}$ and $\{v\}$ are Parikh-dependent,
4. $u_1^* u_2^* z \subseteq_{\text{REL}} C$ and $v^* z \subseteq_{\text{REL}} C$.

(B) There exist $u, v, z \in \mathbb{2}^*$ such that

1. u and v are Parikh-independent,
2. $\pi(u) \geq (1, 1)$ or $\pi(v) \geq (1, 1)$,
3. $u^* v^* z \subseteq_{\text{REL}} C$ and $v^* u^* z \subseteq_{\text{REL}} C$.

(C) There exist $u, v, w, z \in \mathbb{2}^*$ such that

1. $u \in 1^* \setminus \{\varepsilon\}$, $w \in 2^* \setminus \{\varepsilon\}$,
2. $\pi(v) \geq (1, 1)$,
3. $u^* v^* w^* z \subseteq_{\text{REL}} C$ or $w^* v^* u^* z \subseteq_{\text{REL}} C$.

For example, $1^*(12)^*(122)^*$ satisfies A for $u_1 = 1$, $u_2 = 122$, $v = 12$, $z = \varepsilon$; $1^*(12)^*1^*$ satisfies B for $u = 1$, $v = 12$, $z = \varepsilon$; and $1^*(12)^*2^*$ satisfies C for $u = 1$, $v = 12$, $w = 2$, $z = \varepsilon$.

▶ **Observation 11.** *The bad conditions are $\subseteq_{\text{REL}}$-upward closed: If $C \subseteq_{\text{REL}} D$ and C satisfies property* A *(resp.* B*,* C*), then D also satisfies property* A *(resp.* B*,* C*).*

We can now present the characterization theorem.

▶ **Theorem 12.** *For $C \subseteq_{reg} \mathbb{2}^*$, the following are equivalent:*

1. $\text{REL}(C)$ *is closed under intersection (i.e. $\text{REL}(C)^\cap = \text{REL}(C)$);*
2. $\text{REL}(C)^\cap$ *is definable (i.e. there exists $D \subseteq_{reg} \mathbb{2}^*$ such that $\text{REL}(C)^\cap = \text{REL}(D)$);*
3. $\text{REL}(C)^\cap \subseteq$ *Rational;*

4. *for all* $R, S \in \text{Rel}(C)$, $R \cap S \in$ *Rational;*
5. C *does not satisfy any of the bad conditions;*
6. *there exist* $D, X \subseteq_{reg} \mathbb{2}^*$ *Parikh-injective such that* $C =_{\text{Rel}} D \cup X$ *and* $X \subseteq_{\text{Rel}} 1^*2^*$*;*
7. *there exists* $D \subseteq_{reg} \mathbb{2}^*$ *Parikh-injective such that* $C \subseteq_{\text{Rel}} D$*.*

From $1 \Leftrightarrow 7$ and transitivity of $\subseteq_{\text{Rel}}$, closure under intersection is $\subseteq_{\text{Rel}}$-downward closed:

▶ **Corollary 13.** *For* $C, D \subseteq_{reg} \mathbb{2}^*$*, if* $C \subseteq_{\text{Rel}} D$ *and* $\text{Rel}(D)$ *is closed under intersection, then* $\text{Rel}(C)$ *is closed under intersection.*

We first explain the main proof strategy for obtaining Theorem 12, and present the three key technical results we will need to prove (Propositions 14, 16 and 17).

Proof idea of Theorem 12

The proof strategy is by showing $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 6 \Rightarrow 1$ on the one hand, and $6 \Rightarrow 7 \Rightarrow 3$ on the other hand. First observe that $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4$ are trivial. We next prove $6 \Rightarrow 1$, $7 \Rightarrow 3$ and $6 \Rightarrow 7$.

For $6 \Rightarrow 1$, suppose that $C =_{\text{Rel}} D \cup X$ for some Parikh-injective languages D, X such that $X \subseteq_{\text{Rel}} 1^*2^*$. Let $R, S \in \text{Rel}(C)$. Then, by item 3 of Lemma 4, there exist $R_1, S_1 \in \text{Rel}(D)$, $R_2, S_2 \in \text{Rel}(X)$ such that $R = R_1 \cup R_2$ and $S = S_1 \cup S_2$. Note that:

- $R_1 \cap S_1 \in \text{Rel}(D) \subseteq \text{Rel}(C)$ by Lemma 10 applied to D;
- $R_2 \cap S_2 \in \text{Rel}(X) \subseteq \text{Rel}(C)$ by Lemma 10 applied to X; and
- $R_1 \cap S_2, R_2 \cap S_1 \in \text{Rel}(D)$ by closure under recognizable projections (Lemma 8).

It only remains to observe that $R \cap S = (R_1 \cap S_1) \cup (R_1 \cap S_2) \cup (R_2 \cap S_1) \cup (R_2 \cap S_2)$ and obtain that $R \cap S \in \text{Rel}(C)$ due to closure under union (Lemma 8).

On the other hand, $7 \Rightarrow 3$ can be derived from $1 \Rightarrow 3$. Indeed, suppose that $C \subseteq_{\text{Rel}} D$ for some Parikh-injective language D. By Lemma 10, $\text{Rel}(D)$ is closed under intersection and so, by $1 \Rightarrow 3$, $\text{Rel}(C)^\cap \subseteq \text{Rel}(D)^\cap \subseteq$ *Rational*.

For $6 \Rightarrow 7$, suppose that $C =_{\text{Rel}} D \cup X$ for some Parikh-injective languages D, X such that $X \subseteq_{\text{Rel}} 1^*2^*$. By Lemma 1, closure under complement of semi-linear sets and Parikh's Theorem [25], it follows that there exists $\hat{D} \subseteq_{reg} \mathbb{2}^*$ Parikh-injective such that $\pi(\hat{D}) = \mathbb{N}^2 \setminus \pi(D)$. Note that $D \cup \hat{D}$ is Parikh-bijective. Since $D \cup \hat{D}$ is Parikh-surjective, by Lemma 6, item 5, $X \subseteq_{\text{Rel}} 1^*2^* \subseteq_{\text{Rel}} D \cup \hat{D}$ and so, by Lemma 4, item 3 plus closure under union of $D \cup \hat{D}$, we have $C =_{\text{Rel}} D \cup X \subseteq_{\text{Rel}} D \cup \hat{D}$.

The main difficulty will lie on the proofs of $4 \Rightarrow 5$ and $5 \Rightarrow 6$. For $4 \Rightarrow 5$, we will prove the contrapositive statement:

▶ **Proposition 14.** *If* $C \subseteq_{reg} \mathbb{2}^*$ *satisfies any of the bad conditions, then there exist* $R, S \in \text{Rel}(C)$ *such that* $R \cap S \notin$ *Rational.*

To prove $5 \Rightarrow 6$, we define some basic regular languages over $\mathbb{2}$ that we call *basic injective*. A language $C \subseteq \mathbb{2}^*$ is *basic injective* if it can be expressed as u^*v^*z for $u, v, z \in \mathbb{2}^*$ such that if $u, v \neq \varepsilon$, then u and v are Parikh-independent. In particular this implies the following.

▶ **Lemma 15.** *Every basic injective language is Parikh-injective.*

Proof. Let $C = u^*v^*z$ be basic injective. The cases in which u and/or v are empty are straightforward. We will then assume that u and v are Parikh-independent. Suppose then that $\pi(u^r v^s z) = \pi(u^{r'} v^{s'} z)$ for some $r, s, r', s' \in \mathbb{N}$. By Observation 2, $r = r'$ or $s = s'$ which concludes the proof. ◀

Note that singleton sets and languages of the form u^*z for u an arbitrary word are basic injective. The interest of basic injective languages stems from the fact that we can prove the following two results, from which it is not hard to get $5 \Rightarrow 6$.

▶ **Proposition 16.** *If $C \subseteq_{reg} \mathbb{2}^*$ does not satisfy any of the bad conditions, then C is* Rel*-equivalent to a finite union of basic injective languages.*

▶ **Proposition 17.** *If C is a finite union of basic injective languages that are not* Rel*-contained in 1^*2^* and C does not satisfy any of the bad conditions, then C is* Rel*-equivalent to a Parikh-injective regular language.*

To show $5 \Rightarrow 6$ from the two statements above, suppose that C does not satisfy any of the bad conditions. By Proposition 16, $C =_{\text{Rel}} X' \cup D'$, for $X' = \bigcup_{i \in I} X_i$ and $D' = \bigcup_{j \in J} D_j$, where I, J are finite, for every $i \in I$, X_i is basic injective and $X_i \subseteq_{\text{Rel}} 1^*2^*$, and for every $j \in J$, D_j is basic injective and $D_j \not\subseteq_{\text{Rel}} 1^*2^*$. Note that, from the definition of basic injective plus [14, Proposition 7] plus basic properties from Lemma 4, it follows readily that for each $i \in I$, there exist $\ell, s, \ell', s' \in \mathbb{N}$ such that X_i is Rel-equivalent to $1^{\ell *}1^{\ell'}2^{s*}2^{s'}$. Therefore X' is Rel-equivalent to a Parikh-injective language X such that $X \subseteq_{\text{Rel}} 1^*2^*$. On the other hand, by Observation 11, since $D' \subseteq_{\text{Rel}} C$ and C does not satisfy any of the bad condition, neither does D'. Hence, by Proposition 17, D' is Rel-equivalent to a Parikh-injective regular language D. Thus $C =_{\text{Rel}} X \cup D$ which concludes the proof.

We dedicate the rest of the section to prove Propositions 14, 16 and 17.

Proof idea of Proposition 14

We show the proof idea for condition A. The other two conditions follow a similar proof strategy. Suppose that condition A holds, and consider the 3-letter alphabet $\mathbb{A} = \{a_1, a_2, c\}$. Let R, S be the following relations in $(\mathbb{A}^*)^2$:

$$R = [\![(u_1^* \otimes a_1^*) \cdot (u_2^* \otimes a_2^*) \cdot z \otimes c^*]\!], \qquad S = [\![(v^* \otimes \{a_1, a_2\}^*) \cdot z \otimes c^*]\!],$$

note that $R, S \in \text{Rel}(C)$ by condition A.4. It is not hard to show that $|R \cap S| = \infty$ due to condition A.3. We show that $R \cap S \notin \mathit{Rational}$. By means of contradiction, suppose there is an automaton over the alphabet $\mathbb{2} \times \mathbb{A}$ such that the language recognized by this automaton synchronizes $R \cap S$. Since the language is infinite, there is a non-trivial cycle $q_0 \xrightarrow{w_1} q \xrightarrow{w_2} q \xrightarrow{w_3} q_f$ inside some accepting run. By a pumping argument, it can be seen that: 1) $[\![w_2]\!]$ is necessarily of the form (a_i^s, a_i^t) for some i, s, t partly due to A.2; 2) $(s, t) \in \langle\{\pi(u_j)\}\rangle$ for some j; and 3) $(s, t) \in \langle\{\pi(v)\}\rangle$. Since 2) plus 3) are in contradiction with A.1, it follows that $R \cap S \notin \mathit{Rational}$. ◀

Proof idea of Proposition 16

It can be seen that one can reduce to the case in which C is of the form $w_1^* \cdots w_n^* z$ with w_i and w_{i+1} Parikh-independent for all $i = 1, \ldots, n-1$. For this kind of languages, if $n \leq 2$ the result follows trivially since they are already basic injective. A straightforward case inspection shows that if $n \geq 3$ then at least one of the bad conditions holds. ◀

Proof idea of Proposition 17

In order to prove Proposition 17 we show the following stronger statement, which gives a characterization of closure under intersection based on the decomposition into basic injective languages. We denote the *commutative closure* of a language $C \subseteq_{reg} \mathbb{2}^*$ by $[C]_\pi = \{w \in \mathbb{2}^* : \pi(w) \in \pi(C)\}$.

▶ **Lemma 18.** *Given a finite set of basic injective languages* $\{B_i\}$ *that are not* Rel*-contained in* 1^*2^**, the following are equivalent:*

1. $\mathrm{Rel}(\bigcup_i B_i)$ *is closed under intersection;*
2. *for all* $R, S \in \mathrm{Rel}(\bigcup_i B_i)$, $R \cap S \in$ *Rational;*
3. $\bigcup_i B_i$ *does not satisfy any of the bad conditions;*
4. *for every* i, j $B_i \cup B_j$ *does not satisfy any of the bad conditions;*
5. *for every* i, j, $B_i \cap [B_j]_\pi$ *is regular and* $B_i \cap [B_j]_\pi \subseteq_{\mathrm{Rel}} B_j$*;*
6. $\bigcup_i B_i =_{\mathrm{Rel}} C$ *for some Parikh-injective* $C \subseteq_{reg} 2^*$.

Proposition 17 follows from Lemma 18 since it is its implication $3 \Rightarrow 6$. In order to give a proof for Lemma 18, we first define the following property, which is at the core of the next lemmas. A pair of languages B_1, B_2, is said to verify the *dichotomy property* if either

- $B_1 \cup B_2$ satisfies one of the bad conditions; or
- $B_1 \cap [B_2]_\pi$ is regular and $B_1 \cap [B_2]_\pi \subseteq_{\mathrm{Rel}} B_2$.

Note that $B_1 \cap [B_2]_\pi$ may not be regular in general, for example if $B_1 = 1^*2^*$ and $B_2 = (12)^*$. The main ingredient to prove Lemma 18 is given by the following statement.

▶ **Lemma 19.** *Every pair of basic injective languages* B_1, B_2 *such that* $B_1, B_2 \not\subseteq_{\mathrm{Rel}} 1^*2^*$ *satisfies the dichotomy property.*

Proof of Lemma 18. $1 \Rightarrow 2$ is trivial; $2 \Rightarrow 3$ follows from the contrapositive of Proposition 14; $3 \Rightarrow 4$ holds by Observation 11; and $4 \Rightarrow 5$ follows from Lemma 19. For $5 \Rightarrow 6$, we proceed by induction on the number of basic injective languages in $\{B_i\}$. The base case is the empty language, which is (vacuously) Parikh-injective. For the inductive step, consider a union $B \cup \bigcup_i B_i$. First observe that, by Lemma 15, B is Parikh-injective. By inductive hypothesis, there exists $D \subseteq_{reg} 2^*$ Parikh-injective such that $\bigcup_i B_i =_{\mathrm{Rel}} D$. Also, since $B \cap [\bigcup_i B_i]_\pi = \bigcup_i B \cap [B_i]_\pi$, by hypothesis both $B \cap [\bigcup_i B_i]_\pi$ and $B \setminus [\bigcup_i B_i]_\pi$ are regular, and $B \cap [\bigcup_i B_i]_\pi \subseteq_{\mathrm{Rel}} \bigcup_i B_i$. Now it only remains to observe that $(B \setminus [\bigcup_i B_i]_\pi) \cup D$ is Parikh-injective and Rel-equivalent to $B \cup \bigcup_i B_i$. Finally, $6 \Rightarrow 1$ follows from Lemma 10. ◀

Decidability

We finally discuss briefly the decidability procedure to test whether a class $\mathrm{Rel}(C)$ is closed under intersection.

▶ **Proposition 20.** *It is decidable wether a given* $C \subseteq_{reg} 2^*$ *is such that* $\mathrm{Rel}(C)$ *is closed under intersection.*

Proof idea. It follows by the equivalence $1 \Leftrightarrow 5 \Leftrightarrow 7$ of Theorem 12, together with the fact that the set of languages $C \subseteq_{reg} 2^*$ for which there is a Parikh-injective language $D \subseteq_{reg} 2^*$ such that $C \subseteq_{\mathrm{Rel}} D$ is computably enumerable; and the fact that the set of languages $C \subseteq_{reg} 2^*$ which satisfy any of the bad conditions is computably enumerable. ◀

Note that whenever $\mathrm{Rel}(C)$ is closed under intersection, it is effectively so: given $L_1, L_2 \subseteq_{reg} C \otimes \mathbb{A}^*$ it is possible to compute $L \subseteq_{reg} (2 \times \mathbb{A})^*$ with $[\![L]\!] = [\![L_1]\!] \cap [\![L_2]\!]$. Indeed, by the previous proposition we can compute some Parikh-injective D such that $C \subseteq_{\mathrm{Rel}} D$. By the results of [14], one can compute $L_1', L_2' \subseteq_{reg} D \otimes \mathbb{A}^*$ such that $[\![L_1']\!] = [\![L_1]\!]$ and $[\![L_2']\!] = [\![L_2]\!]$; and thus $L = L_1' \cap L_2'$ is such that $[\![L]\!] = [\![L_1]\!] \cap [\![L_2]\!]$ due to injectivity of D and Lemma 6.

4 Closure under complement

We say that a class $\mathcal{C}$ of k-ary relations is *closed under complement* if for every $(R, \mathbb{A}) \in \mathcal{C}$, $((\mathbb{A}^*)^k \setminus R, \mathbb{A}) \in \mathcal{C}$. For every $\text{REL}_k(C)$ and alphabet $\mathbb{A}$, note that there is a unique largest relation $(U, \mathbb{A}) \in \text{REL}_k(C)$ that contains all relations $(R, \mathbb{A}) \in \text{REL}_k(C)$; this is $U = [\![C \otimes \mathbb{A}^*]\!]_k$. Thus, a natural alternative definition for complement could take U, instead of $(\mathbb{A}^*)^k$, as the universe. We say that $\text{REL}_k(C)$ is *closed under relativized complement* if for all $(R, \mathbb{A}) \in \text{REL}_k(C)$ we have $([\![C \otimes \mathbb{A}^*]\!]_k \setminus R, \mathbb{A}) \in \text{REL}_k(C)$. In this section, we give effective characterizations of the languages $C \subseteq_{reg} \mathbb{2}^*$ for which $\text{REL}(C)$ is closed under complement and relativized complement.

Relativized complement

We show that closure under relativized complement, perhaps surprisingly, is equivalent to closure under intersection, and therefore it is decidable whether $\text{REL}(C)$ is closed under relativized complement for a given $C \subseteq_{reg} \mathbb{2}^*$.

▶ **Proposition 21.** *For $C \subseteq_{reg} \mathbb{2}^*$, $\text{REL}(C)$ is closed under relativized complement if, and only if, $\text{REL}(C)$ is closed under intersection.*

Proof. For the left-to-right direction, let $(R, \mathbb{A}), (S, \mathbb{A}) \in \text{REL}(C)$. Recall that $\text{REL}(C)$ is always closed under union and note that $R \cap S = [\![C \otimes \mathbb{A}^*]\!] \setminus ((([\![C \otimes \mathbb{A}^*]\!] \setminus R) \cup ([\![C \otimes \mathbb{A}^*]\!] \setminus S))$, and therefore $(R \cap S, \mathbb{A}) \in \text{REL}(C)$. For the right-to-left direction, let $L \subseteq_{reg} C \otimes \mathbb{A}^*$. We want to check that $[\![C \otimes \mathbb{A}^*]\!] \setminus [\![L]\!] \in \text{REL}(C)$. By the characterization of the previous section (Theorem 12, implication $1 \Rightarrow 6$) we can assume that $C = D \cup X$, for $X \subseteq_{\text{REL}} 1^*2^*$ and X, D Parikh-injective. Then,

$$\begin{aligned}
[\![C \otimes \mathbb{A}^*]\!] \setminus [\![L]\!] &= [\![(D \cup X) \otimes \mathbb{A}^*]\!] \setminus [\![L]\!] = ([\![(D \otimes \mathbb{A}^*) \cup (X \otimes \mathbb{A}^*)]\!]) \setminus [\![L]\!] \\
&= ([\![D \otimes \mathbb{A}^*]\!] \cup [\![X \otimes \mathbb{A}^*]\!]) \setminus [\![L]\!] = ([\![D \otimes \mathbb{A}^*]\!] \setminus [\![L]\!]) \cup ([\![X \otimes \mathbb{A}^*]\!] \setminus [\![L]\!]) \\
&= \underbrace{[\![(D \otimes \mathbb{A}^*) \setminus L]\!]}_{R} \cup \underbrace{[\![X \otimes \mathbb{A}^* \setminus L]\!]}_{S}. \qquad \text{(by Lemma 6, item 3)}
\end{aligned}$$

Since $R, S \in \text{REL}(C)$, by Lemma 8, $R \cup S \in \text{REL}(C)$, and thus $[\![C \otimes \mathbb{A}^*]\!] \setminus [\![L]\!] \in \text{REL}(C)$. ◀

Note that if $C \subseteq \mathbb{2}^*$ is Parikh-surjective, then $[\![C \otimes \mathbb{A}^*]\!] = (\mathbb{A}^*)^2$, and hence closure under relativized complement and closure under complement coincide. Thus, by Proposition 21:

▶ **Observation 22.** *If $C \subseteq \mathbb{2}^*$ is Parikh-surjective, then $\text{REL}(C)$ is closed under complement if, and only if, $\text{REL}(C)$ is closed under intersection.*

Complement

Let $\text{REL}(C)^c$ be the closure under complement of $\text{REL}(C)$, *i.e.*, the smallest class closed under complement containing $\text{REL}(C)$. The following lemma gives sufficient conditions for our characterization.

▶ **Lemma 23.** *For any $C \subseteq_{reg} \mathbb{2}^*$,*
1. *if C is Parikh-bijective, then $\text{REL}(C)$ is closed under complement;*
2. *if $\text{REL}(C)$ is closed under complement, then C is Parikh-surjective.*

Proof. For item 1, let $L \subseteq_{reg} C \otimes \mathbb{A}^*$. By item 4 of Lemma 6, $(\mathbb{A}^*)^2 \setminus [\![L]\!] = [\![C \otimes \mathbb{A}^* \setminus L]\!] \in \text{REL}(C)$ which concludes the proof.

For item 2, let $L = C \otimes \{a\}^*$. Then $((\{a\}^*)^2 \setminus [\![L]\!], \{a\}) \in \textsc{Rel}(C)$ and so there exists $L' \subseteq_{reg} C \otimes \{a\}^*$ such that $[\![L']\!] = (\{a\}^*)^2 \setminus [\![L]\!]$. Then $[\![L \cup L']\!] = [\![L]\!] \cup [\![L']\!] = (\{a\}^*)^2$. Therefore, the Parikh-image of the projection of $L \cup L'$ onto the first component must be $\mathbb{N}^k$ and so C is Parikh-surjective since both L and L' (and hence $L \cup L'$) are $\subseteq_{reg} C \otimes \{a\}^*$. ◀

From Lemma 23 plus Observation 22, we have that $\textsc{Rel}(C)$ is closed under complement if, and only if, $\textsc{Rel}(C)$ is closed under intersection and C is Parikh-surjective. At the end of this section, we will use this to prove that closure under complement is a decidable property.

We now give a characterization for closure under complement without referring to closure under intersection.

▶ **Theorem 24.** *For $C \subseteq_{reg} 2^*$, the following are equivalent:*
1. *there exists $D \subseteq_{reg} 2^*$ Parikh-bijective such that $C =_{\textsc{Rel}} D$;*
2. *$\textsc{Rel}(C)$ is closed under complement (i.e. $\textsc{Rel}(C)^c = \textsc{Rel}(C)$);*
3. *$\textsc{Rel}(C)^c$ is definable (i.e. there is $D \subseteq_{reg} 2^*$ such that $\textsc{Rel}(C)^c = \textsc{Rel}(D)$).*

Before proving the above theorem, we observe that we cannot obtain the third and fourth equivalent statements that we have in Theorem 12.

▶ **Lemma 25.** *There is $C \subseteq_{reg} 2^*$ with $\textsc{Rel}(C)^c \subseteq$ Rational but $\textsc{Rel}(C)^c$ not definable.*

Proof. Consider any language which is Parikh-injective but not Parikh-surjective, *e.g.* $C = (12)^*$. Then, by item 2 of Lemma 23, plus Theorem 24, we have that $\textsc{Rel}(C)^c$ is not definable. The result is then an immediate consequence of the following:

▷ Claim. If $C \subseteq_{reg} 2^*$ is Parikh-injective, then $\textsc{Rel}(C)^c \subseteq$ *Rational*.

Indeed, by Parikh's Theorem [25], $\pi(C)$ is a semi-linear set and then so is $\mathbb{N}^2 \setminus \pi(C)$ (see for example [21]). By Lemma 1, $\mathbb{N}^2 \setminus \pi(C) = \pi(D)$ for some Parikh-injective language D. It follows then that $C \cup D$ is Parikh-bijective and so, by Lemma 23, item 1, $\textsc{Rel}(C \cup D)$ is closed under complement. Then $\textsc{Rel}(C)^c \subseteq \textsc{Rel}(C \cup D) \subseteq$ *Rational*. ◀

Proof of Theorem 24. $1 \Rightarrow 2$ follows from item 1 of Lemma 23; and $2 \Rightarrow 3$ is trivial.

$2 \Rightarrow 1$: Suppose that $\textsc{Rel}(C)$ is closed under complement. By Lemma 8, $\textsc{Rel}(C)$ is also closed under union and so under intersection. Therefore, by Theorem 12, there exist Parikh-injective languages $D, X \subseteq_{reg} 2^*$ such that $X \subseteq_{\textsc{Rel}} 1^*2^*$ and $C =_{\textsc{Rel}} D \cup X$. It follows then that $[\![D \otimes \mathbb{A}^*]\!] \in \textsc{Rel}(C)$ and so $R = (\mathbb{A}^*)^2 \setminus [\![D \otimes \mathbb{A}^*]\!] \in \textsc{Rel}(C)$. Let $L \subseteq_{reg} (D \cup X) \otimes \mathbb{A}^*$ be such that $[\![L]\!] = R$. By definition of R, we get that $L \subseteq_{reg} X \otimes \mathbb{A}^*$ and so $X' = \{u \colon \exists v \text{ such that } u \otimes v \in L\} \subseteq_{reg} X$. Besides, also by definition of R, $\pi(X') = \mathbb{N}^2 \setminus \pi(D)$ and so $D \cup X'$ is Parikh-bijective. It only remains to observe that $C =_{\textsc{Rel}} D \cup X'$: $\supseteq$ is trivial and $\subseteq$ follows from the fact that 1^*2^* is $\textsc{Rel}$-contained in any Parikh-surjective language (Lemma 6, item 5) and so $X \subseteq_{\textsc{Rel}} D \cup X'$.

$3 \Rightarrow 2$: Let $D \subseteq_{reg} 2^*$ such that $\textsc{Rel}(C)^c = \textsc{Rel}(D)$. Since $\textsc{Rel}(D)$ is closed under complement, by $2 \Rightarrow 1$, we can assume wlog that D is Parikh-bijective. By means of contradiction, suppose that $\textsc{Rel}(C)$ is not closed under complement. Therefore, by Observation 22, either C is not Parikh-surjective or $\textsc{Rel}(C)$ is not closed under intersection. We show that in both cases we arrive to a contradiction. If $\textsc{Rel}(C)$ is not closed under intersection, by Theorem 12 (implication $\neg 1 \Rightarrow \neg 4$), there are $R, S \in \textsc{Rel}(C)$ such that $R \cap S \notin$ *Rational*; but since $R \cap S = (\mathbb{A}^*)^2 \setminus ((\mathbb{A}^*)^2 \setminus R \cup (\mathbb{A}^*)^2 \setminus S) \in \textsc{Rel}(C)^c = \textsc{Rel}(D) \subseteq$ *Rational* (recall that $\textsc{Rel}(D)$ is closed under union by Lemma 8), we have a contradiction. On the other hand, if C is not Parikh-surjective, there exists $\overline{x} \in \pi(D) \setminus \pi(C)$. Let $u \in D$

be such that $\pi(u) = \overline{x}$ and let us consider the singleton relation $R = \{[\![u \otimes a^{|u|}]\!]\}$. It is clear that $(R, \{a, b\}) \in \textsc{Rel}(D) = \textsc{Rel}(C)^c$. Then, either $(R, \{a, b\})$ or its complement $((\{a, b\}^*)^2 \setminus R, \{a, b\})$ should be in $\textsc{Rel}(C)$. But it is easy to see that both relations contain a tuple with Parikh-image $\bar{x}$: $[\![u \otimes a^{|u|}]\!] \in R$ and $[\![u \otimes b^{|u|}]\!] \in (\{a, b\}^*)^2 \setminus R$. Since $\bar{x} \notin \pi(C)$, none of the relations is in $\textsc{Rel}(C)$, which is a contradiction. ◀

Decidability

From Observation 22 and item 2 of Lemma 23, decidability of closure under complement follows immediately: $\textsc{Rel}(C)$ is closed under complement if, and only if, $\textsc{Rel}(C)$ is closed under intersection and C is Parikh-surjective. The former is decidable due to Proposition 20, and the latter is decidable through Parikh's Theorem, since universality for semi-linear sets is decidable (see, *e.g.*, [21]).

▶ **Proposition 26.** *Given $C \subseteq_{reg} \mathbb{2}^*$, testing whether $\textsc{Rel}(C)$ is closed under complement is decidable.*

5 Closure under concatenation, Kleene star, and projection

In this section, we characterize languages $C \subseteq_{reg} \mathbb{k}^*$ such that $\textsc{Rel}(C)$ is closed under concatenation, Kleene star, and projection.

- $\mathcal{C}$ is *closed under concatenation* if for all $R, S \in \mathcal{C}$, $R \cdot S \in \mathcal{C}$, where $\cdot$ is the component-wise concatenation operation (*e.g.*, $\{(a, ab), (b, a)\} \cdot \{(b, c)\} = \{(ab, abc), (bb, ac)\}$);
- $\mathcal{C}$ is *closed under Kleene star* if for all $R \in \mathcal{C}$, $R^* \in \mathcal{C}$ for $R^* = \bigcup_{i \in \mathbb{N}} R^{(i)}$, where $R^{(0)} = \{(\varepsilon, \ldots, \varepsilon)\}$, and $R^{(i+1)} = R \cdot R^{(i)}$;
- $\mathcal{C}$ is *closed under projection* if for all $(R, \mathbb{A}) \in \mathcal{C}$ and $K \subseteq \mathbb{k}$, $(R|_K, \mathbb{A}) \in \mathcal{C}$, where $R|_K \subseteq \mathbb{A}^{*k}$ is the projection of R onto the components in K (with ε in the other components). For example, for $R = \{(aa, ab, b), (a, bbb, aab), (aa, ab, ba)\}$ and $K = \{1, 2\}$ we have $R|_K = \{(aa, ab, \varepsilon), (a, bbb, \varepsilon)\}$.

We now give characterizations for closure under concatenation and Kleene star. As we show, closure under concatenation is in fact a necessary condition for closure under Kleene star.

▶ **Proposition 27.** *For every $C, C_1, C_2, C_3 \subseteq_{reg} \mathbb{k}^*$,*

1. *$C_1 \cdot C_2 \subseteq_{\textsc{Rel}} C_3$ iff for every $R_1 \in \textsc{Rel}(C_1)$, $R_2 \in \textsc{Rel}(C_2)$ we have $R_1 \cdot R_2 \in \textsc{Rel}(C_3)$;*
2. *$\textsc{Rel}(C)$ is closed under concatenation iff $C \cdot C \subseteq_{\textsc{Rel}} C$;*
3. *if $\textsc{Rel}(C)$ is closed under Kleene star, then it is closed under concatenation; and*
4. *$\textsc{Rel}(C)$ is closed under Kleene star iff $C^* \subseteq_{\textsc{Rel}} C$.*

Proof sketch. For the left-to-right direction of item 1, let $L_1 \subseteq_{reg} C_1 \otimes \mathbb{A}^*$ and $L_2 \subseteq_{reg} C_2 \otimes \mathbb{A}^*$. Then we only have to observe that $[\![L_1]\!] \cdot [\![L_2]\!] = [\![L_1 \cdot L_2]\!] \in \textsc{Rel}(C_1 \cdot C_2) \subseteq \textsc{Rel}(C_3)$ as we wanted. The right-to-left direction follows from Lemma 8 together with property 1 of Lemma 4. Note that item 2 is a particular case of item 1.

We now turn to item 3. For simplicity assume $k = 2$. Suppose $\textsc{Rel}(C)$ is closed under Kleene star, and take arbitrary $R_1, R_2 \in \textsc{Rel}(C)$ over an alphabet $\mathbb{A}$. Define R_i' over the alphabet $\mathbb{A} \times \{\overline{lst_i}, lst_i\}$ as the result of replacing every pair $(a_1 \cdots a_n, b_1 \cdots b_m) \in R_i$ with $((a_1, \overline{lst_i}) \cdots (a_{n-1}, \overline{lst_i})(a_n, lst_i), (b_1, \overline{lst_i}) \cdots (b_{m-1}, \overline{lst_i})(b_m, lst_i))$. Intuitively, lst_i marks the last symbols of tuples from R_i. It is easy to see that $R_1', R_2' \in \textsc{Rel}(C)$ using closure under componentwise letter-to-letter relations. Observe that $R_1' \cdot R_2' \subseteq (R_1' \cup R_2')^*$ and, by closure under union and Kleene star, that $(R_1' \cup R_2')^* \in \textsc{Rel}(C)$. Let $L \subseteq_{reg} C \otimes (\mathbb{A} \times$

$\{\overline{lst}_1, lst_1, \overline{lst}_2, lst_2\})^*$ such that $[\![L]\!] = (R'_1 \cup R'_2)^*$. It is easy to see that there is $L' \subseteq_{reg} L$ such that $[\![L']\!] = R'_1 \cdot R'_2$, and thus that $R'_1 \cdot R'_2 \in \textsc{Rel}(C)$. Again by closure under component-wise letter-to-letter relations we obtain that $R_1 \cdot R_2 \in \textsc{Rel}(C)$, this time using the relation that projects onto the first component.

Finally, we prove item 4. For the right-to-left direction, let $R \in \textsc{Rel}(C)$ and take $L \subseteq_{reg} C \otimes \mathbb{A}^*$ such that $[\![L]\!] = R$. Therefore $R^* = [\![L]\!]^* = [\![L^*]\!] \in \textsc{Rel}(C^*) \subseteq \textsc{Rel}(C)$ as wanted. For the converse, first observe that $\textsc{Rel}(C)$ is also closed under concatenation due to item 3. Let $R \in \textsc{Rel}(C^*)$. By item 2 of Lemma 4, we have the following:

there are $R_1, \ldots, R_n \in \textsc{Rel}(C)$ *and* $I \subseteq_{reg} \{1, \ldots, n\}^*$ *such that* $R = \bigcup_{w \in I} R_{w[1]} \cdots R_{w[|w|]}$.

Consider any regular expression E denoting the language I above, and replace each occurence of $i \in \{1, \ldots, n\}$ with R_i, in such a way that the resulting expression E' denotes R. Then, by finite application of closure under Kleene star, concatenation and union as given by E', we obtain that $R \in \textsc{Rel}(C)$. ◀

For $C \subseteq_{reg} \Bbbk^*$ and $K \subseteq \Bbbk$, let $C|_K$ be the projection of C onto the alphabet K (which is also regular). We give the following characterization of closure under projection.

▶ **Lemma 28.** *For every* $k \in \mathbb{N}$ *and* $C \subseteq_{reg} \Bbbk^*$, $\textsc{Rel}_k(C)$ *is closed under projection iff* $\textsc{Rel}_k(C|_K) \subseteq \textsc{Rel}_k(C)$ *for every* $K \subseteq \Bbbk$.

Decidability

For the binary case, by previous results [14], it is decidable to test whether a synchronized class is included in another, and thus the characterizations for Kleene star and concatenation are decidable. We leave the general case as an open question.

6 Concluding remarks and future work

We discuss the decidability of paradigmatic problems within $\textsc{Rel}(C)$. First, note that the emptiness problem for relations reduces to the emptiness problem for automata: $[\![L]\!] = \emptyset$ if, and only if, $L = \emptyset$ – and thus the emptiness problem is always decidable. Further, by the results we have shown together with Lemma 6 we obtain the following.

▶ **Lemma 29.** *For* $C \subseteq_{reg} \mathbb{2}^*$, *if* $\textsc{Rel}(C)$ *is closed under...*

- *...intersection, then equivalence and containment problems within* $\textsc{Rel}(C)$ *are decidable;*
- *...complement, then the universality problem within* $\textsc{Rel}(C)$ *is decidable;*
- *...*Op*, then the* Op *operation within* $\textsc{Rel}(C)$ *is computable, for* Op $\in$ {*intersection, complement, concatenation, Kleene star, projection*}.

Proof of Lemma 29. Given $L, M \subseteq_{reg} C \otimes \mathbb{A}^*$, the containment problem between $[\![L]\!]$ and $[\![M]\!]$ amounts to checking if $[\![L]\!] \setminus [\![M]\!]$ is empty. Since $\textsc{Rel}(C)$ is closed under intersection, by Theorem 12, there exists a Parikh-injective language D such that $C \subseteq_{\textsc{Rel}} D$. Moreover, our decidability proof, shows that we can effectively compute such language D. Therefore, by the results on [14], we can effectively construct $L', M' \subseteq_{reg} D \otimes \mathbb{A}^*$ such that $[\![L]\!] = [\![L']\!]$, and $[\![M]\!] = [\![M']\!]$. Then, by Lemma 6, item 3, $[\![L]\!] \setminus [\![M]\!] = [\![L']\!] \setminus [\![M']\!] = [\![L' \setminus M']\!]$ and so the containment problem within $\textsc{Rel}(C)$ reduces to the emptiness problem within $\textsc{Rel}(D)$. The equivalence problem obviously reduces to the containment problem.

The universality problem for $([\![L]\!], \mathbb{A})$ amounts to checking whether $(\mathbb{A}^*)^k \setminus [\![L]\!]$ is empty. Since $\textsc{Rel}(C)$ is closed under complement, by Theorem 24, there exists a Parikh-bijective language D such that $C =_{\textsc{Rel}} D$. As before, we can effectively compute such language D,

and therefore, by the results on [14], we can effectively construct $L' \subseteq_{reg} D \otimes \mathbb{A}^*$ such that $[\![L']\!] = [\![L]\!]$. By Lemma 6, item 4, we thus obtain $(\mathbb{A}^*)^k \setminus [\![L]\!] = (\mathbb{A}^*)^k \setminus [\![L']\!] = [\![(D \otimes \mathbb{A}^*) \setminus L']\!]$ and so the containment problem within $\textsc{Rel}(C)$ reduces to the emptiness problem within $\textsc{Rel}(D)$.

We prove the last item only for intersection; similar (or simpler) arguments can be used for all the other operations. Given $L, M \subseteq_{reg} C \otimes \mathbb{A}^*$, with a similar argument than the one used in the previous item, we can effectively construct a Parikh-injective language D and $L', M' \subseteq_{reg} D \otimes \mathbb{A}^*$ such that $[\![L]\!] = [\![L']\!]$, and $[\![M]\!] = [\![M']\!]$. Then, by Lemma 6, item 3, $[\![L]\!] \cap [\![M]\!] = [\![L']\!] \cap [\![M']\!] = [\![L' \cap M']\!]$ and the result follows. ◀

One can then conclude that classes of synchronized binary relations are generally "well-behaved": *a*) it is decidable to test whether a class is closed under Boolean connectives; *b*) every synchronized class closed under intersection (resp. complement, etc.), is *effectively* closed under intersection (resp. complement, etc.); *c*) every synchronized class which is closed under Boolean connectives has decidable paradigmatic problems (in the sense of Lemma 29); *d*) at least for the binary case, the characterizations for Kleene star and concatenation are decidable.

We leave as future work the question of whether it is decidable to test if $\textsc{Rel}(C)$ is closed under Kleene star, concatenation and projection when $C \subseteq_{reg} \mathbb{k}$. We also leave open the characterization for closure under complement and intersection for k-ary relations. Although it is conceivable that the same characterization for closure under intersection holds for arbitrary arity relations, we were not able to show it – the main issue is that it is not clear how to generalize the bad conditions to a k-ary alphabet, nor what would be the analog of item 6 in Theorem 12.

▶ **Conjecture 30.** *For every $k \in \mathbb{N}$ and $C \subseteq_{reg} \mathbb{k}^*$, $\textsc{Rel}(C)$ is closed under intersection if, and only if, $C \subseteq_{\textsc{Rel}} D$ for some Parikh-injective $D \subseteq_{reg} \mathbb{k}^*$.*

References

1 Parosh Aziz Abdulla, Bengt Jonnson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *International Conference on Concurrency Theory (CONCUR)*, pages 35–48, 2003.

2 Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008. doi:10.1145/1322432.1322433.

3 Kemafor Anyanwu and Amit Sheth. ρ-Queries: enabling querying for semantic associations on the semantic web. In *12th International World Wide Web Conference (WWW'03)*, pages 690–699, 2003.

4 Pablo Barceló, Diego Figueira, and Leonid Libkin. Graph Logics with Rational Relations. *Logical Methods in Computer Science (LMCS)*, 9(3:01), 2013. doi:10.2168/LMCS-9(3:1)2013.

5 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive Languages for Path Queries over Graph-Structured Data. *ACM Trans. Database Syst.*, 37(4):31, 2012. doi:10.1145/2389241.2389250.

6 Pablo Barceló and Pablo Muñoz. Graph Logics with Rational Relations: The Role of Word Combinatorics. *ACM Trans. Comput. Log.*, 18(2):10:1–10:41, 2017. doi:10.1145/3070822.

7 Jean Berstel. *Transductions and Context-Free Languages.* B. G. Teubner, 1979.

8 Achim Blumensath and Erich Grädel. Automatic Structures. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 51–62. IEEE Computer Society Press, 2000. doi:10.1109/LICS.2000.855755.

9 Mikołaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014. doi:10.1007/978-3-662-43951-7.

10 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *International Conference on Computer Aided Verification (CAV)*, pages 403–418, London, UK, 2000. Springer-Verlag.

11 Julius Richard Büchi. Weak Second-order Arithmetic and Finite Automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

12 Olivier Carton. The growth ratio of synchronous rational relations is unique. *Theor. Comput. Sci.*, 376(1-2):52–59, 2007. doi:10.1016/j.tcs.2007.01.012.

13 Christian Choffrut. Relations over Words and Logic: A Chronology. *Bulletin of the EATCS*, 89:159–163, 2006.

14 María Emilia Descotte, Diego Figueira, and Gabriele Puppis. Resynchronizing Classes of Word Relations. In *International Colloquium on Automata, Languages and Programming (ICALP)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 381:1–381:13. Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.381.

15 Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.

16 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *Journal of the ACM*, 62(2):12:1–12:51, 2015. doi:10.1145/2699442.

17 Diego Figueira and Leonid Libkin. Path Logics for Querying Graphs: Combining Expressiveness and Efficiency. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 329–340. IEEE Computer Society Press, 2015. doi:10.1109/LICS.2015.39.

18 Diego Figueira and Leonid Libkin. Synchronizing Relations on Words. *Theory of Computing Systems*, 57(2):287–318, 2015. doi:10.1007/s00224-014-9584-2.

19 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On Equivalence and Uniformisation Problems for Finite Transducers. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 125:1–125:14. Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.125.

20 Christiane Frougny and Jacques Sakarovitch. Synchronized Rational Relations of Finite and Infinite Words. *Theoretical Computer Science*, 108(1):45–82, 1993. doi:10.1016/0304-3975(93)90230-Q.

21 Seymour Ginsburg and Edwin Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

22 Bengt Jonsson and Marcus Nilsson. Transitive Closures of Regular Relations for Verifying Infinite-State Systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 220–234. Springer, 2000.

23 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

24 Maurice Nivat. Transduction des langages de Chomsky. *Annales de l'Institut Fourier*, 18:339–455, 1968.

25 Rohit Parikh. On Context-Free Languages. *Journal of the ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

26 Anthony Widjaja To and Leonid Libkin. Algorithmic Metatheorems for Decidable LTL Model Checking over Infinite Systems. In *International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, pages 221–236, 2010. doi:10.1007/978-3-642-12032-9_16.

27 Sarah Winter. Uniformization Problems for Synchronizations of Automatic Relations on Words. In *International Colloquium on Automata, Languages and Programming (ICALP)*, Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum für Informatik, 2018. URL: http://arxiv.org/abs/1805.02444.

Resource-Bounded Kolmogorov Complexity Provides an Obstacle to Soficness of Multidimensional Shifts

Julien Destombes
LIRMM, University of Montpellier, Montpellier, France
julien.destombes@lirmm.fr

Andrei Romashchenko
LIRMM, University of Montpellier, CNRS, Montpellier, France
andrei.romashchenko@lirmm.fr

Abstract

We suggest necessary conditions of soficness of multidimensional shifts formulated in terms of resource-bounded Kolmogorov complexity. Using this technique we provide examples of effective and non-sofic shifts on $\mathbb{Z}^2$ with very low block complexity: the number of globally admissible patterns of size $n \times n$ grows only as a polynomial in n.

2012 ACM Subject Classification Mathematics of computing → Information theory; Mathematics of computing → Combinatorics

Keywords and phrases Sofic shifts, Block complexity, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.23

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.03929`.

Funding projet ANR-15-CE40-0016 RaCAF

Acknowledgements We are indebted to Bruno Durand, Alexander Shen, and Ilkka Törmä for fruitful discussions. We are grateful to Pierre Guillon and Emmanuel Jeandel for motivating comments. We also thank the anonymous referees of STACS 2019 for many valuable comments.

1 Introduction

Symbolic dynamics originally appeared in mathematics as a branch of the theory of dynamical systems that studies smooth or topological dynamical systems by discretizing the underlying space. Since the late 1930s, symbolic dynamics became an independent field of research, see [9, 10]. A classical *dynamical system* is a space (of states) $\mathcal{S}$ with a function F acting on this space; this function represents the "evolution rule," i.e., the time dependence of a configuration in the space. The central notion of the theory of dynamical systems is a trajectory – a sequence of configurations obtained by iterating the evolution rules,

$$x, F(x), F(F(x)), \dots, F^{(n)}(x), \dots$$

In symbolic dynamics the space of states reduces to a finite set (an *alphabet*). The trajectories are represented by infinite (or bi-infinite) sequences of letters over this alphabet, and the "evolution rule" is the shift operator acting on these sequences. Symbolic dynamics focuses on the *shift spaces* – the sets of bi-infinite sequences of letters (over a finite alphabet) that are defined by a shift-invariant constraint on the factors of finite length. More precisely, a *shift* over an alphabet Σ is a subset of bi-infinite sequences over Σ that is translation invariant and closed in the natural topology of the Cantor space. Every shift can be defined in

terms of *forbidden finite patterns*: we fix a set of (finite) words $\mathcal{F}$ and say that a configuration (a bi-infinite sequence) belongs to the corresponding shift $S_{\mathcal{F}}$ if and only if it does not contain any factor from $\mathcal{F}$.

Obviously, the properties of shifts heavily depend on the corresponding set of forbidden patterns. The following three large classes of shifts play an important role in symbolic dynamics and computability theory:

- *shifts of finite type* (SFT), which are defined by a finite set of forbidden finite patterns;
- *sofic* shifts (introduced in [16]), where the set of forbidden finite patterns is a regular language;
- *effective* (or *effectively closed*) shifts, which are defined by a computable set of forbidden finite patterns.

These three classes are different: [the SFTs] $\subsetneqq$ [the sofic shifts] $\subsetneqq$ [the effective shifts].

The sofic shifts can be equivalently defined as the coordinate-wise projections of configurations from an SFT:

▶ **Definition 1.** *A shift $\mathcal{S}$ over an alphabet Σ is* sofic *if there is an SFT $\mathcal{S}'$ over an alphabet Σ' and a mapping $\pi : \Sigma' \to \Sigma$, such that $\mathcal{S}$ consists of the coordinate-wise projections*

$$(\dots \pi(y_{-1})\pi(y_0)\pi(y_1)\pi(y_2)\dots)$$

of all configurations $(\dots y_{-1}y_0y_1y_2\dots)$ from $\mathcal{S}'$.

There is a simple characterization of soficness. Let us say that two words w_1, w_2 are *equivalent* in a shift $\mathcal{S}$, if exactly the same half-infinite configurations occur in $\mathcal{S}$ immediately to the right of w_1 and to the right of w_2. A shift is sofic if and only if the finite patterns in this shift are subdivided in a finite number of equivalence classes (see [8, Theorem 3.2.10]). Loosely speaking, when we read a configuration from the left to the right and verify that it belongs to a sofic shift, we need to keep in mind only a finite information.

The SFTs and even the sofic shifts are rather restrictive classes of shifts with several very special properties. Not surprisingly, many important examples of effective shifts are not sofic. Non-soficness of a shift is usually proved with some version of the pumping lemma from automata theory.

Multidimensional shifts

The formalism of shifts can be naturally extended to the grids $\mathbb{Z}^d$ for $d > 1$. A shift on $\mathbb{Z}^d$ (over a finite alphabet Σ) is defined as a set of d-dimensional configurations $f : \mathbb{Z}^d \to \Sigma$ that are (i) translation-invariant (under translations in all directions) and (ii) closed in Cantor's topology. Similar to the one-dimensional case, the shifts can be defined in terms of forbidden finite patterns.

The definitions of the *effective shifts* (the set of forbidden patterns is computable) and of the *SFTs* (the set of forbidden patterns is finite) apply to the multidimensional shift spaces directly, without any revision. The *sofic shifts* on $\mathbb{Z}^d$ are defined as in Definition 1 above (as the coordinate-wise projections of SFTs).

For multidimensional shifts spaces, the classes of the effective shifts, the sofic shifts, and the SFTs remain distinct, though the difference between these classes is more elusive than in the one-dimensional case. In this paper we discuss the tools that help to reveal the reasons why one or another effective multidimensional shift is *not* sofic.

The class of sofic shifts in dimension $d \geq 2$ is surprisingly wide. Besides many simple and natural examples, there are shifts whose soficness follow from rather subtle considerations. For instance, S. Mozes showed that the shift generated by (a natural class of) non deterministic

multidimensional substitutions systems are sofic [11]. L. B. Westrick proved that the two-dimensional shift on the alphabet $\{0, 1\}$ whose configurations consist of squares of 1s of pairwise different sizes on a background of 0s, is sofic; moreover, any effectively closed subshift of this shift is also sofic [17].

On the other hand, there are several examples of effective multidimensional shifts that are known to be non-sofic. In what follows we briefly discuss two of them.

▶ **Example 2** (the mirror shift)**.** One of the standard examples of a non-sofic shift is the shift of mirror-symmetric configurations on $\mathbb{Z}^2$. Let Σ be the alphabet with three letters (e.g., *black*, *white*, and *red*), and the configurations of the shift are all black-and-white configurations (without any red cell) and the configurations with an infinite horizontal line of red cells and symmetric black-and-white half-planes above and below this line, see Fig. 1.

It is easy to see that this shift is effective (the forbidden patterns are those where the red cells are not aligned, and those where the areas above and below the horizontal red line are not symmetric). At the same time, this shift is not sofic. The intuitive explanation of this fact is as follows. Let us focus on a pair of symmetric patterns of size $n \times n$ in black and white, above and below the horizontal red line (see the blue squares in Fig. 1). To make sure that the configuration belongs to the shift, we must "compare" these two patterns with each other. To this end, we need to transmit the information about a pattern of size n^2 through its border line (of length $O(n)$). However, in a sofic shift, the "information flow" across a contour of length $O(n)$ is bounded by $O(n)$ bits, and this contradiction implies non-soficness. For a more formal argument see, e.g. [1] and [4], or a similar example [5, Example 2.4].

▶ **Example 3** (the high complexity shift)**.** Let S be the set of all binary configurations on $\mathbb{Z}^2$ where for each $n \times n$ pattern P its Kolmogorov complexity is quadratic, $C(P) = \Omega(n^2)$. Technically, this means that no globally admissible pattern can be produced by a program of size below cn^2, for some factor $c > 0$ (see the formal definition of Kolmogorov complexity below).

This shift is obviously effective: we can algorithmically enumerate the patterns whose Kolmogorov complexity is below the specified threshold. However, this shift is not sofic. This follows from two facts (proven in [2]):

- **(i)** For some $c < 1$, the shift defined above is not empty.
- **(ii)** In every non-empty sofic shift on $\mathbb{Z}^2$, there is a configuration where the Kolmogorov complexity of each $n \times n$ pattern is bounded by $O(n)$.

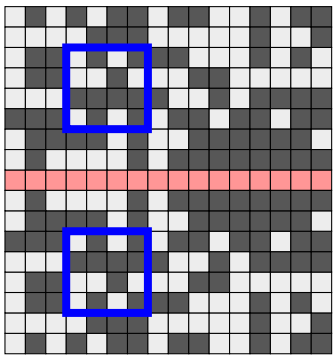


Figure 1 A configuration with mirror symmetry with respect to the horizontal red line. The blue squares select two symmetric black-and-white patterns.

Note that the non-sofic shifts in the two examples above have positive entropy (the number of globally admissible patterns of size $n \times n$ grows as $2^{\Omega(n^2)}$). This is not surprising: the proofs of non-soficness of these shifts use the intuition about the information flows (*super-linear amount of information cannot flow through a linear contour*). This type of argument can be adapted for several shifts where the number of globally admissible patterns of size $n \times n$ grows slower than $2^{\Omega(n^2)}$ but still faster than $2^{O(n)}$ (see, e.g. [15, Proposition 15]). As it was noticed in [17], "*all examples known to the author of effectively closed shifts which are not sofic were obtained by in some sense allowing elements to pack too much important information into a small area.*"

This type of argument was formalized as rather general sufficient conditions of non-soficness in [13] and [5]. The theorems by Kass and Madden ([8, Theorem 3.2.10]) and Pavlov ([13, Theorem 1.1]) apply only to the two-dimensional shifts where the number of globally admissible $n \times n$ patterns is greater than $2^{O(n)}$. However, there is no reason to think that this condition is necessary for non-soficness (see, e.g. the discussion in [4, Section 1.2.2]). It is instructive to observe that non-effective non-sofic shifts can have very low block complexity [5, 12].

In this paper we extend the usual approach to the proof of non-soficness. We show that a shift cannot be sofic if the essential information contained in an $n \times n$ pattern cannot be compressed to the size $O(n)$ in *bounded time*.

The intuition behind our argument is similar to those used in [2] and [5] but with the idea of compression with *bounded computational resources*. This approach applies to several shifts with very low block complexity: we cannot "communicate" the essential information across a contour not because this information is too large, but since we do not have enough time and space to compress it. In particular, we provide examples of non-sofic effective shifts with only polynomial block complexity (and thus zero entropy).

▶ Remark 4. A standard and straightforward approach to the measure of the "*information flow through the border line of a pattern*" uses the notion of *extender*. Let $\mathcal{S}$ be a shift and P be a globally admissible pattern for this shift. The *extender* of P in $\mathcal{S}$ is the set of all configurations Q completing P to a valid configuration of $\mathcal{S}$ (in particular, the support of Q should be the complement of the support of P).

Let $\mathcal{S}$ be a shift on $\mathbb{Z}^2$; denote by N_k the number of different extenders for the patterns with a support of size $k \times k$. (Several patterns can share one and the same extender, so the number of extenders might be much less than the number of globally admissible patterns of this size). It seems natural to interpret $\log N_k$ as "*the information flowing going through the border line*" of a $k \times k$ pattern.

However, this interpretation is deceptive. In a sofic shift the value of $\log N_k$ can grow much faster the length of the border line of the pattern ([5] attributes this observation to unpublished works of C. Hoffman, A. Quas, and R. Pavlov). In fact, for a sofic shift on $\mathbb{Z}^2$, the value of $\log N_k$ can grow even as $\Omega(k^2)$. Therefore, we cannot use the asymptotic of $\log N_k$ to prove non-soficness of a multi-dimensional shift. This why we need a subtler implementation of the intuition of "information flows" in the sofic shifts.

The rest of the paper is organized as follows. After recalling the main definitions of the theory of Kolmogorov in the second section, we prove in the third one our main result. In the last section we elaborate our technique to a more general setting; in particular, we show that an argument from [5] (a proof of non-soficness with the method of union-increasing sequences of extenders) can be explained in the language of Kolmogorov complexity.

2 Preliminaries

Kolmogorov complexity

In this section we recall the main definitions of the theory of Kolmogorov complexity. Let U be a (partial) computable function. The complexity of x with respect to the description method U is defined as $\mathrm{C}_U(x) := \min\{\ |p|\ :\ U(p) = x\ \}$.

If there is no p such that $U(p) = x$, we assume that $\mathrm{C}_U(x) = \infty$. Here U is understood as a programming language; p is a program that prints x; the complexity of x is the length of (one of) the shortest programs p that generate x (on the empty input).

The obvious problem with this definition is its dependence on U. The theory of Kolmogorov complexity becomes possible due to the invariance theorem:

▶ **Theorem 5** (Kolmogorov [6]). *There exists a computable function U such that for any other computable function V there is a constant c such that $\mathrm{C}_U(x) \le \mathrm{C}_V(x) + c$ for all x.*

This U is called *an optimal description method*. We fix an optimal U and in what follows omit the subscript in $\mathrm{C}_U(x)$. The value $\mathrm{C}(x)$ is called the (plain) Kolmogorov complexity of x.

In a similar way, we define Kolmogorov complexity in terms of programs with bounded resources (the time of computation). Let U be a Turing machine; we define the Kolmogorov complexity $\mathrm{C}^t_U(x)$ as the length of the shortest p such that $U(p)$ produces x in at most t steps. There exists an *optimal description method* U in the following sense: for every Turing machine V we have $\mathrm{C}^{\mathrm{poly}(t)}_U(x) \le \mathrm{C}^t_V(x) + O(1)$.

For multi-tape Turing machines a slightly stronger statement can be proven:

▶ **Theorem 6** (see [7]; the proof uses the simulation technique from [3]). *There exists an* optimal description method (*multi-tape Turing machine*) *U in the following sense: for every multi-tape Turing machine V there exists a constant c such that $\mathrm{C}^{ct\log t}_U(x) \le \mathrm{C}^t_V(x) + c$ for all strings x.*

We fix such a machine U, and in the sequel use for the resource-bounded version of Kolmogorov complexity the notation $\mathrm{C}^t(x)$ instead of $\mathrm{C}^t_U(x)$. Without loss of generality we may assume that $\mathrm{C}(x) \le \mathrm{C}^t(x)$ for all x and for all t.

We fix a computable enumeration of finite patterns (over a finite alphabet) that assigns a binary string (a *code*) to each pattern in dimension two. In the sequel we take the liberty of talking about Kolmogorov complexity of finite patterns in dimension two (assuming the Kolmogorov complexity of the *codes* of these patterns).

Shift spaces

In this paper we focus on two-dimensional shifts, though all arguments can be extended to the shifts on $\mathbb{Z}^d$ for all $d \ge 2$. A (finite) *pattern* on $\mathbb{Z}^2$ over a finite alphabet Σ is a mapping from a (finite) subset of $\mathbb{Z}^2$ to Σ; the domain of this mapping is the *support* of the pattern. Sometimes a pattern P with a support $\mathcal{A}$ is called a *coloring* of $\mathcal{A}$ (the "colors" are letter from Σ).

For a shift S, we say that a pattern P is *globally admissible*, if P is a restriction of a configuration from S to some finite support. For a shift of finite type determined by a set of forbidden patterns $\mathcal{F}$, we say that a pattern is *locally admissible* if it contains no forbidden patterns from $\mathcal{F}$.

The *block complexity* of a shift is a function that gives for each integer $n > 0$ the number of globally admissible patterns of size $n \times n$ (patterns with support $\{1, \dots, n\}^2$) in this shift.

If a sofic shift S is a coordinate-wise projection of configurations from $\hat{S}$, we say that $\hat{S}$ is a covering of S. Every sofic shift has a covering SFT such that the supports of all forbidden patterns in this SFT are pairs of neighboring cells (see, e.g. [8]).

3 High resource-bounded Kolmogorov complexity is compatible with low block complexity

The following theorem was proven implicitly in [2]:

▶ **Theorem 7.** *In every non-empty sofic shift S there exists a configuration x such that for all $n \times n$-patterns P in x, we have $\mathrm{C}^{T(n)}(P) = O(n)$ for a time threshold $T(n) = 2^{O(n^2)}$.*

In [2] a weaker version of this theorem is stated: it is claimed only that the *plain* complexity of $n \times n$ patterns is $O(n)$. However, the argument from [2] implies a bound for a *resource-bounded* version of Kolmogorov complexity. For the sake of self-containedness, we provide a proof of this theorem in the full version of this paper.

▶ **Theorem 8.** *For every $\epsilon > 0$ and for every computable $T(n)$ there exists an effective shift on $\mathbb{Z}^2$ such that for every n and for every globally admissible pattern P of size $n \times n$, we have that*

(i) $\mathrm{C}(P) = O(\log n)$, *and*
(ii) $\mathrm{C}^{T(n)}(P) = \Omega(n^{2-\epsilon})$.

Theorem 8 is proven is the full version of the paper. In what follows we prove a slightly weaker version of this theorem, which is nevertheless strong enough for our main applications:

▶ **Theorem 8′.** *For every computable $T(n)$ there exists an effective shift on $\mathbb{Z}^2$ such that*

(i) *for every n and for every globally admissible pattern P of size $n \times n$, we have $\mathrm{C}(P) = O(\log n)$, and*
(ii) *for infinitely many n and for every globally admissible pattern P of size $n \times n$, we have that $\mathrm{C}^{T(n)}(P) = \Omega(n^{1.5})$.*

From Theorem 7 and Theorem 8′ we deduce the following corollary:

▶ **Corollary 9.** *There exists an effective non-sofic shift on $\mathbb{Z}^2$ with block complexity $\mathrm{poly}(n)$, i.e., with $\leq \mathrm{poly}(n)$ globally admissible blocks of size $n \times n$.*

Proof. We take the shift from Theorem 8′ assuming that the threshold $T(n)$ is much greater than $2^{\Omega(n^2)}$ (e.g., we can let $T(n) = 2^{n^3}$). On the one hand, property (ii) of Theorem 8′ and Theorem 7 guarantee that this shift is not sofic. On the other hand, property (i) of Theorem 8′ implies that the number of globally admissible blocks of size $n \times n$ is not greater than $2^{O(\log n)}$. ◀

▶ Remark 10. Our proof of Theorem 8 implies a stronger bound than property (i). In fact, instead of the bound $\mathrm{C}(P) = O(\log n)$ we can prove that for every globally admissible $n \times n$ pattern P in this shift,

$$\mathrm{C}^{\hat{T}(n)}(P) \leq \lambda \log n, \tag{1}$$

where λ is a (large enough) constant and $\hat{T}(n)$ is a (large enough) computable function of n. The constant λ and the threshold $\hat{T}(n)$ can be defined quite explicitly given $T(n)$ and ϵ.

When $\hat{T}(n)$ (compatible with given ϵ and $T(n)$) is chosen, we can define another shift $\mathcal{S}_{T,\epsilon}$ that consists of the configurations where all $n \times n$ patterns P satisfy (1). The shift from

Theorem 8 is a proper subshift of $\mathcal{S}_{T,\epsilon}$. Besides all configurations from Theorem 8, the shift $\mathcal{S}_{T,\epsilon}$ contains also configurations with patterns of very low time bounded complexity (e.g., the configuration with all 0s and the configuration with all 1s). In the next section we use this shift $\mathcal{S}_{T,\epsilon}$ to construct some other examples of effective non-sofic shifts.

Proof of Theorem 8′. In this proof we construct the required shift explicitly. Let us fix a sequence (n_i) where n_0 is a large enough integer number, and

$$n_{i+1} := (n_0 \cdot \ldots \cdot n_i)^c \quad \text{for } i = 0, 1, 2, \ldots, \tag{2}$$

where $c \geq 3$ is a constant. We set $N_i := n_0 \cdot \ldots \cdot n_i$. In what follows we construct for each i a pair of *standard* binary patterns Q_i^0 and Q_i^1 of size $N_i \times N_i$ such that

- the plain Kolmogorov complexities of the standard patterns $\mathrm{C}(Q_i^0)$ and $\mathrm{C}(Q_i^1)$ are not greater than $O(\log N_i)$, and
- the resource-bounded Kolmogorov complexities $\mathrm{C}^{T(N_i)}(Q_i^0)$ and $\mathrm{C}^{T(N_i)}(Q_i^1)$ are not less than $\Omega(N_i^{1.5})$.

The construction is hierarchical: both Q_i^0 and Q_i^1 are defined as $n_i \times n_i$ matrices composed of patterns Q_{i-1}^0 and Q_{i-1}^1; for each i the blocks Q_i^0 and Q_i^1 are bitwise inversions of each other.

When the standard patterns Q_i^0 and Q_i^1 are constructed for all i, we define the shift as the closure of these patterns: we say that a finite pattern is globally admissible if and only if it appears in some standard pattern Q_i^j or at least in a 2×2-block composed of Q_i^0 and Q_i^1 (for some i).

▶ Remark 11. Due to the hierarchical structure of the standard patterns, we can guarantee that every globally admissible pattern P of size $N_i \times N_i$ appears in a 2×2-block composed of Q_i^0 and Q_i^1 (no need to try the blocks Q_s^j for $s > i$).

Since the construction of Q_i^j is explicit, the resulting shift is effective. Properties (i) and (ii) of the theorem will follow from the properties of the standard patterns.

In what follows we explain an inductive construction of Q_i^0 and Q_i^1. Let Q_0^0 and Q_1^0 be the squares composed of only 0s and only 1s respectively. Further, for every i we take the lexicographically first binary matrix R_i of size $n_i \times n_i$ such that

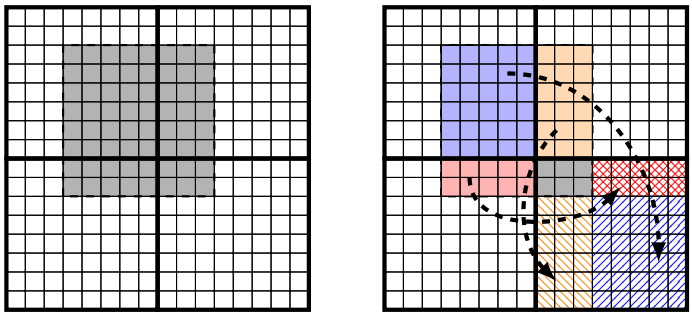
$$\mathrm{C}^{t_i}(R_i) \geq n_i^2 \tag{3}$$

(the time bound t_i is fixed in the sequel). We claim that such a matrix exists. Indeed, there exists a matrix of size $n_i \times n_i$ that is incompressible in the sense of the plain Kolmogorov complexity. The resource-bounded Kolmogorov complexity of a matrix can be only greater than the plain complexity. Therefore, there exists at least one matrix satisfying (3). If t_i is a computable function of i, then given i we can find R_i algorithmically.

Now we substitute in R_i instead of each zero and one entry the copies of Q_{i-1}^0 and Q_{i-1}^1 respectively, e.g.,

$$R_i = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \implies Q_i^0 := \left(\begin{array}{c|c|c|c|c} Q_{i-1}^0 & Q_{i-1}^0 & Q_{i-1}^0 & Q_{i-1}^0 & Q_{i-1}^1 \\ \hline Q_{i-1}^0 & Q_{i-1}^1 & Q_{i-1}^0 & Q_{i-1}^0 & Q_{i-1}^1 \\ \hline Q_{i-1}^1 & Q_{i-1}^1 & Q_{i-1}^1 & Q_{i-1}^1 & Q_{i-1}^0 \\ \hline Q_{i-1}^0 & Q_{i-1}^1 & Q_{i-1}^1 & Q_{i-1}^0 & Q_{i-1}^0 \\ \hline Q_{i-1}^0 & Q_{i-1}^1 & Q_{i-1}^0 & Q_{i-1}^1 & Q_{i-1}^0 \end{array}\right)$$

The resulting matrix (of size $N_i \times N_i$) is denoted Q_i^0. Matrix Q_i^1 is defined as the bitwise inversion of Q_i^0.



(a) (b)

Figure 2 A pattern of size $N_k \times N_k$ (shown in gray in fig. (a)) covered by a quadruple of standard blocks of the same size contains enough information to reconstruct a standard pattern (fig. (b)).

▷ Claim 12. Assuming that $t'_i \ll t_i$ (in what follows we discuss the choice of t'_i in more detail) we have

$$\mathrm{C}^{t'_i}(Q_i^0) = \Omega(N_i^{1.5}) \text{ and } \mathrm{C}^{t'_i}(Q_i^1) = \Omega(N_i^{1.5}).$$

Proof of Claim 12. Given Q_i^j (for $j = 0, 1$) we can retrieve the matrix R_i (this retrieval can be implemented in polynomial time). Therefore, for every time bound t

$$\mathrm{C}^{t+\mathrm{poly}(N_i)}(R_i) \le \mathrm{C}^t(Q_i^j) + O(1).$$

Therefore, if $t_i > t'_i + \mathrm{poly}(N_i)$ then

$$n_i^2 \le \mathrm{C}^{t_i}(R_i) \le \mathrm{C}^{t'_i}(Q_i^j).$$

It remains to observe that our choice of parameters in (2) with $c \ge 3$ implies $n_i^{1/2} \ge (n_0 \cdot \ldots \cdot n_{i-1})^{3/2}$, and therefore

$$n_i^2 \ge (n_0 \cdot \ldots \cdot n_i)^{1.5} = (N_i)^{1.5}.$$

Thus, we obtain $\mathrm{C}^{t'_i}(Q_i^j) \ge (N_i)^{1.5} - O(1)$, and the claim is proven. ◁

▶ Remark 13. By choosing a larger constant c in (2), we can achieve a lower bound $\mathrm{C}^{t'_i}(Q_i^j) = \Omega(n^{2-\epsilon})$ for any $\epsilon > 0$.

▷ Claim 14. For every globally admissible pattern P of size $N_i \times N_i$ (and not only for the standard patterns, as it was in Claim 1) its time-bounded Kolmogorov complexity $\mathrm{C}^{T(N_i)}(P)$ is $\Omega(n^{1.5})$ (assuming that $T(N_i) \ll t'_i$).

Proof of Claim 14. If a pattern P of size $N_i \times N_i$ is globally admissible then it is covered by a quadruple of standard patterns of rank i, see Remark 11 on p 7 above. Then P can be divided into four rectangles which are "corners" of standard patterns of rank i, see Fig. 2 (a). Since the standard blocks Q_i^0 and Q_i^1 are the inversions of each other, these four "corners" (with a bitwise inversion if necessary) form together the entire standard pattern, as shown in Fig. 2 (b). Therefore, we can reconstruct Q_i^j from P given (a) the position of P with respect to the grid of standard blocks (this involves $O(\log N_i)$ bits) and (b) the four bits identifying the standard blocks covering P (we need to know which of them is a copy of Q_i^0 and which one is a copy of Q_i^1).

The retrieval of Q_i^j from P requires only $\mathrm{poly}(N_i)$ steps of computation (in addition to the time we need to produce P). Now the claim follows from the bound for the resource-bounded Kolmogorov complexity of the standard patterns Q_i^0 and Q_i^1. ◁

▷ Claim 15. For every $k \times k$-pattern in Q_i^0 or Q_i^1, its plain Kolmogorov complexity is at most $O(\log k)$.

Proof of Claim 15. First of all, we observe that the standard patterns Q_i^0 or Q_i^1 can be computed given i. Therefore, $\mathrm{C}(Q_i^0) = O(\log i)$ and $\mathrm{C}(Q_i^1) = O(\log i)$.

Every globally admissible $k \times k$-pattern is covered by at most four standard patterns Q_i^0 or Q_i^1 with

$$N_{i-1} < k \le N_i,$$

see Remark 11 on p. 7. Therefore, to obtain a globally admissible pattern P of size $k \times k$ we need to produce a quadruple of standard patterns of size $N_i \times N_i$ and then to specify the position of P with respect to the grid of standard blocks. This description consists of only $O(\log N_i)$ bits, and we conclude that $\mathrm{C}(P) = O(\log k)$. ◁

Given a computable threshold $T(N_i)$, we choose a suitable $t_i' \gg T(N_i)$ and then a suitable $t_i \gg t_i'$. The theorem follows from Claim 14 and Claim 15. ◀

▶ Remark 16. For all large enough i, the incompressible pattern R_i constructed in the proof of Theorem 8′ contains copies of all 2^4 binary patterns of size 2×2. Therefore, we can guarantee that every standard block Q_i^j contains all globally admissible patterns of size $N_{i-1} \times N_{i-1}$. It follows that the shift constructed in Theorem 8′ is transitive and even minimal.

There exists a non-empty effective shift on $\mathbb{Z}^2$ where the Kolmogorov complexity of all $n \times n$ patterns is $\Omega(n^2)$ (see [2] and [14]). So a natural question arises: can we improve Theorem 8 and strengthen condition (ii) to $\mathrm{C}^{T(n)}(P) = \Omega(n^2)$? The answer is negative: we cannot achieve the resource bounded complexity $\Omega(n^2)$, even with a much weaker version of property (i) for the plain complexity:

▶ **Proposition 17.** *For all large enough time bounds $T(n)$, there is no shift on $\mathbb{Z}^2$ such that*

(i) *for every globally admissible pattern P of size $n \times n$, we have that $\mathrm{C}(P) = o(n^2)$, and*

(ii) *for infinitely many n and for every globally admissible pattern P of size $n \times n$, we have that $\mathrm{C}^{T(n)}(P) = \Omega(n^2)$.*

Proof. Assume for the sake of contradiction that such a shift exists. For every k, the number of globally admissible $k \times k$ patterns in this shift is not greater than

$$L_k \le 2^{o(k^2)} \ll 2^{k^2}.$$

Therefore, for any N, every globally admissible pattern P of size $(Nk) \times (Nk)$ can be specified by

- the list of all globally admissible patterns of size $k \times k$ (which requires $L_k \cdot k^2$ bits),
- by an array of $N \times N$ indices of $k \times k$ blocks that constitute P (which requires $N^2 \cdot \log L_k$ bits).

Clearly, P can be reconstructed from such a description in polynomial time. It follows that

$$\mathrm{C}^{\mathrm{poly}(Nk)}(P) \le 2^{o(k^2)} \cdot k^2 + N^2 \cdot o(k^2).$$

For $N \gg 2^{o(k^2)}$ this bound contradicts the condition $\mathrm{C}^{T(Nk)}(P) = \Omega\big((Nk)^2\big)$. ◀

4 Epitomes

The technique from Section 3 does not apply to the shifts that contain very simple configurations (with low resource-bounded Kolmogorov complexity of all patterns). In particular, it does not apply to Example 2 from Introduction. In this section we propose a different technique (also based on resource-bounded Kolmogorov complexity) that helps to handle these examples. The intuitive idea behind this technique is as follows: we try to capture the "essential" information in each pattern (discarding irrelevant data) and then measure the resource-bounded Kolmogorov complexity of an "epitome" of this essential information.

Let us fix some notation. We denote by B_n the set $\{0,\dots,n-1\}^2 \subset \mathbb{Z}^2$ and by F_n its complement, $F_n := \mathbb{Z}^2 \setminus B_n$. We say that two patterns with disjoint supports are *compatible* (for a shift $\mathcal{S}$) if the union of these patterns is globally admissible in $\mathcal{S}$. In particular, a finite pattern P with support B_n and an infinite pattern R with support F_n are compatible, if the union of these patterns is a valid configuration of the shift.

4.1 Plain epitomes

▶ **Definition 18.** *We say that a family of functions*

$$\mathcal{E}_n \ : \ [\text{pattern of size } n\times n] \mapsto [\text{binary string}]$$

is a family of epitomes *for a shift S, if for every globally admissible pattern P with support B_n there exists a pattern R on F_n compatible with P such that for all patterns P' with support B_n compatible with R, we have*

$$\mathcal{E}_n(P') = \mathcal{E}_n(P)$$

(i.e., the pattern R on the complement of B_n determines the $\mathcal{E}_n$-epitome of the pattern on B_n). We say that a family of epitomes is uniformly computable *if there is an algorithm (one algorithm for all n) that computes the mappings $\mathcal{E}_n$. If, in addition, $\mathcal{E}_n$ are computable in time $2^{O(n^2)}$, we say that this family of epitomes is* exp-time computable.

▶ **Proposition 19.** *For every sofic shift with an exp-time computable family of epitomes $\mathcal{E}_n$, for every globally admissible pattern P of size $n\times n$, we have $\mathrm{C}^{T(n)}(\mathcal{E}_n(P)) = O(n)$ for a time threshold $T(n) = 2^{O(n^2)}$.*

▶ Remark 20. If patterns $P_1,\dots,P_m$ with support B_n have pairwise distinct epitomes, then these patterns have a *union-increasing sequence of extenders* in the sense of [5]. Thus, a version of Proposition 19 with the plain (non time bounded) Kolmogorov complexity is a special case of [5, Theorem 2.3].

Proof. Assume S is a sofic shift with a covering SFT $\hat{S}$ (S is a coordinate-wise π-projection of $\hat{S}$). Let P be a pattern with support B_n in S and R be the pattern on the complement of B_n that enforces the value of the $\mathcal{E}_n$-epitome of P (as specified in Definition 18). Denote by y a configuration in $\hat{S}$ whose π-projection gives the union of P and R. Let Q be a pattern of size $n\times n$ in y such that P is a coordinate-wise projection of Q, see Fig. 3. Denote by ∂Q the border of Q.

We assume that the local constraints in $\hat{S}$ involve only pairs of neighboring nodes in $\mathbb{Z}^2$. Then, every locally admissible pattern Q' of size $n\times n$ that is compatible with the border ∂Q, must be compatible with the rest of configuration y. Therefore, the π-projections of these Q' are compatible with R. Thus, the $\mathcal{E}_n$-epitomes of the projections of these Q' must be equal to the $\mathcal{E}_n$-epitome of P.

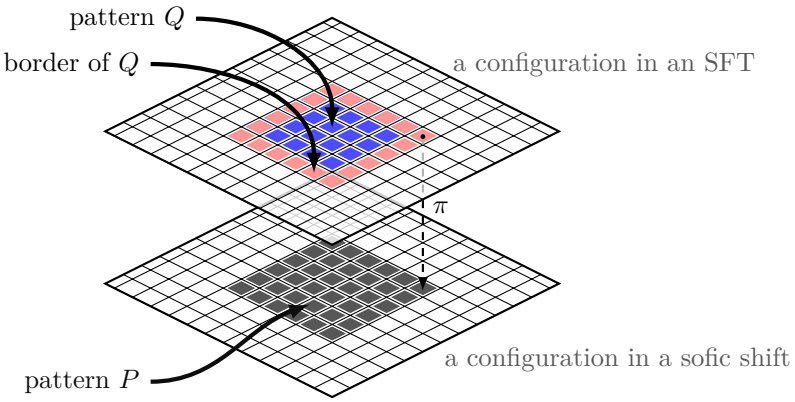


Figure 3 Projection of an $n \times n$ pattern from an SFT onto a sofic shift.

It follows that $\mathcal{E}_n(P)$ can be computed in time $2^{O(n^2)}$ given only the coloring of the border line ∂Q: we use the brute-force search to find *one* Q' computable with this border, apply projection π, and then compute the epitome. Observe that the computed projection $\pi(Q')$ may be different from P, but the epitome must coincide with the epitome of P. Since the size of ∂Q is linear in n, we conclude that $\mathrm{C}^{2^{O(n^2)}}(P) = O(n)$. ◀

Proposition 19 gives a necessary condition for soficness. To prove that a shift is not sofic, we need to provide an exp-time computable family of epitomes with high resource-bounded Kolmogorov complexities. In what follows we discuss a simple application of this technique.

Example 2 revisited

Let $\mathcal{S}$ be the shift from Example 2 in the Introduction (the mirror-symmetric configurations). For this example we can define epitome functions $\mathcal{E}_n$ as follows:

- if an $n \times n$ pattern P contains only black and white letters, then $\mathcal{E}_n(P)$ maps it to a binary string of length n^2 that identifies P uniquely (roughly speaking, $\mathcal{E}_n$ does not compress the patterns in black and white);
- all patterns with red letters are mapped to the empty string.

It is not hard to see that $\mathcal{E}_n$ is an exp-time computable family of epitomes for this shift (since a configuration below the red line determines all black-and-white patterns above this line). Since for some patterns of size $n \times n$ we have $\mathrm{C}(P) \geq n^2$ (i.e., even the plain Kolmogorov complexity of P is super-linear), we can apply Proposition 19 and conclude that the shift is not sofic.

Example 2 with low plain Kolmogorov complexity

Let us consider a subshift of $\mathcal{S}$: we still admit only symmetric configurations, but we now allow only those $n \times n$ patterns P in black and white that are globally admissible for the shift $\mathcal{S}_{T,\epsilon}$ defined in Remark 10 on p. 6, assuming $T(n) = 2^{n^3}$. (We have chosen the time threshold so that $T(n) \gg 2^{O(n^2)}$.) A typical configuration of this shift looks as follows: there is an infinite horizontal line in red, and the symmetric half-planes above and below this line are areas in black and white, with $n \times n$ patterns P such that $\mathrm{C}^{\hat{T}(n)}(P) = O(\log n)$.

The new shift is effective, and the number of globally admissible patterns is $2^{O(\log n)} = \mathrm{poly}(n)$. Due to Theorem 8 know that *some* $n \times n$ patterns in this shift satisfy the condition $\mathrm{C}^{2^{n^3}}(P) = \Omega(n^{2-\epsilon})$.

We cannot apply Theorem 8 directly and conclude that the new shift is non-sofic. Indeed, this shift also admits patterns with very low time-bounded complexity. For example, the shift admits the configuration with an infinite horizontal line in red and only white cells above and below this line.

Note that the functions $\mathcal{E}_n$ defined above provide for this shift an exp-time computable family of epitomes. Since for some (though not for all) $n \times n$ patterns P we have

$$\mathrm{C}^{2^{n^3}}(\mathcal{E}_n(P)) = \Omega(n^{2-\epsilon}),$$

it follows from Proposition 19 that the shift is not sofic.

4.2 Ordered epitomes

The argument based on Definition 18 does not apply to [5, Example 2.5] and similar examples. To handle this class of (non-sofic) shifts we introduce a slightly more general version for epitomes:

▶ **Definition 21.** *Let E_n be a finite set with a partial order $\leq_n$ on it, and*

$$\mathcal{E}_n \ : \ [\text{pattern of size } n \times n] \mapsto [\text{element of } E_n]$$

be a partial *function, for each integer $n > 0$. We say that $(\mathcal{E}_n, \leq_n)$ is a family of* ordered epitomes *for a shift S, if for every globally admissible pattern P with support B_n such that $\mathcal{E}_n(P)$ is defined, there exists a pattern R on F_n such that*

- **(i)** *R is compatible with P, i.e., the union of P and R forms a valid configuration in $\mathcal{S}$, and*
- **(ii)** *for every pattern P' on support B_n compatible with R, if $\mathcal{E}_n(P')$ is defined then*

$$\mathcal{E}_n(P') \leq_n \mathcal{E}_n(P)$$

(i.e., this configuration R on the complement of B_n determines the maximum of the $\mathcal{E}_n$-epitomes over all valid P').

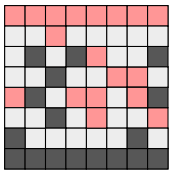
We say that a family of ordered epitomes is uniformly computable *if there is an algorithm (one algorithm for all n) that computes the relations $\leq_n$ and the mappings $\mathcal{E}_n$. If, moreover, $\mathcal{E}_n$ and $\leq_n$ are computable in time $2^{O(n^2)}$, we say that this family of ordered epitomes is* exp-time computable*.*

▶ Remark 22. When we say that a partial function is *computable* (or *computable in bounded time*), we assume that its domain is decidable (respectively, decidable in bounded time). Thus, for an exp-time computable family of epitomes we can decide effectively whether $\mathcal{E}_n(x)$ is defined.

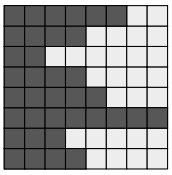
Definition 18 can be viewed as a special case of Definition 21. If $\mathcal{E}_n$ is a family of exp-time computable epitomes in the sense of Definition 18 and $\leq_n$ is an arbitrary effectively computable order on the $\mathcal{E}_n$-epitomes, then $(\mathcal{E}_n, \leq_n)$ is an exp-time computable family of ordered epitomes in the sense of Definition 21 (in Definition 18, the neighborhood R enforces the exact value of $\mathcal{E}_n(P')$ over all P' compatible with R, while in Definition 21 we need to enforce only the maximum of $\mathcal{E}_n(P')$).

▶ **Proposition 23.** *For every sofic shift with an exp-time computable ordered family of epitomes $(\mathcal{E}_n, \leq_n)$, for every globally admissible pattern P of size $n \times n$, $\mathrm{C}^{T(n)}(\mathcal{E}_n(P)) = O(n)$ for a time threshold $T(n) = 2^{O(n^2)}$.*

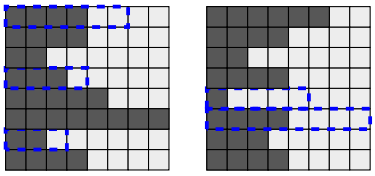
Proof. The proof is similar to the proof of Proposition 19, except for the last part. In the previous proof, we use brute-force search to find *one* pattern Q' compatible with the given border line ∂Q, apply projection π, and then compute the epitome. Now we should find *all* patterns Q' compatible with ∂Q, apply to each of them the projection π, try to compute their epitomes ($\mathcal{E}_n$ is partial), and then take the maximum of the obtained results. It remains to notice that for an exp-time computable ordered family of epitomes this exhaustive search runs in time $2^{O(n^2)}$. ◀



(a) Forbidden pattern: a square with a red top and a black bottom.



(b) A pattern for which the epitome $\mathcal{E}_n$ is defined: each row starts with a few black cells on the left followed by white cells on the right.



(c) A pair of incomparable patterns.

Figure 4

▶ **Example 24** (the shift with no hidden red-black squares)**.** Now we discuss an example proposed by Kass and Madden in [5, Example 2.5], and reformulate the argument given in [5] in the language of Kolmogorov complexity, in terms of ordered epitomes.

Let Σ be the alphabet with three letters (e.g., *black*, *white*, and *red*), and the forbidden patterns be all squares (of all sizes) where the top side consists of red cells, and the bottom one consists of black cells (*hidden red-black squares*), as shown in Fig. 4a.

▶ **Proposition 25** ([5])**.** *The shift on $\mathbb{Z}^2$ defined by the set of forbidden patterns specified above is not sofic.*

In [5] this proposition was proven with the technique of *union-increasing sequence of extenders*. In what follows we propose a similar argument, but explain it in terms of ordered epitomes.

Proof of Proposition 25. We define for this shift a family of ordered epitomes. First of all, we define a class of *simple patterns*: the simple patterns are all square patterns that (i) consist of only black and white letters (with no red letters), where (ii) every row starts with a few successive black letters followed by a sequence of white letters, as show in Fig. 4b. Every simple pattern of size $n \times n$ can be specified by its *profile* – a tuple of integers $(k_1, \dots, k_n)$, where k_i is the number of black cells in the i-th row of the pattern. (Thus, a simple pattern with the profile $(k_1, \dots, k_n)$ is an $n \times n$ square where each i-th row starts with k_i black letters followed by $(n - k_i)$ white letters.)

Let epitome $\mathcal{E}_n$ assign to each simple pattern its profile, and be undefined for all other patterns. For example, for the pattern P show in Fig. 4b we have $\mathcal{E}_8(P) = (4, 3, 8, 5, 4, 2, 4, 6)$.

We introduce the natural order $\leq_n$ on the profiles of simple patterns of size $n \times n$; we say that the profile of P_1 is *not greater* than the profile of P_2, if the first profile is coordinate-wise not greater than the second profile. For example, the profiles of the two patterns shown in Fig. 4c are not greater than the profile of the pattern in Fig. 4b (and incomparable with each other).

The introduced $\mathcal{E}_n$ and $\leq_n$ are obviously computable, even in polynomial time. Some work is required to show that $\mathcal{E}_n$ and $\leq_n$ satisfy Definition 21:

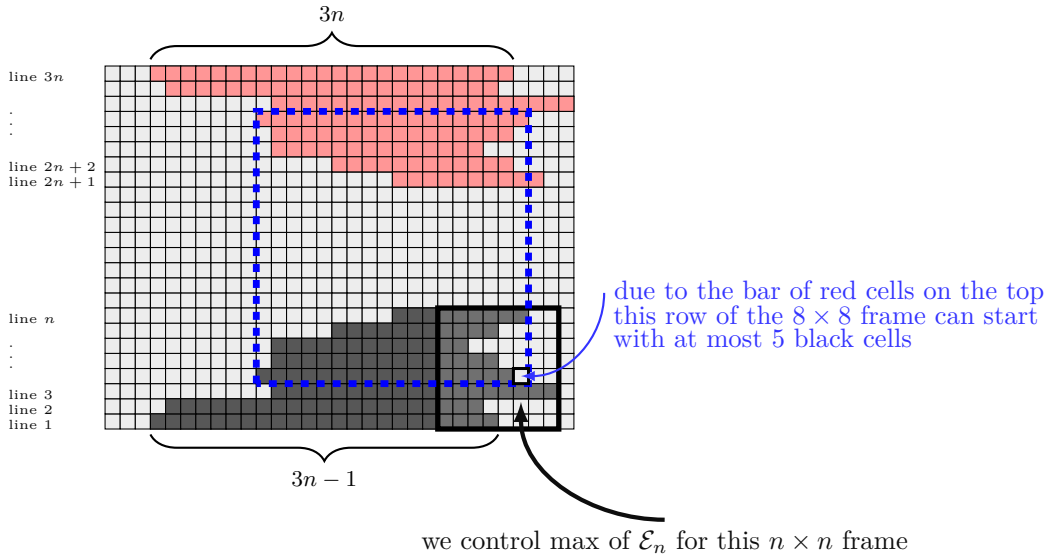


Figure 5 An $n \times n$ pattern P with a neighborhood that enforces the desired maximum of $\mathcal{E}_n$.

▶ **Lemma 26.** *The defined above* $(\mathcal{E}_n, \leq_n)$ *provide a family of exp-time computable ordered epitomes for the shift under consideration.*

This lemma is proven implicitly in [5]. In what follows, for the sake of self-containedness, we sketch this proof.

Proof of Lemma 26. For every *simple pattern* P of size $n \times n$ we should construct a configuration R on the complement of B_n, so that

(i) P and R are compatible,

(ii) for every other simple pattern P' compatible with R we have $\mathcal{E}_n(P') \leq_n \mathcal{E}_n(P)$.

We build R by following the construction from [5]. By definition, each row of P consists of a contiguous sequence of black cells followed by a contiguous sequence of white cells, as shown in Fig. 4b. The pattern R will consist of a finite number of black and red cells (the other cells will be white).

Black cells in R. To construct R, we extend each stripes of black cells in P to the left, so that in the first line we get a contiguous sequence of $(3n-1)$ black cells (including those black cells that belong to P), in the second line a contiguous sequence of $(3n-3)$ black cells, in the third line a contiguous sequence of $(3n-5)$ black cells, etc. In the n-th line we obtain a contiguous sequence of $(n+1)$ black cell, see Fig. 5.

Red cells in R. Similarly, we put in R stripes of red cells: $3n$ contiguous red cells in line $3n$, $(3n-2)$ contiguous red cells in line $3n-1$, …, $(n+2)$ contiguous red cells in line $(2n+1)$. We place these stripes of red cells so that for each $i = 1, \ldots, n$ the leftmost red cell in the line $(3n-i+1)$ is vertically aligned with the leftmost black cell in the line i, as shown in Fig. 5.

All other cells outside B_n are made white.

▷ Claim 27. The constructed R is compatible with P.

Proof of Claim 27. This fact is easy to verify: we have chosen the lengths of black and red stripes so that they cannot form a forbidden pattern (as in Fig. 4a), regardless the horizontal placement of each stripes. Indeed, on the one hand, the black cells of the i-th line cannot

interfere with the red stripes in lines $3n, 3n-1, \ldots, 3n-i$, since *this black stripe* is too short to form a forbidden pattern together with any of these red stripes; on the other hand, the black cells of the i-th line cannot interfere with the red stripes in lines $3n-i-1, 3n-i-2, \ldots, 2n+1$, since *those red stripes* are too short. ◁

▷ Claim 28. The constructed pattern R is compatible only with simple patterns P' such that $\mathcal{E}_n(P') \leq_n \mathcal{E}_n(P)$.

Proof of Claim 28. If R is compatible with an $n \times n$ pattern P', the profile of P' is not determined uniquely. In fact, R can be compatible with simple patterns P' whose profiles are *strictly less* than the profile of P (in each row of P' the number of black cells must be not greater than the number of black cells in the corresponding row of P), see Fig. 6 below. On the other hand, if at least one row of P' contains more black cells that the same row in P, than P' and R are incompatible, i.e., the joint of P' and R contains a forbidden pattern, as shown in Fig. 7. ◁

The lemma follows from Claim 27 and Claim 28. For a more detailed argument we refer the reader to [5]. ◀

▶ Remark 29. In the construction discussed above, pattern R does not determine uniquely the epitomes of P' compatible with R (these epitomes can be different, though they must be *not greater* than the epitome of the initial pattern P). This is why we cannot apply Proposition 19, and we have to employ the extended definition of *partial* epitomes.

To prove the proposition, it remains to observe that for every n there are $(n+1)^n$ simple patterns of size $n \times n$ (in each row of a simple pattern the frontier between black and white areas varies between 0 and n). Therefore, for some simple patterns P of size $n \times n$ the Kolmogorov complexity of their profile is greater than $n \log(n+1)$, i.e., even the plain Kolmogorov complexity $\mathrm{C}(P)$ is super-linear. We apply Proposition 23 and conclude that the shift is not sofic. ◀

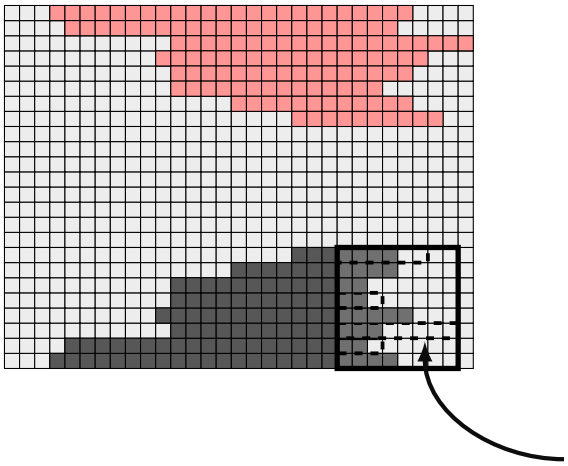


Figure 6 A pattern P' with $\mathcal{E}_n(P') \leq_n \mathcal{E}_n(P)$ matches the neighborhood.

▶ **Open Problem 1.** Is there any *sufficient* condition of soficness for effective shifts that can be formulated in terms of resource-bounded Kolmogorov complexity?

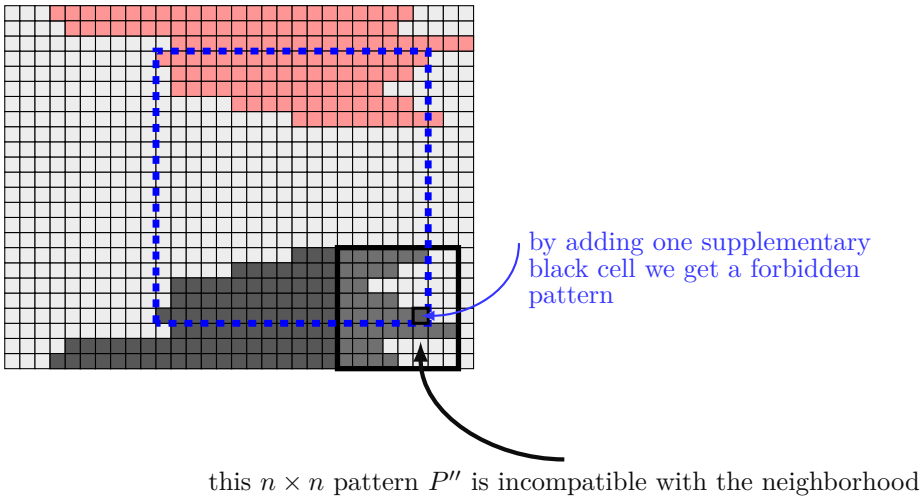


Figure 7 A pattern P'' with $\mathcal{E}_n(P'') \not\leq_n \mathcal{E}_n(P)$ does not match the neighborhood.

▶ **Open Problem 2.** The shift in Example 24 has positive entropy, and in the argument discussed above we could employ the definition of uniformly computable (but not exp-time computable) ordered epitomes. It would be interesting to suggest a natural example of an effective (but non-sofic) shift where the technique of *exp-time computable* ordered epitomes is valid while *uniformly computable but not exp-time computable* ordered epitomes do not apply.

References

1 N. Aubrun, S. Barbieri, and E. Jeandel. About the Domino Problem for Subshifts on Groups. Sequences, Groups, and Number Theory. *Birkhäuser, Cham*, pages 331–389, 2018.

2 B. Durand, L. Levin, and A. Shen. Complex tilings. *The Journal of Symbolic Logic*, 73:2:593–613, 2008.

3 F.C. Hennie and R.E. Stearns. Two-tape simulation of multitape Turing machines. *Journal of the ACM*, 13(4):533–546, 1966.

4 E. Jeandel. Propriétés structurelles et calculatoires des pavages. *Habilitation thesis, Université Montpellier 2*, 2011.

5 S. Kass and K. Madden. A sufficient condition for non-soficness of higher-dimensional subshifts. *Proceedings of the American Mathematical Society*, 141:11:3803–3816, 2013.

6 A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of information transmission*, 1:1:1–7, 1965.

7 M. Li and P. Vitányi. An introduction to Kolmogorov complexity and its applications. *3rd ed. Springer, New York*, 2008.

8 D. Lind and B. Marcus. An Introduction to Symbolic Dynamics and Coding. *Cambridge University Press*, 1995.

9 M.Morse and G.A. Hedlund. Symbolic dynamics. *Amer. J. Math.*, 60:815–866, 1938.

10 M. Morse and G. A. Hedlund. Symbolic dynamics II: Sturmian trajectories. *Amer. J. Math.*, 62:1–42, 1940.

11 S. Mozes. Tilings, substitution systems and dynamical systems generated by them. *Journal d'analyse mathématique (Jerusalem)*, 53:139–186, 1989.

12 N. Ormes and R. Pavlov. Extender sets and multidimensional subshifts. *Ergodic Theory and Dynamical Systems*, 36:3:908–923, 2016.

13 R. Pavlov. A class of nonsofic multidimensional shift spaces. *Proceedings of the American Mathematical Society*, 141:3:987–996, 2013.

14 A. Rumyantsev and M. Ushakov. Forbidden substrings, Kolmogorov complexity and almost periodic sequences. *In Proc. Annual Symposium on Theoretical Aspects of Computer Science*, pages 396–407, 2006.

15 V. Salo. Subshifts with sparse projective subdynamics. *arXiv preprint*, arXiv:1605.09623, 2016.

16 B. Weiss. Subshifts of finite type and sofic systems. *Monatsh. Math.*, 77:462–474, 1973.

17 L.B. Westrick. Seas of squares with sizes from a Π^0_1 set. *Israel Journal of Mathematics*, 22:1, 2017.

Constant-Time Retrieval with $O(\log m)$ Extra Bits

Martin Dietzfelbinger
Technische Universität Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de

Abstract

For a set $\mathcal{U}$ (the *universe*), *retrieval* is the following problem. Given a finite subset $S \subseteq \mathcal{U}$ of size m and $f\colon S \to \{0,1\}^r$ for a small constant r, build a data structure D_f with the property that for a suitable query algorithm query we have $\mathsf{query}(D_f, x) = f(x)$ for all $x \in S$. For $x \in \mathcal{U} \setminus S$ the value $\mathsf{query}(D_f, x)$ is arbitrary in $\{0,1\}^r$. The number of bits needed for D_f should be $(1+\varepsilon)rm$ with *overhead* $\varepsilon = \varepsilon(m) \geq 0$ as small as possible, while the query time should be small. Of course, the time for constructing D_f is relevant as well.

We assume fully random hash functions on $\mathcal{U}$ with constant evaluation time are available. It is known that with $\varepsilon \approx 0.09$ one can achieve linear construction time and constant query time, and with overhead $\varepsilon_k \approx e^{-k}$ it is possible to have $O(k)$ query time and $O(m^{1+\alpha})$ construction time, for arbitrary $\alpha > 0$. Furthermore, a theoretical construction with $\varepsilon = O((\log\log m)/\sqrt{\log m})$ gives constant query time and linear construction time. Known constructions avoiding all overhead, except for a seed value of size $O(\log\log m)$, require logarithmic query time.

In this paper, we present a method for treating the retrieval problem with overhead $\varepsilon = O((\log m)/m)$, which corresponds to $O(1)$ extra memory words ($O(\log m)$ bits), and an extremely simple, constant-time query operation. The price to pay is a construction time of $O(m^2)$. We employ the usual framework for retrieval data structures, where construction is effected by solving a sparse linear system of equations over the 2-element field $\mathbb{F}_2$ and a query is effected by a dot product calculation. Our main technical contribution is the design and analysis of a new and natural family of sparse random linear systems with m equations and $(1+\varepsilon)m$ variables, which combines good locality properties with high probability of having full rank.

Paying a larger overhead of $\varepsilon = O((\log m)/m^{\alpha})$, the construction time can be reduced to $O(m^{1+\alpha})$ for arbitrary constant $0 < \alpha < 1$. In combination with an adaptation of known techniques for solving sparse linear systems of equations, our approach leads to a highly practical algorithm for retrieval. In a particular benchmark with $m = 10^7$ we achieve an order-of-magnitude improvement over previous techniques with $\varepsilon = 0.24\%$ instead of the previously best result of $\varepsilon \approx 3\%$, with better query time and no significant sacrifices in construction time.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Retrieval, Hashing, Succinct Data Structure, Randomised Data Structure, Structured Gaussian Elimination, Method of Four Russians

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.24

1 Introduction

A *retrieval data structure* for a universe $\mathcal{U}$ (a set) and r-bit values represents a function from $\mathcal{U}$ to $R = \{0,1\}^r$ with prescribed values on a set $S \subseteq \mathcal{U}$ of size m. We need an algorithm construct that builds the data structure and an operation query. The input for construct is a function $f\colon S \to R$ (given as a list of argument-value pairs), the result is a data structure D_f, whose binary length we denote by $|D_f|$. The query algorithm query has two inputs, D_f and $x \in \mathcal{U}$, and returns an element of R. We require that

$\mathsf{query}(D_f, x) = f(x)$, for all $x \in S$.

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 24; pp. 24:1–24:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The values $\mathsf{query}(D_f, x)$ for $x \in \mathcal{U} \setminus S$ are irrelevant.

Relevant parameters of a retrieval data structure are (1) the space $|D_f|$ in bits in terms of $m = |S|$ and r, (2) the running time of query, and (3) the running time of construct. In this paper, we focus on minimising (1) while keeping (2) a constant, as small as possible. As for (3), it is kept in the $O(m^{1+\alpha})$ range. We make some efforts to extend the range of practical usability of our approach. (This is only partly reflected in this paper.) Note that storing all pairs $(x, f(x))$ for $x \in S$ in a dictionary data structure is not good enough for our purposes, as in general this requires $|D_f| = rm + \Omega(m \log m)$ bits. Since it is not necessary to decide membership in S or a subset of S, space $|D_f| = O(rm)$ is sufficient.

There is a close connection between retrieval data structures and perfect hashing. Most of the perfect hash functions that get by with linear space utilise a retrieval data structure with $r = 2$, see [6, 7, 15]. (Exceptions are the optimal theoretical construction in [16] and Hash-Displace-Compress in [4].) Conversely, if a hash function $g\colon \mathcal{U} \to [p]$ is given that is perfect on S, one can store $f(x)$, $x \in S$, in position $T[g(x)]$ of a table $T[0..p-1]$ to obtain a retrieval data structure with space pr plus the space for storing g. However, if r is small and the goal is a retrieval structure with very small space overhead, this detour via perfect hashing is inefficient.

1.1 Basic Data Structure

The basic setup of the data structure is well known and well studied. For some $n = n(m) \geq m$, a set $W \subseteq \{0,1\}^n$ of (typically sparse) vectors is chosen. One hash function or several hash functions are used to map elements x of $\mathcal{U}$ to $row(x) \in W$.

The data structure D_f consists of row[1] and a vector $\vec{z} = (z_1, \dots, z_n) \in (\{0,1\}^r)^n$, which in construct is chosen in such a way that the query algorithm

$$\mathsf{query}(D_f, x) = \bigoplus_{\substack{1 \leq i \leq n \\ (row(x))_i = 1}} z_i, \quad \text{for } x \in \mathcal{U}, \tag{1}$$

yields $f(x)$ for $x \in S$. To carry out $\mathsf{query}(D_f, x)$, one only has to find the components z_i with $(row(x))_i = 1$ and perform $\bigoplus$, the bitwise XOR, of these components.

Note that in this construction the $r \geq 1$ components of elements of R are just treated independently (or "in parallel"). In order to simplify notation, we concentrate on the case $r = 1$ from here on. (The generalisation is immediate. The query time has to be multiplied by r.) In this case (1) turns into

$$f(x) = \langle row(x), \vec{z} \rangle, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ is the dot product of n-bit vectors. For construct one has to solve a linear system $(\langle row(x), \vec{z} \rangle = f(x))_{x \in S}$ over $\mathbb{F}_2$ for the vector $\vec{z}$ of "unknowns". For this to be possible it is sufficient that the $m \times n$ matrix $A(S, row)$ with rows $row(x)$, $x \in S$, has full row rank. We assume that $row(x)$, $x \in U$, are stochastically independent and identically distributed, and we may simply write $A_{m,n}$ for a random variable distributed as $A(S, row)$, if the distribution of row is clear from the context.

[1] In all our constructions we assume that fully random independent hash functions $(\varphi_{i,j})_{i,j}$ with suitable range $\{1, \dots, \mathrm{poly}(m)\}$ are available for free and can be evaluated on $x \in \mathcal{U}$ in constant time. Fixing a seed value j, one gets $(h_0, h_1, \dots) = (\varphi_{0,j}, \varphi_{1,j}, \dots)$ from which row is constructed. Hence, an index j suffices to identify row.

Algorithm
construct($S \subseteq \mathcal{U}, f\colon S \to \{0,1\}$):
  pick $row\colon S \to W \subseteq \{0,1\}^{(1+\varepsilon)|S|}$
  solve $(\langle row(x), \vec{z}\rangle = f(x))_{x\in S}$ for $\vec{z}$
  restart with new row if unsolvable
  return $D = (row, \vec{z})$

Algorithm
query($D_f = (row, \vec{z}), x \in \mathcal{U}$):
  return $\langle row(x), \vec{z}\rangle$

Figure 1 The general framework for retrieval data structures for $R = \{0,1\}$.

1.2 Previous Work and Relevant Techniques

Construction time and query time depend very much on the structure of the vectors in W. Most earlier works [6, 7, 8, 11, 15] chose W as the set of vectors with constant *Hamming weight* k, i.e. k entries of $row(x)$ are 1, for some $k \geq 3$. The operation query$(x, \vec{z})$ then just reads the k positions given by $row(x)$ in $\vec{z}$, which results in running time $O(k)$ for a query. (Only in [11, 21] sets W with vectors of Hamming weight $\Theta(\log m)$ are considered. Applied in the obvious way, this will lead to query times of $\Theta(\log m)$.) There is a simple case, which also admits linear construction time, namely when the rows of $A_{m,n}$ can be brought into triangular form by row and column *exchanges* alone. This is equivalent to the k-uniform hypergraph $G_{m,n}$ with incidence matrix $A_{m,n}^{\mathsf{T}}$ being *peelable*, i.e. having an empty *2-core* [18]. In this case we do not even need a field to compute in (and no linear algebra), but (1) can be taken to be a formula over any group $(R, \oplus)$ (like $\mathbb{Z}/m\mathbb{Z}$ with addition modulo m). This approach is underlying the constructions in [6, 7, 8, 15, 18, 23]. The common feature utilised in these works is that for fixed $k \geq 2$ there is a density threshold $c_k^{\triangle}$ such that, for arbitrary constant $\delta > 0$, $G_{m,n}$ is peelable with high probability (whp) for $m < (c_k^{\triangle} - \delta)n$ and not peelable whp for $m > (c_k^{\triangle} + \delta)n$. A description of these thresholds can be found in [18], a proof in [19]. The largest threshold value is $c_3^{\triangle} \approx 0.81847$. In [22] it was shown that rows with nonuniform weight (e.g. 3 for about 88% of keys and 21 for the rest) can raise the threshold to over 0.92. It is open if with constant average Hamming weight a quotient m/n arbitrarily close to 1 will still lead to peelability whp.

As our aim is to achieve $m = cn$ for $c = 1 - o(1)$, we have to give up peelability. A standard approach [2, 11, 15] uses bit vectors with constant weight $k \geq 3$ and in the construct routine requires solving the linear system (2) over $\mathbb{F}_2$. In [10] it was claimed and in [20] it was rigorously proved that there are thresholds c_k^* such that for $c < c_k^*$ and $m = cn$, $m, n \to \infty$ we have solvability whp, and for $c > c_k^*$ and $m = cn$, $m, n \to \infty$ we have unsolvability whp. These thresholds are the same as for simple orientability of k-uniform hypergraphs [10, 13, 14]. (Numerical values for some k can be found in [10].) The values $1 - c_k^*$ approach e^{-k} as k increases, and hence c_k^* is quite close to 1, but for constant k a constant gap remains. Queries take time $\Theta(k)$ and are not cache efficient, since k random components of $\vec{z}$ are accessed. In [11] it was shown (utilising a result from [9]) that with $k = k(m) = O(\log m)$, one can achieve solvability for $m = n$, with constant probability. A similar observation was made in [21]. The query time increases to $O(\log m)$.

A serious obstacle in these methods for reducing the overhead is that one has to solve a linear system without the advantage of peelability. So it is necessary to address the running time of this task. Gaussian elimination, applied naively, needs time $\Theta(m^3)$, which already for moderately large m becomes infeasible. Exploiting the fact that the rows are sparse, a variant of Wiedemann's algorithm [24] will reduce the time to $O(m^2 \log^2 m)$ (for fixed k) and to $O(m^2 \log m)$ (for $k = k(x) = O(\log m)$). Alternatively, one may use clever

variants of Gaussian elimination ("structured" in [17] and "lazy" in [15]), which try to delay the system filling up with 1's. As far as we know, there is no mathematical analysis of such techniques, although in experiments they drastically reduce the running time of naive Gaussian elimination on sparse random matrices, outperforming Wiedemann's algorithm for small to medium inputs. So, we do not use these techniques in our theoretical analysis, but we utilise them in our experimental implementations. Another standard speedup technique for Gaussian elimination, of course, is word-level parallelism, where the rows of the linear system are split into machine words and row operations can be performed by a sequence of bitwise boolean XORs. If the word length is w, the running times may be divided by w. Finally, the "Technique of Four Russians" is applicable to Gaussian elimination. By filling a lookup-table with certain precomputed row sums, it achieves a speedup by a factor of $\Theta(\log m)$; for a description see [3].

A last, very important technique for achieving feasible construction times for large retrieval data structures is *input partitioning*. It has been used in many works on dictionary-like data structures, see, e.g., [2, 11, 15, 21]. Let $[n]$ denote the set $\{1, \ldots, n\}$. Using a "level-1" hash function $h_0 \colon \mathcal{U} \to [m/C]$, for some C (which w.l.o.g. divides m), one splits S into *chunks* $S_i = h_0^{-1}(\{i\}) \cap S$, for $i = 1, \ldots, m/C$. The expected size of each chunk is C, and if h_0 is fully random and C is not too small, one has $|S_i| = O(C)$ for all i, whp. One sets up a separate retrieval data structure $D_{f \restriction S_i}$ for each S_i. When using Gaussian elimination with $|C| = m^\alpha$ to construct each $D_{f \restriction S_i}$, the total construction time is reduced to $O((m/C) \cdot C^3) = O(mC^2) = O(m^{1+2\alpha})$, and with Wiedemann's algorithm [24] we get construction time $O((m/C) \cdot C^2 \log^2(C)) = O(m^{1+\alpha} \log^2 m)$. The downside is that one gets an additional "outer overhead" for saving for each i a pointer to $D_{f \restriction S_i}$. Using the offsets $\sum_{1 \le i < j} |D_{f \restriction S_i}|$ for this purpose costs $O((m/C) \log m)$ bits. This space we have to pay for the reduction in construction time. In [11] and [21] chunks of size $O(\sqrt{\log m})$ are used. An extra auxiliary data structure is employed to accommodate "bad" keys from chunks that overflow. Moreover, these papers use lookup tables for solving tiny systems of equations ($O(\sqrt{\log m})$ variables and equations). While using sublogarithmic chunk sizes has good properties in theory, it does not seem to be competitive in practice, see [2].[2]

1.3 Our Contribution

There are three degrees of freedom in the retrieval framework described above, regarding both theory and implementation:

- **(F1)** What is the set W of "sparse" vectors that is the range of *row*?
- **(F2)** Which method is used for solving the linear system?
- **(F3)** How, if at all, do we partition the input to reduce the influence of a high running time of the solver?

The main contribution of this paper is to propose and analyse a new answer to **(F1)**. The effect is that the query time is now constant – it involves only access to two memory locations and a small dot product calculation –, while the overhead drops to $m^{-\alpha}$ for a constant $\alpha \in (0, 1)$, or even to $O((\log m)/m)$, which means that $n = m + O(\log m)$. The method is very simple and natural: The 1's in $row(x)$ are concentrated in two blocks of size $O(\log m)$, so that the non-zero part of $row(x)$ fits in $O(1)$ words in the (standard) word RAM model.

[2] In [12] it is explained how one can justify the full-randomness assumption by partitioning S and employing an auxiliary data structure of size $C^{1+\Omega(1)}$ to provide fully random hash functions on each chunk. We do not discuss this aspect of partitioning here.

(This computational model was also used in [11, 21], and it is the basis of the speedup of the Four Russians method.) We are not aware of this idea having been used in the context of retrieval structures or perfect hashing before.[3] Its appeal is in its simplicity and in its apparent practicality. (Our experiments show that a significant reduction in overhead is possible for sets S of realistic size, with construction times comparable to the other approaches.) Let $n = m \cdot (1+\varepsilon)$. We describe $W \subseteq \{0,1\}^n$ and a distribution of $row(x) \in W$ as follows. Fix a *block size* $\ell = O(\log n)$ that divides n. For $b \in [n/\ell]$ and $p \in \{0,1\}^\ell$ we let $B_{b,p} = 0^{b\ell-\ell}p0^{n-b\ell} \in \{0,1\}^n$. Two block indices $b_1, b_2 \in [n/\ell]$ and two patterns $p_1, p_2 \in \{0,1\}^\ell$ are chosen uniformly at random, and $row(x)$ is set to be $B_{b_1,p_1} \oplus B_{b_2,p_2}$, with at most two non-zero blocks. Bit parallelism allows computing dot products involving $row(x)$, and thus answer queries, in $O(1)$ time. It is the main contribution of this work to establish that an overhead of $\Theta(\frac{\log m}{m})$ is achievable using this approach (we call it "inner overhead" if we want to distinguish it from the "outer overhead" needed due to the use of partitioning techniques). The theoretical analysis of the probability of obtaining a solvable system of equations, meaning that $A_{m,n}$ has full row rank, uses a first moment argument. Note that our result is also a significant theoretical improvement. Previously, overhead $\varepsilon = m^{-\delta}$ for constant δ required query cost $O(\log m)$ while query cost $O(1)$ required overhead $\varepsilon = \Omega(\frac{\log\log m}{\sqrt{\log m}})$, see Table 1.

▶ **Theorem 1** (Main Theorem). *Assume the context of a word RAM with word length $w = \Theta(\log n)$ and access to fully random hash functions*[4]*.*

- **(i)** *For any $r < m$ there is an r-bit retrieval data structure with overhead $\varepsilon = O(\frac{\log m}{m})$ and a construction that succeeds in time $O(\frac{m^3}{w \log m})$ whp. Queries take $O(r)$ time and access two contiguous segments of memory.*
- **(ii)** *An alternative construction based on Wiedemann's algorithm runs in time $O(rm^2)$ whp.*

Construction time can be improved by taking **(F3)** into account: randomly partition the input into chunks of expected size C and use the construction from Theorem 1 on each of the $\frac{m}{C}$ chunks. For each chunk, we need to store a pointer to its data and a seed for the function row used in the successful construction. This requires $O(\frac{m}{C}\log m)$ and $O(\frac{m}{C}\log\log m)$ extra bits, respectively. We then get:

▶ **Corollary 2.** *Under the same conditions as Theorem 1, we have:*

- **(i)** *For any $C = m^\alpha$ $(0 < \alpha \leq 1)$ and any $r < C$ there is an r-bit retrieval data structure with overhead $\varepsilon = O(\frac{\log m}{C})$ and a construction that succeeds in time $O(\frac{mC^2}{w \log C})$ whp. Queries take $O(r)$ time and access three contiguous segments of memory*[5]*.*
- **(ii)** *A alternative construction based on Wiedemann's algorithm runs in time $O(rmC)$.*

Table 1 summarises previous work and the new construction if the effect of partitioning is taken into account. Also, obvious improvements achieved by replacing Gaussian elimination by Wiedemann's algorithm are reflected. The choice of C constitutes a trade-off between construction time and total overhead, which is the sum of the overhead from the chunks ("inner") and from organising the data structures for the single chunks ("outer"). For $C = m$ there are no chunks, the construction time is maximal, and only the "inner" overhead is

[3] Possibly it was vaguely anticipated in [21], where the author suggested using "sparse equations that are more or less local".

[4] For any universe $\mathcal{U}$ and any finite domain D we assume oracle access to fully random functions $(h_i : \mathcal{U} \to D)_{i\in\mathbb{N}}$, meaning we need to only store an index i to describe such a function. This assumption is motivated by the observation that good (pseudo-)randomness is usually not an issue in practice.

[5] In practice it is reasonable to expect two cache faults per query.

Table 1 Comparison of previous work and the results of this paper. Where query times are not enclosed in O-Notation the number vaguely counts accesses to random memory locations. The column $t_{\mathsf{construct}}$ reports the construction times given in the respective paper and alternatively times improved by utilising Wiedemann's algorithm [24]. Regarding the results of [11], the better thresholds from [20] are substituted.

Paper	t_{query}	$t_{\mathsf{construct}}$	"inner overhead" + "outer overhead"	Practical?
[7, 18]	3	$O(m)$	$0.23 + 0$	✓
[22]	$O(1)$	$O(m)$	$0.087 + 0$	✓
[11]	k	$O(m^3)$ or $O(m^2 \log^2 m)$	$\mathrm{e}^{-k} + o(\mathrm{e}^{-k}) + 0$	✗
[11]	$O(k)$	$O(m)$	$\mathrm{e}^{-k} + o(\mathrm{e}^{-k}) + \Omega((\log m)^{-1/4})$	✗
[11, 21]	$\log m$	$O(m^3)$ or $O(m^2 \log^2 m)$	$0 + O(\frac{\log\log m}{m})$	✗
[21]	$O(1)$	$O(m)$	$0 + \Omega(\frac{\log\log m}{\sqrt{\log m}})$	✗
[2]	$O(k)$	$O(mC^2)$ or $O(mC \log^2 C)$	$\mathrm{e}^{-k} + o(\mathrm{e}^{-k}) + \Omega(C^{-1/2})$	(✓)
[15]	3 [or 4]	$O(\frac{mC^2}{w})$ or $O(mC \log^2 C)$	0.09 [or 0.024] $+ \Theta(\frac{\log m}{C})$	✓
⟨**new**⟩	2	$O(\frac{mC^2}{w \log C})$ or $O(mC)$	$\Theta(\frac{\log m}{C}) + \Theta(\frac{\log m}{C})$	✓

relevant. Both from a theoretical and practical point of view the reduction from "ε is constant" or "$\varepsilon = O(\frac{\log\log m}{\sqrt{\log m}})$" to a polynomially small overhead is significant. Pleasingly, our approach compares very favourably with previous results in practical benchmarks, as shown in Table 2 and explained in Section 5.

Table 2 Comparison of our algorithm in the form presented in Section 5 to the arguably best-so-far results reported in [15]. We achieve much smaller overhead with comparable run times.

	overhead	Construction [µs/key]	Lookup [ns]
[15] $k = 3$	9%	1.12	210
[15] $k = 4$	3%	1.75	236
⟨this paper⟩	0.24%	2.6	75–125

Structure of the paper. In Section 2 we define a matrix $A^{\ell}_{m,n}$ and a related graph $G^{\ell}_{m,n}$ that formally capture the problem of constructing our retrieval data structure. In Section 3 we show that $A^{\ell}_{m,n}$ has full rank whp, which is the main ingredient used to prove our Main Theorem in Section 4. Lastly, in Section 5 we briefly present an implementation of our approach. A discussion of practical improvements we employed is postponed to the full version of this paper.

2 The Construction Problem in Matrix and Graph Terminology

We now formalise our idea of using "vectors with coefficients within two blocks".

Let $S \subseteq \mathcal{U}$ be a set annotated with $f\colon S \to \{0,1\}$, $\ell \in \mathbb{N}$ the *block size*, $|S| = m$ and $n \geq m$ with $\ell \mid n$. Moreover, we pick a uniformly random function $h\colon \mathcal{U} \to [n/\ell] \times [n/\ell] \times \{0,1\}^{\ell} \times \{0,1\}^{\ell}$ with components we call $h = (b_1, b_2, p_1, p_2)$ that implicitly characterise *row*. Together, (S, f, m, n, ℓ, h) is an instance of the construction problem for our retrieval data structures. The task to be solved can be expressed in two equivalent ways.

Matrix terminology. For $b \in [n/\ell]$ and $p \in \{0,1\}^\ell$ let $B_{b,p} = 0^{b\ell-\ell}p0^{n-b\ell} \in \{0,1\}^n$. Then each $x \in S$ is identified with the equation $\langle B_{b_1(x),p_1(x)} \oplus B_{b_2(x),p_2(x)}, \vec{z}\rangle = f(x)$ where $\vec{z} \in \{0,1\}^n$ is a vector of unknowns. Together, S is a system of equations $A\vec{z} = \vec{b}$ where $\vec{b} = (f(x))_{x\in S}$. The matrix $A = A^\ell_{m,n}$ will be examined thoroughly in Section 3.

Graph terminology. The problem instance can also be conveniently captured as a graph $G = G^\ell_{n,m} = ([n/\ell], S)$ with labels. Each vertex corresponds to a block of ℓ variables and $x \in S$ is identified with an edge $\{b_1(x), b_2(x)\}$ where the incidence $(x, b_1(x))$ is labelled with $p_1(x)$, the incidence $(x, b_2(x))$ is labelled with $p_2(x)$ and x itself is labelled with $f(x)$. A solution is now a vertex labelling $x\colon [n/\ell] \to \{0,1\}^\ell$ with $\langle p_1(x), x(b_1(x))\rangle \oplus \langle p_2(x), x(b_2(x))\rangle = f(x)$ for all $x \in S$.

We will borrow notions from graph theory in algebraic discussions, when convenient. For instance, we may speak of the degree of a block of variables or a connected set of equations, meaning the degree of a corresponding vertex or the connectedness of a corresponding set of edges.

Loops and parallel edges. It is possible to have $b_1(x) = b_2(x) = b$ for some $x \in S$. Then $row(x)$ contains $p_1 \oplus p_2$ in block b and G has a loop at vertex b with two labels p_1 and p_2. Moreover two distinct elements $x \neq x'$ may be associated with the same two blocks. In this case G is a multigraph.

Forbidding the all-zero pattern. Let $h^*\colon \mathcal{U} \to [n/\ell]\times[n/\ell]\times(\{0,1\}^\ell\setminus\{0^\ell\})\times(\{0,1\}^\ell\setminus\{0^\ell\})$ be uniformly random. Compared to h, the pattern 0^ℓ is forbidden in h^*. This gives rise to random matrices $A^{\ell*}_{m,n}$ and graphs $G^{\ell*}_{m,n}$ with higher probability of admitting solutions, see Proposition 3.

3 Full Rank of the Linear Systems

We now provide the main ingredient for Theorem 1, establishing that the matrices $A^\ell_{cn,n}$ defined in Section 2 have full rank whp. We also consider two natural variations concerning $A^{\ell*}_{cn,n}$.

Throughout this section, logarithms have base 2, $\bar{c}$ is a shorthand for $1-c$ and *with high probability* (whp) refers to a probability of $1 - n^{-\varepsilon}$ for some $\varepsilon > 0$.

▶ **Proposition 3.** *Let $\beta = 27$ and $\gamma = 1/4$. Then we have:*

- **(i)** *If $2\ell = 2\ell(n) \geq (1+\delta)\log n$ for $\delta > 0$, then $A^\ell_{cn,n}$ has independent rows whp, provided that $\bar{c} \geq \max\{2^{-\gamma\ell}, \beta\log(n)/n\}$.*
- **(ii)** *If $\ell = \ell(n) = \omega(1)$ then $A^{\ell*}_{cn,n}$ has independent rows with probability $1-o(1)$, provided that $\bar{c} \geq \max\{2^{-\gamma\ell}, \beta\log(n)/n\}$.*
- **(iii)** *If ℓ is a large enough constant, then $A^{\ell*}_{cn,n}$ has independent rows with probability $\Theta(1)$, provided that $\bar{c} \geq 2^{-\gamma\ell}$.*

Remarks.

- For $2\ell = \log n$ the matrix $A^\ell_{cn,n}$ has dependent rows with constant probability, simply because the number of all-zero rows is binomially distributed with expectation $cn \cdot 2^{-2\ell} =$

$c = \Theta(1)$. In this sense **(i)** is best possible. This motivates considering $A^{\ell*}_{cn,n}$ where all-zero rows are far less likely[6].

- For $\ell = \Theta(1)$ the probability that $A^{\ell*}_{cn,n}$ has two identical rows is $\Theta(1)$. This implies that $\ell = \omega(1)$ is best possible in **(ii)** and the probability of $\Theta(1)$ is best possible in **(iii)**.
- We have no reason to believe that $\gamma = 1/4$ and $\beta = 27$ are "best possible" or even "good". Note that for $\ell = 4\log n$ the bound on $\bar{c}$ becomes $\bar{c} \geq \beta \log(n)/n = \Theta(\frac{\log n}{n})$.
- We conjecture that for each $\ell \geq 2$ there is a threshold value $c^*_\ell \in (0,1)$ such that for $c < c^*_\ell$ the matrix $A^{\ell*}_{cn,n}$ has independent rows with probability at least $1/2$ and for $c > c^*_\ell$ it has dependent rows whp.

3.1 Proof of Proposition 3 (i)

Recall from Section 2 how $A = A^\ell_{cn,n}$ is obtained from S via a random hash function $h = (b_1, b_2, p_1, p_2)$ mapping elements to rows. If A does not have independent rows, then this is *witnessed* by a non-trivial subset $W \subseteq S$ of elements such that the corresponding rows of A sum to zero. We use a first moment calculation to show that whp no *inclusion-minimal* witness Y exists. We fix two parameters of candidate sets Y: The number $s = |W|$ of elements/rows, with $1 \leq s \leq m = cn$, and the number $t = |B| \in [n/\ell]$, where $B = \bigcup_{w\in W}\{b_1(w), b_2(w)\}$ is the set of variable blocks involved in at least one of the rows.

There are $\binom{m}{s}$ ways to choose Y, and $\binom{n/\ell}{t}$ ways to choose B. The probability that the rows corresponding to Y involve exactly the blocks from B is

$$\Pr[B = \bigcup_{w\in W}\{b_1(w), b_2(w)\}] \leq \prod_{w\in W} \Pr[b_1(w) \in B \wedge b_2(w) \in B] = (\tfrac{t}{n/\ell})^{2s}.$$

The event that the rows corresponding to W sum to zero is the intersection of the independent events that the rows sum to zero within each block $b \in B$. Its probability is therefore

$$\prod_{b\in B} \Pr\Big[\bigoplus_{\substack{(w,i)\in W\times\{0,1\}\\ b_i(w)=b}} p_i(w) = 0^\ell \;\Big|\; \exists (w,i) \in W \times \{0,1\} : b_i(w) = b\Big] = \prod_{b\in B} 2^{-\ell} = 2^{-\ell t}.$$

In the following, it is often convenient to deal with the fraction $\sigma = s/m$ of rows and the fraction $\tau = t/(n/\ell) = \ell t/n$ of blocks involved in a witness. Accordingly, we define $O(n^2/\ell)$ values $p_{\sigma,\tau}$, where $p_{\sigma,\tau}$ is the probability that some set of equations involving σm rows and exactly $\tau n/\ell$ blocks is a minimal witness. This gives

$$p_{\sigma,\tau} \leq \binom{m}{s}\binom{n/\ell}{t}(\tfrac{t}{n/\ell})^{2s}2^{-\ell t} = \binom{m}{\sigma m}\binom{n/\ell}{\tau n/\ell}\tau^{2\sigma m}2^{-\tau n}. \tag{3}$$

We now list a few bounds that will be useful later. Throughout, $\sigma \in \{\frac{1}{m}, \ldots, \frac{m}{m} = 1\}$ and $\tau \in \{\frac{1}{n/\ell}, \ldots, \frac{n/\ell}{n/\ell} = 1\}$.

▶ **Lemma 4.** *Let $\bar{\tau} = 1 - \tau$, $\bar{c} = 1 - c$, and let H be the binary entropy function. Then*

(a) $\frac{\ell}{n}\log(p_{\sigma,\tau}) \leq c\ell H(\sigma) + H(\tau) + 2\sigma c\ell \log\tau - \ell\tau = c\ell(H(\sigma) + \sigma\log\tau^2) + H(\tau) - \ell\tau.$

(b) $\frac{\ell}{n}\log(p_{\sigma,\tau}) \leq \ell(c\log(1+\tau^2) - \tau) + H(\tau).$

(c) $-\log\bar{c} \leq \ell/4$ (c1) $\qquad c \geq \frac{3}{4}$ (c2) $\qquad \bar{c} \geq 27\log(n)/n$ (c3)

(d) *All minimal witnesses satisfy $t \leq s + 1$.*

[6] An all-zero row requires an element x with hash value $h(x)$ fulfilling $b_1(x) = b_2(x)$ and $p_1(x) = p_2(x)$.

(e) $\log(1+\tau^2) \le \begin{cases} \tau \cdot 2\log\frac{5}{4} \le \frac{2}{3}\tau & \text{if } 0 < \tau \le \frac{1}{2}, \\ 1 - 2\bar{\tau} \cdot (1 - \log\frac{5}{4}) \le 1 - \frac{4}{3}\bar{\tau} & \text{if } \frac{1}{2} \le \tau \le 1, \\ \tau & \text{if } 0 < \tau \le 1. \end{cases}$

(f) $-\log\tau \le 2\bar{\tau}$ *if* $\frac{1}{2} \le \tau \le 1$.

(g) $-\tau_1 \log\tau_1 \le -\tau_2 \log\tau_2$ *for* $0 < \tau_1 < \tau_2 < \frac{1}{4}$.

(h) $H(\tau) \le -\tau\log\tau + 2\tau$ *if* $\tau \le \frac{1}{2}$.

(i) $H(\tau_1) < H(\tau_2)$ *for* $0 < \tau_1 < \tau_2 \le \frac{1}{2}$ *and* $H(\tau) = H(\bar{\tau})$ *for* $0 < \tau \le 1$.

(j) *If* $s \ge t$ *and* $\tau < 1/\ell$ *then* $\frac{\ell}{n}\log(p_{\sigma,\tau}) \le -\tau\ell/2$.

The claims of Lemma 4 can be verified with simple calculations, found in Section 3.3.

Different arguments will be used to get bounds on $p_{\sigma,\tau}$ for different ranges of τ. The sum of all $p_{\sigma,\tau}$ belonging to the same case will be $n^{-\varepsilon}$ for some $\varepsilon > 0$, which implies that A has full rank whp. In the proofs, we refer to parts of Lemma 4 by their labels.

Case 1: $c \le \tau \le 1$.

$$\frac{\ell}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,e)}}{\le} \ell(c\tau - \tau) + H(\tau) \overset{\text{(i)}}{\le} -\ell\tau\bar{c} + H(\bar{c}) \overset{\text{(h)}}{\le} -\ell\tau\bar{c} - \bar{c}\log\bar{c} + 2\bar{c}$$
$$\overset{\text{(c1)}}{\le} -\ell c\bar{c} + \bar{c}\ell/4 + 2\bar{c} = -\bar{c}(\ell c - \ell/4 - 2) \overset{\text{(c2)}}{\le} -\bar{c}\ell/2.$$

This gives a bound of:

$$p_{\sigma,\tau} \le 2^{-\bar{c}n/2} \overset{\text{(c3)}}{\le} n^{-27/2}.$$

Multiplying with $O(n^2)$ choices for σ and τ, this still gives a bound of $O(n^{-23/2})$.

Case 2: $1/2 \le \tau \le c$.

$$\frac{\ell}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,e)}}{\le} \ell(c(1-\tfrac{4}{3}\bar{\tau}) - \tau) + H(\tau) \overset{\text{(h,i)}}{\le} \ell(\bar{\tau} - \tfrac{4}{3}\bar{\tau}) + 2\bar{\tau} - \bar{\tau}\log\bar{\tau}$$
$$\le \bar{\tau}(-\ell/3 + 2 - \log\bar{c}) \overset{\text{(c1)}}{\le} \bar{\tau}(-\ell/3 + 2 + \ell/4) \le -\bar{c}\ell/13.$$

From this we obtain a bound $p_{\sigma,\tau} = n^{-27/13}$ and proceed as in Case 1.

Case 3: $\bar{c} \le \tau \le 1/2$.

$$\frac{\ell}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,e,f)}}{\le} \ell(\tfrac{2}{3}\tau - \tau) + H(\tau) \overset{\text{(h)}}{\le} -\tau\ell/3 - \tau\log\tau + 2\tau$$
$$\le \tau(-\ell/3 - \log\bar{c} + 2) \overset{\text{(c1)}}{\le} \tau(-\ell/3 + \ell/4 + 2) \le -\tau\ell/13 \le -\bar{c}\ell/13.$$

This is the same bound as in Case 2.

Case 4: $8\log(n)/n < \tau < 1/\ell$. Assuming $s \ge t$ for the moment, we may apply (j) to obtain $\frac{\ell}{n}\log(p_{\sigma,\tau}) \le -\tau\ell/2$. This gives $p_{\sigma,\tau} = 2^{-\tau n/2} \le 2^{-4\log n} \le n^{-4}$.
Inconveniently, (d) only gives $s \ge t-1$ instead of $s \ge t$ and the $O(n)$ cases with $s = t-1$ are not yet handled. Luckily, changing s by 1 (or equivalently σ by $1/m$) affects the upper bound in Equation (3) by at most $O(n^2)$ and the combined contribution of the cases in question is bounded by $O(n) \cdot O(n^2) \cdot n^{-4} = O(n^{-1})$.

Case 5: $2 \le t$ and $\ell^2 t^2 \le \frac{n}{2\mathrm{e}}$. We refine Equation (3) to get (recall $\sigma = \frac{s}{m}$, $\tau = \frac{\ell t}{n}$)

$$\begin{aligned} p_{\sigma,\tau} &\le \binom{m}{s}\binom{n/\ell}{t}\left(\tfrac{t}{n/\ell}\right)^{2s} 2^{-\ell t} \le \left(\frac{m\mathrm{e}}{s}\right)^s \left(\frac{en/\ell}{t}\right)^t \left(\frac{t}{n/\ell}\right)^{2s} 2^{-\ell t} \\ &= \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)^s \left(\frac{\mathrm{en}}{t\ell 2^\ell}\right)^t = \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)^{1+(s-t+1)+(t-2)} \left(\frac{\mathrm{en}}{t\ell 2^\ell}\right)^{2+(t-2)} \\ &= \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)\left(\frac{\mathrm{en}}{t\ell 2^\ell}\right)^2 \left(\frac{\mathrm{ce}^2\ell t}{s2^\ell}\right)^{t-2} \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)^{s-t+1} \\ &\le \frac{\mathrm{ce}^3 n}{2^{2\ell}}(\tfrac{1}{2})^{t-2}(\tfrac{1}{2})^{s-t+1} = \frac{\mathrm{ce}^3 n}{2^{2\ell}}(\tfrac{1}{2})^{s-1} \end{aligned}$$

where the last inequality used $\ell/2^\ell \le c\mathrm{e}^2/4$ (which holds for n sufficiently large), the upper bound on ℓt and $t/s \le 2$. It is crucial that the exponents $t-2$ and $s-t+1$ are nonnegative; for the latter exponent this is because of (d). The sum over all applicable s and t is dominated by the contribution for $t=2$ and $s=1$, since

$$\sum_{s\ge 1}\sum_{2\le t\le s+1} p_{\sigma,\tau} \le \sum_{s\ge 1}\sum_{2\le t\le s+1} \frac{\mathrm{ce}^3 n}{2^{2\ell}}(\tfrac{1}{2})^{s-1} = \frac{\mathrm{ce}^3 n}{2^{2\ell}}\sum_{s\ge 1} s\cdot(\tfrac{1}{2})^{s-1} = \frac{4\mathrm{ce}^3 n}{2^{2\ell}}.$$

Finally, using the assumption $2\ell \ge (1+\delta)\log n$, and thus $2^{2\ell} \ge n^{1+\delta}$, we obtain

$$\frac{4ce^3 n}{2^{2\ell}} = 4ce^3 n^{-\delta} = O(n^{-\delta}).$$

Case 5': $1 = t$ and $\ell \le n^{1/4}$. The argument from Case 5 essentially works, but the trivial bound $s \ge t-1 = 0$ needs to be replaced with $s \ge 1$. We get

$$\begin{aligned} \sum_{s\ge 1} p_{\sigma,\tau} &\le \sum_{s\ge 1}\binom{m}{s}\tfrac{n}{\ell}\left(\tfrac{\ell}{n}\right)^{2s} 2^{-\ell} \le \sum_{s\ge 1}\tfrac{n}{\ell\cdot 2^\ell}\left(\frac{m\ell^2}{sn^2}\right)^s \\ &= \tfrac{\mathrm{ce}\ell}{2^\ell}\sum_{s\ge 1}\left(\frac{\mathrm{ce}\ell^2}{sn}\right)^{s-1} \le O(n^{-1/2})\sum_{s\ge 1}\left(n^{-1/2}\right)^{s-1} = O(n^{-1/2}). \end{aligned}$$

Finally, we need to check that the case distinction is complete. If $\ell = \omega(\log n)$ then $t=1$ corresponds to $\tau = \ell/n \ge 27\log(n)/n = \bar{c}$ (using (c)) and Cases 1–3 already cover the entire range of τ. For more interesting values of $\ell = O(\log n)$ Cases 3 and 4 overlap due to (c) and Cases 4 and 5 overlap since the upper bound $\ell^2 t^2 \le \frac{n}{2\mathrm{e}}$ corresponds to $\tau = \ell t/n = O(n^{-1/2})$.

3.2 Adjustments for Proposition 3 (ii) and (iii)

We first outline how the argument from **(i)** needs to be modified to prove **(ii)**. Firstly, the probability that a sum within a block $b \in B$ is $\vec{0}$ is no longer $2^{-\ell}$. Assuming $\deg(b) = k$ and incidences labelled with uniformly random values $p_1, \ldots, p_k \in \{0,1\}^k - \{\vec{0}\}$ it is

$$\begin{aligned} &\Pr[p_1 \oplus \ldots \oplus p_k = \vec{0}] \\ =&\Pr[p_k = p_1 \oplus \ldots \oplus p_{k-1} \mid p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] \cdot \Pr[p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] \\ \le&\Pr[p_k = p_1 \oplus \ldots \oplus p_{k-1} \mid p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] = 1/(2^\ell - 1). \end{aligned}$$

The additive difference to the bound on $\frac{\ell}{n}\log(p_{\sigma,\tau})$ in Lemma 4(a) is

$$\tfrac{\ell}{n}\log((2^\ell/(2^\ell-1))^{\tau n/\ell}) = \tau\log(1+1/(2^\ell-1)) < 2t/(2^\ell-1) < 4\tau 2^{-\ell} \le 4\tau\bar{c}^4.$$

This is of lower order than the upper bound required in Cases 1 to 4 and thus inconsequential.

The only case where work is needed is Case 5 where the bound $\ell \geq (1+\delta)\log n$ is not available. Since the all-zero vector is forbidden as a coefficient vector for blocks of an equation, we know that minimal witnesses are not only connected but have minimum degree 2. The most extreme cases are then not trees with $s = t-1$, but cycles with $s = t$. The dominating term is then upper bounded by $\frac{c^2 e^4 \ell^2}{(2^\ell-1)^2}$, which is $o(1)$ because $\ell = \omega(1)$.

For **(iii)** we argue as in **(ii)**, except that when ℓ is constant the dominating term $\frac{c^2 e^4 \ell^2}{(2^\ell-1)^2}$ of Case 5 does not vanish for $n \to \infty$. However, if ℓ is large enough then the sum is less than 1, yielding a constant probability that no Case-5-type witness exists. Witnesses of other types have vanishing probability as before.

3.3 Proof of Lemma 4

Proof.

(a) This follows from Equation (3) after taking logarithms and multiplying by ℓ/n on both sides, using the standard approximation $\log \binom{n}{k} \leq nH(\frac{k}{n})$.

(b) This is obtained from (a) by observing that $H(\sigma) + \sigma \log \tau^2$ is concave as a function of σ and assumes its unique maximum value at $\sigma^* = \sigma^*(\tau) = \frac{\tau^2}{1+\tau^2}$.

(c) This is part of the assumption of Proposition 3**(i)**.

(d) If $W \subseteq S$, viewed as a subgraph of $G = G^\ell_{cn,n}$ (see Section 2), has two connected components $W = W_1 \cup W_2$, then the rows corresponding to W sum to zero if and only if the rows corresponding to W_1 and W_2 sum to zero individually, as they involve disjoint sets of variable blocks. In that case, W is not a minimal witness for dependence. In other words, we can restrict our attention to *connected* sets W. From this $t \leq s+1$ follows, the upper bound being attained if W corresponds to a tree in G.

(e) Since $g(\tau) = \log(1+\tau^2)$ is convex on $[0,1]$, we may the obtain upper bounds on g by linearly interpolating between the values $g(0) = 0$, $g(\frac{1}{2}) = \log \frac{5}{4}$ and $g(1) = 1$.

(f) Using that $g(\tau) = -\log \tau$ is convex we may obtain bounds on g by linearly interpolating between the values $g(1/2) = 1$, $g(1) = 0$.

(g) The function $g(\tau) = -\tau \log \tau$ is clearly continuous and its unique maximum is easily determined to be at $\tau = 1/\mathrm{e} > 1/4$, which implies the claim.

(h) $H(\tau) = -\tau \log \tau - \bar{\tau} \log \bar{\tau} \leq -\tau \log \tau - \log \bar{\tau} \overset{\text{(f)}}{\leq} -\tau \log \tau + 2\tau$.

(i) These properties of the entropy function are well known and easily checked.

(j) From $\sigma cn = s \geq t = \tau n/\ell$ we get $\sigma \geq \frac{\tau}{c\ell}$. Using the upper bound on τ we continue with $\sigma \geq \frac{\tau^2}{c} \geq \frac{\tau^2}{1+\tau^2}$. This means that all values permitted for σ exceed the argument $\sigma^* = \frac{\tau^2}{1+\tau^2}$ from (b) that maximises $H(\sigma) + \sigma \log \tau^2$. Again by concavity of this function we may refine the upper bound from (a) by substituting the smallest admissible value $\sigma = \frac{\tau}{c\ell}$. This yields:

$$\begin{aligned}\tfrac{\ell}{n} \log(p_{\sigma,\tau}) &\leq c\ell H(\tfrac{\tau}{c\ell}) + 2\tau \log \tau + H(\tau) - \ell\tau \overset{\text{(h)}}{\leq} -\tau \log(\tfrac{\tau}{c\ell}) + 2\tau + \tau \log \tau + 2\tau - \ell\tau \\ &= \tau \log(c\ell) + 4\tau - \ell\tau = \tau(\log(c\ell) - \ell + 4) \leq -\tau\ell/2.\end{aligned}$$

◀

4 Proof of the Main Theorem

With Proposition 3 in place, we can now prove Theorem 1.

Proof of Theorem 1.

(i) Aiming to apply Proposition 3**(i)**, we pick $\ell := 4\lceil \log m \rceil$, $\bar{c}' := \frac{27 \log m}{m}$, $c' := 1 - \bar{c}'$ and n as the least multiple of ℓ exceeding m/c'. We generate the matrix $A = A^{\ell}_{m,n}$ as defined in Section 2. For $c := m/n$ we can derive that by construction $\bar{c} = 1 - c = 1 - m/n \geq 1 - c' = \bar{c}' = \frac{27 \log m}{m}$ and then clearly $\bar{c} \geq 2^{-\ell/4} = O(m^{-1})$ holds as well. Thus Proposition 3**(i)** implies that A has full rank whp. Assume $r = 1$ for now. Solving a corresponding system $A\vec{z} = \vec{b}$ yields a retrieval data structure occupying n bits. The overhead is $O(\frac{\log m}{m})$ since

$$\frac{n}{m} - 1 \leq \frac{m/c' + \ell}{m} - 1 = O(\tfrac{\log m}{m}) + \tfrac{1-c'}{c'} \leq O(\tfrac{\log m}{m}) + \bar{c}' = O(\tfrac{\log m}{m}).$$

Construction time is dominated by the time to solve the linear system. We employ the Method of Four Russians [3] – a variant of Gaussian elimination – which requires $O(m^2/\log m)$ row additions. As rows contain $n + 1 = O(m)$ bits and w bits can be added in one word operation, we obtain a total runtime of $O(\frac{m^3}{w \log m})$. Queries access $\lceil \frac{\ell}{w} \rceil = O(1)$ memory words in two contiguous areas of memory, and require $O(1)$ bit-wise and operations as well as a parity operation. Query times are thus $O(1)$.
If $r > 1$ we need to solve $A \cdot X = B$ with $B \in \mathbb{F}_2^{m \times r}$ for $X \in \mathbb{F}_2^{n \times r}$. Solving the linear system for several right hand sides simultaneously comes at negligible additional cost. For cache efficient queries, blocks of X of size $\ell \times r$ should be stored contiguously, and each block should be stored column-wise.

(ii) We use Wiedemann's algorithm [24] to solve the system $A\vec{z} = \vec{b}$. As the algorithm only works for regular matrices we must first append $n - m$ rows to the full-rank matrix $A \in \mathbb{F}_2^{m \times n}$ such that the resulting square matrix $A' \in \mathbb{F}_2^{n \times n}$ is regular. It is well known that when picking rows uniformly from $\mathbb{F}_2^n$ this succeeds with probability $\Theta(1)$. We also append $n - m$ zeroes to b to obtain b'. The running time for solving the regular system $A'\vec{z} = \vec{b'}$ with Wiedemann's algorithm is dominated by the running time of $O(n)$ matrix-vector multplications involving A'. Note that multiplications with A can be carried out in time $O(m)$ using word operations. The additional rows of A' increase this by $O((m-n)n/w) = O(\frac{m \log m}{w})$, which is also $O(m)$ if $w = \Omega(\log m)$. The total runtime for $r = 1$ is thus $O(m^2)$. For $r > 1$, the algorithm must be repeated for each bit. ◀

In Section 5 we demonstrate that the approach in **(i)** admits a particularly efficient implementation in practice.

5 Experiments and Practical Considerations

5.1 Experimental Overhead

The benchmark uses the universe $\mathcal{U} = \text{ASCII}^*$ of strings, $S \subset \mathcal{U}$ is taken as the first $m = 10^7$ URLs from a `eu-2015-host` dataset gathered by [5] with ≈80 bytes per key, and for simplicity $f : \mathcal{U} \to \{0,1\}$ is taken to be the parity of the string length[7].

For hashing, we use $\mathsf{murmur} = \texttt{MurmurHash3\_x64\_128} : (\{0,1\}^8)^* \times \{0,1\}^{32} \to \{0,1\}^{128}$ [1], which conveniently has a second parameter. We use $(\mathsf{murmur}(\cdot, s))_{s \in \{0,1\}^{32}}$ as though it were a sequence of random independent hash functions. A hash function on $\mathcal{U}$ can thus

[7] Since the sequence of operations performed by the algorithm does not depend on f except, possibly, in the rare cases where singular linear systems are involved, the choice of f is largely inconsequential.

Table 3 Overview of all bits used in the data structure. The concrete values on the right correspond to a run on a data set with $m = 10^7$ keys, chunk size $C = 10^4$, $\ell = 16$ and $\varepsilon = 0.0005$. In that run $\lceil \log(1 + \max_i s_i) \rceil = 2$ and $\lceil \log(1 + \max_i d_i) \rceil = 9$.

Number of bits	bits used for	per element
m	entropy lower bound	1.000000
εm	intended inner overhead	0.000500
$\sum_i n_i - (1+\varepsilon)m$	padding ensuring $\ell \mid n_i$	0.000716
$\lceil \log(1 + \max_i s_i) \rceil \cdot m/C$	seed for each chunk	0.000200
$\lceil \log(1 + \max_i d_i) \rceil \cdot m/C$	offset info for each chunk	0.000900
[not discussed]	various global counters	0.000062
	all of the above	1.002378

be identified simply by s, the *seed*. One such hash function $h_0 : \mathcal{U} \to [\lceil m/C \rceil]$ partitions S into chunks $S_i = \{x \in S \mid h_0(x) = i\}$ for $0 \le i < \lceil m/C \rceil$ where the average chunk size was chosen as $C = 10^4$. The actual chunk sizes $m_i = |S_i|$ vary slightly around C.

For each i let n_i be the least multiple of the block size $\ell = 16$ that is at least $(1+\varepsilon)m_i$. Here, $\varepsilon = 0.0005$ is the intended inner overhead. Note that $n_i - (1+\varepsilon)m_i$ has an expectation of roughly $\frac{\ell-1}{2} = 7.5$. Within each chunk we generate and solve a system $A^{\ell}_{m_i,n_i}\vec{z_i} = \vec{b_i}$ yielding $\vec{z_i} \in \{0,1\}^{n_i}$. Construction is repeated with a new seed if necessary. Let s_i be the seed of the first successful construction for chunk i.

The vectors $\vec{z_i}$ are concatenated into one bit string $\vec{z}$. Let $o_i = \sum_{j<i} n_i/\ell$ be the offset (counted in blocks) where z_i starts within z. We store the values $d_i = o_i - \lfloor \frac{i-1}{\lceil m/C \rceil} |\vec{z}| \rfloor \approx o_i - \mathbb{E}[o_i]$ instead of the values o_i as their binary representation is typically only half as long.

Finally, let $\hat{d} := \max_i d_i$ and $\hat{s} := \max_i s_i$. In addition to $\vec{x}$, m, C, $\hat{d}$ and $\hat{s}$ we need to store the meta data $((s_i, d_i))_i$ for the chunks using $(\lceil \log(\hat{d}+1) \rceil + \lceil \log(\hat{s}+1) \rceil) \lceil m/C \rceil$ bits. A full account of everything that needs to be saved with concrete numbers is given in Table 3.

5.2 Experimental Runtimes

All tests were performed on an desktop computer with an Intel® Core i7-2600 Processor @ 3.40GHz. A direct comparison to results from [15] is given in Table 2.[8]

Construction. To solve the sparse linear system $A\vec{z} = \vec{b}$ in a chunk, we first employ a heuristic that reduces the system to a system $A'\vec{z}' = \vec{b}'$ that is dense but substantially smaller than A. We dub this step *BlockedLazyGauss* as it is heavily inspired by the LazyGauss algorithm from [15]. In our case of $\ell = 16$ only 15% of the variables from A remain in A'. The reduced system is then solved using the Method of Four Russians.

To highlight the influence of the chunk size C, we now consider construction time *per key*, which is $O(\frac{C^2}{w \log C})$. In Figure 2 we report the runtimes of our solver for $A^{\ell}_{m,n}\vec{z} = \vec{b}$, as well as the relative contribution of the LazyGauss and the Four-Russian phases. The time per key of ≈1.8µs reported there for $C = 10^4$ is also the main component of the time per key of ≈2.6µs for the complete construction algorithm. The additional time is mostly spent on

[8] Note that the implementations may be optimised to different degrees, and despite the fact that very similar CPUs were used, runtime comparisons should not be overinterpreted. The authors of [15] estimate a "tight C implementation [of their algorithm] would be about twice as fast".

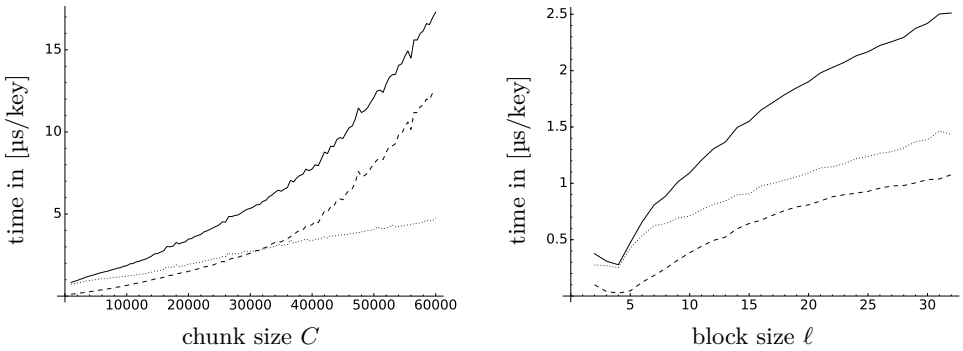


Figure 2 Time per key of our linear system solver with ⋰ representing the time for the BlockedLazyGauss-phase, ⋱ the time for the FourRussian-phase and ╱ the sum. On the left the block size is $\ell = 16$ and the chunk size C varies. On the right $C = 10^4$ and ℓ varies. The number of equations per chunk is $0.9995C$ and C, respectively.

streaming the key from a zipped file (≈0.3µs), hashing it, sorting it into the correct chunk as well as allocating and initialising the linear systems. Only a fraction of ≈0.005 of the linear systems fail to have full rank and require a restart for the chunk.

The work on the 10^3 chunks can be parallelised in a straightforward way, which brings construction time down to 1.1µs per key, using 4 cores with 2 logical processors each.

Query. A query involves computing hash values, accessing two ℓ-bit words, and very cheap and, xor and parity operations. In our experiments, computing hash values took ≈35ns. Overall query time was ≈75ns for $m = 10^6$ (ℓ, ε, C as above), when the retrieval data structure could reasonably be expected to reside in cache. Time increased to ≈125ns for $m = 10^8$, where the retrieval data structure certainly did not fit into cache.

6 Conclusion and Future Work

We introduced a new variant of constructing a retrieval data structure for m elements and range $\{0,1\}^r$ on the basis of the classical method of transforming keys into the rows of a linear system of equations over $\mathbb{F}_2$, and using the solution vector as the data structure. The new idea of having $O(\log m)$ many 1 entries in a row, concentrated in two blocks, in combination with word parallelism, gives constant query time on a word RAM. The construction time can be reduced by both exact and heuristic methods so as to achieve space $(1 + O((\log m)/m))mr = mr + O(\log m)r$ in theory and $1.0024m$ in realistic experiments with $r = 1$.[9] Future work could examine:

- Is it possible to achieve constant access time and additive overhead $O(\log \log n)$ by a variant of our construction using a square system? (This would be the case if such a systems had full rank with constant probability.)
- Study the behaviour of systems of equations as considered here for fields $\mathbb{F}_q$ of constant size $q > 2$.

[9] Table 3 suggests that we can obtain $\approx 1.0012mr + 0.0012m$ for general r.

References

1 Austin Appleby. MurmurHash3, 2012. URL: https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp.

2 Martin Aumüller, Martin Dietzfelbinger, and Michael Rink. Experimental Variations of a Theoretically Good Retrieval Data Structure. In *Proc. 17th ESA*, pages 742–751, 2009. doi:10.1007/978-3-642-04128-0_66.

3 Gregory V. Bard. *Algebraic Cryptanalysis*, chapter The Method of Four Russians, pages 133–158. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-88757-9_9.

4 Djamal Belazzougui, Fabiano Cupertino Botelho, and Martin Dietzfelbinger. Hash, Displace, and Compress. In *Proc. 17th ESA*, pages 682–693, 2009. doi:10.1007/978-3-642-04128-0_61.

5 Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUbiNG: Massive crawling for the masses. In *Proc. 23rd WWW'14*, pages 227–228, 2014. doi:10.1145/2567948.2577304.

6 Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Simple and Space-Efficient Minimal Perfect Hash Functions. In *Proc. 10th WADS*, pages 139–150, 2007. doi:10.1007/978-3-540-73951-7_13.

7 Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Practical Perfect Hashing in Nearly Optimal Space. *Inf. Syst.*, pages 108–131, 2013. doi:10.1016/j.is.2012.06.002.

8 Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables. In *Proc. 15th SODA*, pages 30–39, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982797.

9 Colin Cooper. On the rank of random matrices. *Random Structures & Algorithms*, 16(2):209–232, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<209::AID-RSA6>3.0.CO;2-1.

10 Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. In *Proc. 37th ICALP (1)*, pages 213–225, 2010. doi:10.1007/978-3-642-14165-2_19.

11 Martin Dietzfelbinger and Rasmus Pagh. Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract). In *Proc. 35th ICALP (1)*, pages 385–396, 2008. doi:10.1007/978-3-540-70575-8_32.

12 Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.

13 Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp Load Thresholds for Cuckoo Hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012. doi:10.1002/rsa.20426.

14 Alan M. Frieze and Páll Melsted. Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hash Tables. *Random Struct. Algorithms*, 41(3):334–364, 2012. doi:10.1002/rsa.20427.

15 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast Scalable Construction of (Minimal Perfect Hash) Functions. In *Proc. 15th SEA*, pages 339–352, 2016. doi:10.1007/978-3-319-38851-9_23.

16 Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proc. 18st STACS*, pages 317–326, 2001. doi:10.1007/3-540-44693-1_28.

17 Brian A. LaMacchia and Andrew M. Odlyzko. Solving large sparse linear systems over finite fields. In *CRYPTO '90, 10th Annual International Cryptology Conference*, pages 109–133, 1990. doi:10.1007/3-540-38424-3_8.

18 Bohdan S. Majewski, Nicholas C. Wormald, George Havas, and Zbigniew J. Czech. A Family of Perfect Hashing Methods. *Comput. J.*, pages 547–554, 1996. doi:10.1093/comjnl/39.6.547.

19 Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In *Proc. 15th SODA*, pages 672–681, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982896.

20 Boris Pittel and Gregory B. Sorkin. The Satisfiability Threshold for k-XORSAT. *Combinatorics, Probability & Computing*, 25(2):236–268, 2016. doi:10.1017/S0963548315000097.

21 Ely Porat. An Optimal Bloom Filter Replacement Based on Matrix Solving. In *Proc. 4th CSR*, pages 263–273, 2009. doi:10.1007/978-3-642-03351-3_25.

22 Michael Rink. Mixed Hypergraphs for Linear-Time Construction of Denser Hashing-Based Data Structures. In *Proc. 39th SOFSEM*, pages 356–368, 2013. doi:10.1007/978-3-642-35843-2_31.

23 Steven S. Seiden and Daniel S. Hirschberg. Finding succinct ordered minimal perfect hash functions. *Inf. Process. Lett.*, pages 283–288, 1994. doi:10.1016/0020-0190(94)00108-1.

24 Douglas H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. *IEEE Transactions on Information Theory*, pages 54–62, 1986. doi:10.1109/TIT.1986.1057137.

Complexity of the Steiner Network Problem with Respect to the Number of Terminals

Eduard Eiben
Department of Informatics, University of Bergen, Bergen, Norway
eduard.eiben@uib.no

Dušan Knop
Algorithmics and Computational Complexity, Faculty IV, TU Berlin
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
dusan.knop@tu-berlin.de

Fahad Panolan
Department of Informatics, University of Bergen, Bergen, Norway
fahad.panolan@uib.no

Ondřej Suchý
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
ondrej.suchy@fit.cvut.cz

Abstract

In the Directed Steiner Network problem we are given an arc-weighted digraph G, a set of terminals $T \subseteq V(G)$ with $|T| = q$, and an (unweighted) directed request graph R with $V(R) = T$. Our task is to output a subgraph $H \subseteq G$ of the minimum cost such that there is a directed path from s to t in H for all $st \in A(R)$.

It is known that the problem can be solved in time $|V(G)|^{O(|A(R)|)}$ [Feldman&Ruhl, SIAM J. Comput. 2006] and cannot be solved in time $|V(G)|^{o(|A(R)|)}$ even if G is planar, unless the Exponential-Time Hypothesis (ETH) fails [Chitnis et al., SODA 2014]. However, the reduction (and other reductions showing hardness of the problem) only shows that the problem cannot be solved in time $|V(G)|^{o(q)}$, unless ETH fails. Therefore, there is a significant gap in the complexity with respect to q in the exponent.

We show that Directed Steiner Network is solvable in time $f(q) \cdot |V(G)|^{O(c_g \cdot q)}$, where c_g is a constant depending solely on the genus of G and f is a computable function. We complement this result by showing that there is no $f(q) \cdot |V(G)|^{o(q^2/\log q)}$ algorithm for any function f for the problem on general graphs, unless ETH fails.

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

Keywords and phrases Directed Steiner Network, Planar Graphs, Parameterized Algorithms, Bounded Genus, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.25

Related Version `https://arxiv.org/abs/1802.08189`

Funding *Eduard Eiben*: Eduard Eiben was supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744).
Dušan Knop: Partially supported by DFG under project "MaMu", NI 369/19.
Ondřej Suchý: Supported by grant 17-20065S of the Czech Science Foundation.

1 Introduction

Steiner Tree is one of the most fundamental and well studied problems in combinatorial optimization. The input of Steiner Tree is an edge-weighted undirected graph G and a set $T \subseteq V(G)$ of terminals. Here, the task is to find a least cost connected subgraph H of G containing all the terminals. The problem is known to be NP-complete and, in fact, was one of the 21 NP-complete problems in Karp's original list [29]. The problem is known to be APX-complete, even when the input graph is a complete graph and all edge weights are 1 or 2 [1]. On the other hand, the problem admits a constant factor approximation algorithm and the current best approximation ratio is less than 1.39 [3]. For an overview of the results and applications of Steiner Tree, the reader is referred to monographs [8, 27, 35].

Steiner Tree is well studied in parameterized complexity. The most natural parameter for the problem is the number of terminals q. The first FPT-algorithm for the problem is the $O\big(3^q \cdot n + 2^q \cdot n^2 + n(n \log n + m)\big)$-time algorithm of Dreyfus and Wagner [14] (independently found by Levin [30]) from 1970s; here and on n denotes $|V(G)|$ and m denotes $|E(G)|$. This algorithm, as well as its later improvements [16, 22, 2] subsequently approaching the $O(2^q \operatorname{poly}(n+m))$ running time, uses exponential space. The running time of $O(2^q \operatorname{poly}(n+m))$ is optimal assuming Set Cover Conjecture [9]. There have been many studies for designing algorithms with lower space complexity. Polynomial space FPT-algorithms appeared only recently: First by Nederlof [34] for weights bounded by a constant and later by Fomin et al. [20] for arbitrary weights.

Steiner Tree can be generalized to digraphs. There are many variants of Steiner-type problems on digraphs; the two most natural are Directed Steiner Tree (DST) and Strongly Connected Steiner Subgraph (SCSS). In DST, we are given an arc-weighted directed graph G, a set $T \subseteq V(G)$ of q terminals, and a root vertex $r \in V(G)$. Our task is to find a least cost subgraph H of G such that for every $t \in T$, t is reachable from r in H. In SCSS, the input is an arc-weighted directed graph G and a set $T \subseteq V(G)$ of terminals. The task is to find a least cost subgraph H of G such that for every $s, t \in T$, there are directed paths from s to t and from t to s in H. That is, H is a least cost strongly connected subgraph containing all the terminals. A common generalization of DST and SCSS is Directed Steiner Network (DSN). In DSN, we are given an arc-weighted digraph G, a set $T \subseteq V(G)$ of q terminals, and a digraph R on T. The task is to find a least cost subgraph H of G which realizes all paths prescribed by the arcs of R. That is, for every arc $st \in A(R)$, there is a directed path from s to t in H. Observe that, in DSN, request graphs R and R' yield the same set of solutions if their transitive closures are the same. DST is a special case of DSN where R is a single out-tree on $T \cup \{r\}$ with r being the root and T being the set of leaves. Similarly, SCSS is a special case of DSN where R is a single directed cycle on T.

Existence of an α-approximation algorithm for DST implies a 2α-approximation algorithm for SCSS because of the following simple observation. The union of an in-tree and an out-tree from one fixed terminal in T yields a strongly connected subgraph containing T. The best known approximation ratio in polynomial time for DST and SCSS is $O(q^\varepsilon)$ for any $\varepsilon > 0$ [4]. The same paper also yields an $O(\log^2 q)$-approximation algorithm in quasi-polynomial time. A result of Halperin and Krauthgamer [26] implies that DST and SCSS have no $O(\log^{2-\varepsilon} n)$-approximation for any $\varepsilon > 0$, unless NP has quasi-polynomial time Las Vegas algorithms. The best known approximation algorithm for DSN is by Chekuri et al. [5] with an approximation factor of $O(|A(R)|^{1/2+\epsilon})$ for any $\epsilon > 0$. On the other hand, DSN cannot be approximated to within a factor of $O(2^{\log^{1-\epsilon} n})$ for any fixed $\epsilon > 0$, unless NP $\subseteq$ TIME$(2^{\operatorname{polylog}(n)})$ [13].

Recently Dinur and Manurangsi [12] showed that, under ETH, no polynomial time algorithm and, under Gap-ETH, even no algorithm parameterized by q can approximate DSN to within a factor of $O(|A(R)|^{1/4-o(1)})$.

Using essentially the same techniques as for Steiner Tree [14], one can show that there is an $O\big(3^q \cdot n + 2^q \cdot n^2 + n(n \log n + m)\big)$ time algorithm for DST. On the other hand, Guo et al. [25] showed that SCSS parameterized by q is W[1]-hard. That is, there is no $f(q) \cdot n^{O(1)}$ time algorithm for SCSS for any function f, unless FPT=W[1]. Later a stronger lower bound has been shown by Chitnis et al. [7]. They showed that, in fact, there is no $f(q)n^{o(q/\log q)}$ algorithm for SCSS for any function f, unless Exponential Time Hypothesis (ETH) of Impagliazzo and Paturi [28] fails. This stimulated the research on DSN for restricted classes of request graphs [36, 18] and host graphs [6].

As DSN is a generalization of SCSS, DSN is also W[1]-hard parameterized by q. On the positive side, Feldman and Ruhl [17] showed that DSN can be solved in $n^{O(|A(R)|)}$ time. An independent algorithm with a similar running time also follows from the classification work of Feldmann and Marx [18]. Complementing these results Chitnis et al. [7] showed that DSN cannot be solved in $f(q)n^{o(|A(R)|)}$ time for any function f, even when restricting the host graph G to be planar and all arc weights equal to 1, unless ETH fails. In this reduction (as well as in the reduction given for SCSS), the number of arcs of the request graph $|A(R)|$ is only linear in the number of terminals q. Hence, viewed in terms of the number of terminals, this lower bound implies that there is no $f(q)n^{o(q)}$ time algorithm for any function f, unless ETH fails. But both the known algorithms have running time $n^{\Theta(q^2)}$ in the worst case, leaving a significant gap between the upper and the lower bound for DSN. In this work we contribute to fill this gap.

1.1 Our Results

▶ **Theorem 1.1.** *There is an algorithm which solves any instance (G, R) of DSN in time $2^{c_g q^2 \log(c_g q)} \cdot n^{O(c_g \cdot q)}$, where g is the Euler genus of the graph G and $c_g = 20^{8g+12}g$.*

The main idea behind the algorithm is as follows. Let H be a least cost subgraph of G which realizes all paths prescribed by the arcs of R (call it *an optimum solution*). By the result of Feldmann and Marx [18], if the treewidth[1] of H is ω, then there is an algorithm for solving DSN running in time $2^{O(q \cdot \omega \log \omega)} \cdot n^{O(\omega)}$.[2] Towards proving Theorem 1.1 we construct a graph H' from H such that

- the genus of H' is at most g (recall that g is the genus of the input graph G),
- H' and H have the same grid minors and hence $\mathrm{tw}(H) \leq 20^{4 \cdot (2g+3)} \mathrm{tw}(H')$, and
- the diameter of H' is $O(q)$.

Finally, since H' has genus g and diameter $O(q)$, it follows from a result of Eppstein [15] that $\mathrm{tw}(H') = O(g \cdot q)$. We conclude that H has treewidth $O(c_g \cdot q)$ and our result follows using the algorithm of Feldmann and Marx [18].

We complement the above positive result by the following negative one for general graphs.

▶ **Theorem 1.2.** *There is no $f(q) \cdot n^{o(q^2/\log q)}$ time algorithm for DSN on general graphs for any function f, unless ETH fails.*

Towards this result, we give a reduction from Partitioned Subgraph Isomorphism (PSI).

[1] Since H is a directed graph, we have to clarify that by the treewidth of a directed graph we mean the treewidth of the underlying undirected graph of H (that is, in a graph on the same vertex set that contains an edge $\{u, v\}$ if and only if H contained an arc (u, v)).

[2] The exact running time bound is more complicated, see Proposition 2.1 for exact statement.

2 Preliminaries

For a positive integer η, we use $[\eta]$ to denote the set $\{1, \ldots, \eta\}$. We consider simple directed graphs and use mostly standard notation that can be found for example in the textbook by Diestel [11]. Let G be a directed graph and let $V(G)$ and $A(G)$ denote its vertex set and arc set, respectively. For vertices $u, v \in V(G)$ the arc from u to v is denoted by uv or (u, v). A *walk* $P = (p_0, \ldots, p_\ell)$ of length ℓ in G is a tuple of vertices, that is, $p_i \in V(G)$ for all $0 \leq i \leq \ell$, such that $p_i p_{i+1} \in A(G)$ for all $0 \leq i < \ell$. A *directed path* $P = (p_0, \ldots, p_\ell)$ in G is a walk of length ℓ with all vertices distinct, that is $p_i \neq p_j$ for all $0 \leq i < j \leq \ell$. We let $V(P) = \{p_0, \ldots, p_\ell\}$. We say that the path P is from p_0 to p_ℓ; we call p_0 and p_ℓ the *endpoints* of P while the other vertices of P are called *internal* (we denote the set of all internal vertices of P by $\mathring{P}$). Path P is between u and v if it is either from u to v or from v to u. Let W be a set of vertices, we say that a path Q is a *W-avoiding path* if $\mathring{Q} \cap W = \emptyset$; if P is a path we say that Q is *P-avoiding path* if it is a $V(P)$-avoiding path. Let P be a walk from u to v and let Q be a walk from v to w. By $P \circ Q$ we denote the *concatenation* of P and Q, that is, the walk from u to w that follows P from u to v and then follows Q from v to w. Let $P = (p_0, \ldots, p_\ell)$ be a directed path and $u, v \in V(P)$. We write $u \leq_P v$ if u is before v on P, in other words, $u = p_i$ and $v = p_j$ such that $i \leq j$. Furthermore, for $i \leq j$ the *subpath of P between p_i and p_j*, denoted $p_i[P]p_j$, is the path $(p_i, \ldots, p_j)$.

For a vertex $v \in V(G)$ its *in-degree* is defined as $\deg^-_G(v) = |\{u \in V \mid uv \in A(G)\}|$. The *out-degree* of v is $\deg^+_G(v) = |\{u \in V \mid vu \in A(G)\}|$. Finally, the *total degree* of v is $\deg_G(v) = \deg^-_G(v) + \deg^+_G(v)$. If the graph G is clear from the context we drop the subscript G. We use $\mathrm{sym}(G)$ to denote the underlying undirected graph of a directed graph G. To *subdivide an arc $e \in A(G)$* is to delete $e = uv$, add a new vertex w, and add the arcs uw, wv. We say that H is a *subdivision* of G if it can be obtained by repeated subdivision of arcs of G, that is, there exist graphs $G = G_0, \ldots, G_\eta = H$ such that G_{i+1} is the result of arc subdivision in G_i.

We consider the following problem:

Directed Steiner Network (DSN)	
Input:	An arc-weighted directed graph G and an (unweighted) directed graph R such that $V(R) \subseteq V(G)$.
Question:	Find a minimum-cost subgraph H of G in which there is a path from s to t for every $st \in A(R)$.

The problem is also called Directed Steiner Forest or Point-to-Point Connection. We only consider positive weights on arcs, since it is possible to include all non-positive weight arcs into the solution. We call a subgraph H of G *a solution* to the instance (G, R) of DSN if H contains a path from s to t for every $st \in A(R)$. Moreover, we say that H is an *inclusion-minimal* solution to R, if H is a solution for some instance (G, R), but for every $e \in A(H)$, $H - e$ is not. Note that an optimum solution (one with the least sum of weights) is necessarily inclusion-minimal, as we assume positive weights.

▶ **Proposition 2.1** (Feldmann and Marx [18, Theorem 5] (see also [19])). *Let an instance of DSN be given by a graph G with n vertices and a pattern R on q terminals with vertex cover number τ. If the optimum solution to R in G has treewidth ω, then the optimum can be computed in time $2^{O(q+\tau\omega\log\omega)}n^{O(\omega)}$.*

▶ **Proposition 2.2** (Demaine, Hajiaghayi, and Kawarabayashi [10]). *Suppose G is a graph with no $K_{3,k}$-minor. If the treewidth of G is at least $20^{4k}r$, then G has an $r \times r$ grid minor.*

For the rest of the paper, by the genus of a graph we always mean *Euler genus*; that is the minimum integer g such that the graph can be drawn without crossing itself on a sphere with g cross-caps or with $g/2$ handles. For a more detailed treatment of topological graph theory the reader is referred to [33] or [24].

▶ **Proposition 2.3** (Ringel, see [33, Theorem 4.4.7]). *If G has Euler genus at most g, then G does not contain $K_{3,2g+3}$ as a minor.*

▶ **Proposition 2.4** (Eppstein [15, Theorem 2]). *Let G be a graph of Euler genus[3] g and diameter D. Then G has treewidth $O(gD)$.*

t-Boundaried Graphs and Gluing. A *t-boundaried graph* is a graph G and a set $B \subseteq V(G)$ of size at most t with each vertex $v \in B$ having a label $G(v) \in \{1, \ldots, t\}$. Each vertex in B has a unique label. We refer to B as the boundary of G. For a t-boundaried graph G the function $\delta(G)$ returns the boundary of G. Two t-boundaried graphs G_1 and G_2 can be *glued* together to form a graph $G = G_1 \oplus G_2$. The gluing operation takes the disjoint union of G_1 and G_2 and identifies the vertices of $\delta(G_1)$ and $\delta(G_2)$ with the same label.

A t-boundaried graph H is a minor of a t-boundaried graph G if (a t-boundaried graph isomorphic to) H can be obtained from G by deleting vertices or edges or contracting edges, but never contracting edges with both endpoints being boundary vertices[4]. For more details see e.g. [21].

Monadic Second Order Logic. The syntax of Monadic second order logic (MSO) includes the logical connectives $\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall, \exists$ that can be applied to these variables, and the following five binary relations:

1. $u \in U$ where u is a vertex variable and U is a vertex set variable;
2. $d \in D$ where d is an edge variable and D is an edge set variable;
3. $\mathrm{inc}(d, u)$, where d is an edge variable, u is a vertex variable; and the interpretation is that the edge d is incident on the vertex u;
4. $\mathrm{adj}(u, v)$, where u and v are vertex variables, and the interpretation is that u and v are adjacent;
5. equality of variables representing vertices, edges, set of vertices and set of edges.

Many common graph-theoretic notions such as vertex degree, connectivity, planarity, outerplanarity, being acyclic, and so on, can be expressed in MSO, as can be seen from introductory expositions [31].

3 Solving DSN on a Fixed Surface

Fix an instance (G, R) of DSN. Let the genus of G be a fixed constant g and let H be an inclusion-minimal solution to (G, R). Note that, since H is a subgraph of G, the genus of H is at most g.

The goal of this section is to show the following theorem.

[3] The original paper of Eppstein states genus instead of Euler genus; however, the proof works for both orientable and non-orientable genus and hence also for Euler genus.
[4] Note that these operations preserve the labeling of the boundary vertices.

▶ **Theorem 3.1.** *Let g be a fixed constant. If (G, R) is an instance of DSN such that the genus of G is at most g and H is an inclusion-minimal solution to (G, R), then the treewidth of H is $O\big(20^{4(2g+3)} g \cdot q\big)$.*

With this theorem at hand Theorem 1.1 follows from Proposition 2.1. Note that we can treat every connected component of H separately. More precisely, for each connected component H_C of H, we apply the rest of the proof to H_C and $R[T \cap V(H_C)]$. Hence, we assume that H is connected.

Reversing Arcs – Symmetry. Let $\overleftarrow{G}$, $\overleftarrow{H}$, and $\overleftarrow{R}$ be the directed graphs we obtain from G, H, and R, respectively, by reversing all the arcs. That is, for example, $\overleftarrow{G}$ contains an arc uv if and only if G contains the arc vu. Note that there is a one-to-one correspondence between an s-t path in H and a t-s path in $\overleftarrow{H}$. Hence, if H is an optimum solution to the instance (G, R), then $\overleftarrow{H}$ is an optimum solution to the instance $\big(\overleftarrow{G}, \overleftarrow{R}\big)$. The importance of $\overleftarrow{G}, \overleftarrow{H}, \overleftarrow{R}$ is that every lemma holds in both H, R and $\overleftarrow{H}, \overleftarrow{R}$. In this way we obtain symmetric results without reproving everything twice.

▶ **Lemma 3.2.** *Let (G, R) be an instance of DSN, H be an inclusion-minimal solution to (G, R), and let H be connected. Let R' be a directed graph with vertex set T and for every $s, t \in T$ with $s \neq t$ satisfying $st \in A(R')$ if and only if there is a T-avoiding s-t path in H. Then the following holds:*

1. *H is an inclusion-minimal solution to R' and*
2. *$\mathrm{sym}(R')$ is connected.*

Proof. Assume for the contradiction that $\mathrm{sym}(R')$ is not connected and let R_1 be a connected component of R'. Note that since H is inclusion-minimal every vertex of H lies on some s-t path with $s, t \in T$ and $st \in A(R')$. Now let V_1 be the set of vertices that lie on some s-t path in H for $s, t \in V(R_1)$ and $V_2 = V(H) \setminus V_1$. Clearly, $T \setminus V(R_1) \subseteq V_2$, hence V_2 is not empty. Otherwise, by the definition of R', R' would contain an arc between a vertex in $V(R_1)$ and $V(R') \setminus V(R_1)$. Moreover, every vertex in V_2 lies on some terminal-to-terminal path for two terminals in $T \setminus V(R_1)$. Now let $u \in V_1$ and $v \in V_2$. Clearly, u lies on some s_1-t_1 path between two terminals in R_1 and v lies on a s_2-t_2 path between two terminals in $T \setminus V(R_1)$. Since R' does not contain arcs s_1t_2 nor s_2t_1, it follows that there is no arc between u and v. Since this is true for any two vertices $u \in V_1$ and $v \in V_2$ it follows that $H[V_1]$ is a connected component of H, which contradicts the assumption that H is connected. ◀

From now on we replace R with R'.

▶ **Definition 3.3.** *Let H_1, H_2 be two directed graphs. We say that the pair (H_1, H_2) is a* c-admissible pair *if the genus of H_2 is at most the genus of H_1 and $\mathrm{tw}(H_1) \leq c \cdot \mathrm{tw}(H_2)$.*

Overview of the Proof of Theorem 3.1. We transform the solution graph H into a graph H' containing all terminals and preserving all terminal-to-terminal connections such that (H, H') is an c-admissible pair for some constant c and H' has bounded diameter (and thus by Proposition 2.4 has bounded treewidth). We do this by exploiting that a terminal-to-terminal path in H contains only $O(q)$, so called, important and marked vertices. Furthermore, a subpart of the solution "between" two consecutive marked or important vertices has constant treewidth and contains few vertices with arcs to the rest of the solution H. This allows us to reduce this part of the solution to constant size while preserving genus and all terminal-to-terminal connections. Thus, obtaining the graph H' of bounded diameter.

The following lemma shows that we can assume that each non-terminal vertex in H has at least 3 neighbors.

▶ **Lemma 3.4.** *Let (G, R) be an instance of DSN, H be an inclusion-minimal solution to (G, R), and let H be connected. There exists a directed graph $H_{\geq 3}$ such that $H_{\geq 3}$ is an inclusion-minimal solution to R, $H_{\geq 3}$ is connected, every non-terminal vertex in $H_{\geq 3}$ has at least three neighbors, and $(H, H_{\geq 3})$ is a 1-admissible pair. Moreover, for any $s, t \in T$, there is a T-avoiding s-t path in H if and only if there is one in $H_{\geq 3}$.*

Proof Sketch. We exhaustively repeat the following. Let v be a non-terminal vertex and suppose u, w are the only two neighbors of v. Note that v cannot have only one neighbor, since H is an inclusion-minimal solution. We delete v from H and add an arc uw if both uv and vw were in H, similarly for an arc wu. Denote the resulting graph $H_{\geq 3}$. ◀

3.1 Important and Marked Vertices

For a fixed T-avoiding directed path P in H between two terminals s and t, we say that a vertex $u \in V(P)$ is *important with respect to P* if there is a P-avoiding directed path from some terminal not on P to u or from u to some terminal not on P. Let I_P denote the set of all important vertices with respect to P. Let I be the union of important vertices over all T-avoiding paths in H between terminals.

Let $s, t \in T$ and $P = (s = p_1, \ldots, p_r = t)$ be fixed for the rest of this subsection.

▶ **Lemma 3.5.** *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R). Let $P = (s = p_1, \ldots, p_r = t)$ be a T-avoiding directed path between $s, t \in T$. There are at most $2q - 2$ important vertices on P. Moreover, there exists a function $g_P \colon I_P \to T$ with $\left|g_P^{-1}(x)\right| \leq 2$ for every $x \in T$ such that for every $v \in I_P$ there is either v-$g(v)$ or $g(v)$-v directed $(V(P) \cup T)$-avoiding path.*

Proof. We bound the number of important vertices by inspecting the interaction between the path P and other paths in the solution H. In order to do this, we construct a partial labeling $\mathcal{L} \colon V(P) \to 2^{(T \times \{\leftarrow, \rightarrow\})}$ as follows. For a vertex $v \in V(P)$ we have $(x, \leftarrow) \in \mathcal{L}(v)$ if there is a directed P-avoiding path from a terminal x to v in H and v is the closest to s among all such vertices of P. Similarly, we have $(x, \rightarrow) \in \mathcal{L}(v)$ if there is a directed P-avoiding path from v to a terminal x in H and v is the closest to t among all such vertices of P.

▷ Claim 3.6 ($\star$[5]). Every important vertex received some label.

It follows from the above claim that the number of important vertices is bounded by the possible number of labels which is $2q - 2$. This is because by the definition of the labeling every label in $T \times \{\leftarrow, \rightarrow\}$ is used at most once and $(s, \leftarrow)$ and $(t, \rightarrow)$ labels are never assigned to any vertex of P (as they would be assigned to s and t, respectively).

As to the moreover part, it follows from the labeling procedure that if $(\leftarrow, x) \in \mathcal{L}(v)$, then there is a P-avoiding path Q in H from x to v. If this path contains another terminal, then let y be the terminal closest to v on Q. We claim that $(\leftarrow, y) \in \mathcal{L}(v)$ as well. If not, then there would be another vertex v' on P with this label with $v' <_P v$ and a P-avoiding path Q' from y to v'. But then $x[Q]y \circ y[Q']v'$ is a walk that can be shortened to a P-avoiding path from x to v', contradicting $(x, \leftarrow) \in \mathcal{L}(v)$. Hence, each important vertex has a $(V(P) \cup T)$-avoiding path to or from some terminal, such that it has a label of that terminal. To prove the moreover part it remains to set $g_P(v)$ to any such terminal. ◀

[5] Proofs of claims and lemmata marked with ($\star$) were deferred to the full version of the paper.

▶ **Lemma 3.7** (⋆). *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R). Let $P = (s = p_1, \ldots, p_r = t)$ be a T-avoiding directed path between $s, t \in T$. If v is a vertex in $V(P) \setminus I_P$, then its out-degree is at most 2. Moreover, if u is its out-neighbor not on P, then there is a P-avoiding path from u to some vertex $v' \in V(P)$ with $v' <_P v$.*

The following expresses that in order to bound the diameter of H' it is enough to bound the length of the path P linearly in $|I_P|$.

▶ **Lemma 3.8.** *Let (G, R) be an instance of DSN, H be an inclusion-minimal solution to (G, R), and let H be connected. Moreover, assume that for every $s, t \in T$ with $s \neq t$ that $st \in A(R)$ if and only if there is a T-avoiding s-t path in H. If for every $\overline{st} \in A(R)$ there is a T-avoiding path $\tilde{P}$ in H of length at most $c \cdot |I_{\tilde{P}}|$, for some constant c, then the distance between any two terminal vertices in the underlying undirected graph $\mathrm{sym}(H)$ of H is at most $8cq$.*

Proof. By assumption and Lemma 3.2 both H and R are connected. Let $t_1, t_2 \in T$ be two arbitrary terminal vertices and let $Q = (t_1 = t^1, t^2, \ldots, t^\ell = t_2)$ be a shortest path from t_1 to t_2 in $\mathrm{sym}(R)$. Now let $\mathcal{Q} = (Q_1, \ldots, Q_{\ell-1})$ be a realization of the path Q in H, that is, Q_i is a directed T-avoiding path between t^i and t^{i+1} of length at most $c \cdot |I_{Q_i}|$ in H for every $1 \leq i \leq \ell - 1$. Note that it does not matter whether Q_i is a directed path from t^i to t^{i+1} or vice versa.

For $1 \leq i \leq \ell - 1$ let g_i be the function g_{Q_i} for the path Q_i from Lemma 3.5. Let $v \in I_{Q_i}$ be an important vertex on Q_i. From Lemma 3.5 it follows that there is a $(V(Q_i) \cup T)$-avoiding directed path either from v to $g_i(v)$ or from $g_i(v)$ to v. Moreover, since Q_i is T-avoiding, there are two T-avoiding directed paths in H either one from t^i to v and the other from v to t^{i+1} or one from t^{i+1} to v and the other from v to t^i. Therefore, it follows that if a terminal t' is in $g_i(I_{Q_i})$, then there is a T-avoiding directed path either between t' and t^i or between t' and t^{i+1} in H and consequently, by our assumptions on R, there is an arc between t' and either t^i or t^{i+1} in R.

Now, for a terminal t', let $1 \leq i < j \leq \ell - 1$ be such that $t' \in \big(g_i(I_{Q_i}) \cap g_j(I_{Q_j})\big)$. Then we claim that $j - i \leq 3$. From the argument above, it follows that there is an edge between t' and t^i or t^{i+1} and between t' and t^j or t^{j+1} in $\mathrm{sym}(R)$. However, if $j - i \geq 4$, then we can obtain a shorter path than Q in $\mathrm{sym}(R)$ from t_1 to t_2 by going along Q from t_1 to t^i or to t^{i+1}, then using the aforementioned edges to t' and from t' to t^j or t^{j+1} and continuing on Q. This is a contradiction with the choice of Q. Therefore, for each terminal t' there are at most 4 paths $\bar{Q} \in \mathcal{Q}$ such that $t' \in g_{\bar{Q}}(I_{\bar{Q}})$. Since for each path $\bar{Q}$ and terminal t', it holds that $\left|g_{\bar{Q}}^{-1}(t')\right| \leq 2$, it follows that $\sum_{i=1}^{\ell-1} |I_{Q_i}| \leq 2 \cdot 4 \cdot q$. Therefore, the distance between t_1 and t_2 is at most $\sum_{i=1}^{\ell-1} |Q_i| \leq \sum_{i=1}^{\ell-1} c \cdot |I_{Q_i}| \leq 8cq$ and the lemma follows. ◀

▶ **Lemma 3.9.** *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R). Let $P = (s = p_1, \ldots, p_r = t)$ be a T-avoiding directed path between $s, t \in T$. Let p_i, p_j, p_k be three vertices on P such that*

(1) *$i < j < k$,*
(2) *there is a path Q from p_k to p_i that avoids p_j, and*
(3) *every directed path P' from some terminal s' to p_j in H intersects P in a vertex p_ℓ such that $p_\ell \neq p_j$ and $\ell \leq k$.*

Then p_j has no in-neighbor other than p_{j-1} in H.

Proof. Refer to Fig. 1. Let $u \neq p_{j-1}$ be an in-neighbor of p_j. Let $s't'$ be an arc in R such that the arc up_j is on a path P' from s' to t' in H. We show that there is a directed path from s' to t' in $H - up_j$. By our assumption, it follows that $s'[P']p_j$ intersects P in a vertex

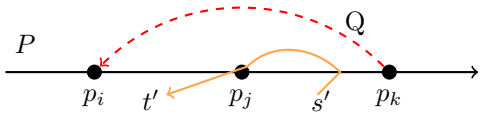


Figure 1 The three vertices p_i, p_j, p_k on a directed path P as in Lemma 3.9. The orange (light gray) path cannot exist as it is rerouted via p_k and Q (dashed); contradicting the minimality of the solution.

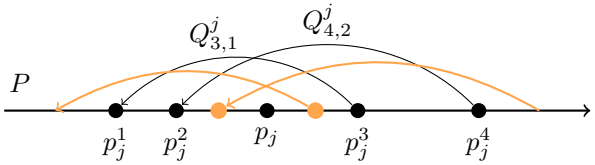


Figure 2 The four vertices on P for the vertex p_j. By the choice of p^1_j and p^4_j, the orange (light gray) paths cannot exist.

p_ℓ such that $\ell < k$. Therefore, the walk $s'[P']p_\ell \circ p_\ell[P]p_k \circ p_k[Q]p_i \circ p_i[P]p_j \circ p_j[P']t'$ induces a directed path from s' to t' in $H - up_j$. Since this is true for every pair of terminals s', t' with an s'-t' path in H, it contradicts the inclusion-minimality of H and hence the only in-neighbor of p_j is p_{j-1}. ◀

For a vertex $p_j \in V(P)$ let $p^1_j, p^2_j, p^3_j, p^4_j$ denote the following four vertices (see Fig. 2):

- p^1_j is the $\leq_P$-minimal vertex on P such that there is a P-avoiding path from a vertex p_x, with $x \geq j$ to p^1_j,
- p^3_j is a vertex such that $p_j \leq_P p^3_j$ and p^3_j is the first vertex of some P-avoiding path to p^1_j,
- p^4_j is the $\leq_P$-maximal vertex on P such that there is a P-avoiding path from p^4_j, to some vertex p_y with $y \leq j$, and
- p^2_j is a vertex such that $p^2_j \leq_P p_j$ and p^2_j is the last vertex of some P-avoiding path from p^4_j.

Furthermore, let us denote $Q^j_{3,1}$ and $Q^j_{4,2}$ the P-avoiding paths from p^3_j to p^1_j and from p^4_j to p^2_j, respectively, and let $Q^j_{4,1}$ denote the path $Q^j_{4,2} \circ p^2_j[P]p^3_j \circ Q^j_{3,1}$.

▶ **Lemma 3.10** (⋆). *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R) such that every non-terminal vertex in H has at least three neighbors. Let $P = (s = p_1, \ldots, p_r = t)$ be a T-avoiding directed path between $s, t \in T$. For every p_j there are at most two vertices in $V(P) \setminus (I_P \cup \{p^2_j, p^3_j\})$ between p^1_j and p^4_j.*

For the rest of this section, let us define the set $Q_P = \{p^1_j, p^2_j, p^3_j, p^4_j \mid p_j \in I_P\}$. Clearly, $|Q_P| \leq 4\,|I_P|$. We will call the set Q_P the *set of marked vertices for P*. Note that the same vertex in Q_P may be marked for different reasons at the same time. That is, for example, the same vertex can be denoted p^1_j, because it is the first vertex for the important vertex p_j and at the same time it can be denoted p^3_k, because it is also the third marked vertex for the important vertex p_k with respect to P.

3.2 Ladders

In this subsection we define ladder graphs. These graphs play crucial a role as we will be able to show that if there is a T-avoiding s-t path for $st \in A(R)$ that is "long", then in H there is a "large" ladder (Lemma 3.13). Moreover, it is possible to replace such a ladder with one having constant size while preserving all connections and inclusion-minimality (Lemma 3.14).

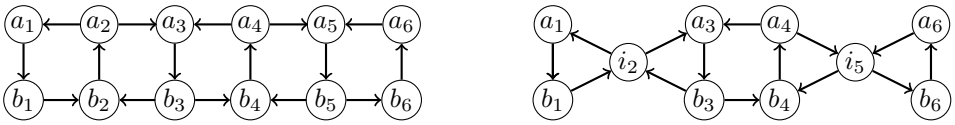


Figure 3 Example ladder graphs. The ladder $G_6 = G_{6,\emptyset}$ on the left and $G_{6,\{2,5\}}$ on the right.

▶ **Definition 3.11** (Class of Ladders). *Let η be a positive integer and $I \subseteq [\eta]$. We define the directed graph G_η and the directed graph $G_{\eta,I}$ as follows (see Fig. 3). The vertex set $V(G_\eta)$ is the set $\{a_i, b_i \mid i \in [\eta]\}$ and the arc set $A(G_\eta)$ is the set*

$$\{a_{2i+1}b_{2i+1} \mid 0 \leq i < \eta/2\} \cup \{b_{2i}a_{2i} \mid 1 \leq i \leq \eta/2\} \cup \{a_{2i}a_{2i-1} \mid 1 \leq i \leq \eta/2\} \cup$$
$$\{a_{2i}a_{2i+1} \mid 1 \leq i < \eta/2\} \cup \{b_{2i+1}b_{2i} \mid 1 \leq i < \eta/2\} \cup \{b_{2i-1}b_{2i} \mid 1 \leq i \leq \eta/2\}.$$

The graph $G_{\eta,I}$ is the graph G_η where we identify the vertices a_i and b_i whenever $i \in I$ (i.e., G_η and $G_{\eta,\emptyset}$ is the same graph). We emphasize that we suppress any loops in $G_{\eta,I}$. We say that η is the length *of the ladder $G_{\eta,I}$.*

▶ **Lemma 3.12** (⋆). *Given a positive integer η and $I \subseteq [\eta]$, the ladder $G_{\eta,I}$ is a union of two paths P_1 from a_1 to a_η and P_2 from b_η to b_1 if η is even or paths P_1 from a_1 to b_η and P_2 from a_η to b_1, if η is odd. Moreover, $G_{\eta,I}$ is an inclusion-minimal strongly connected graph connecting the set of terminals $\{a_1, b_1, a_\eta, b_\eta\}$.*

3.3 Finishing the Proof

Let again P be a T-avoiding directed path in H between two terminals s and t. In the following technical lemma we show that if the distance on P between any two consecutive vertices $p_i, p_j \in Q_P \cup I_P$ with $i < j$ is at least 5, then $p_i = p_k^4$ and $p_j = p_\ell^1$ where $p_k, p_\ell \in I_P$ and $k < \ell$. Moreover, there exists a path from p_j to p_i in H and between p_i and p_j there is a ladder with a constant-sized boundary.

▶ **Lemma 3.13** (⋆). *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R) such that every non-terminal vertex in H has at least three neighbors. Let P be a T-avoiding directed path in H between two terminals s and t. Let $p_i, p_j \in Q_P \cup I_P$ with $i < j$ such that there is no $p \in Q_P \cup I_P$ with $p_i \leq_P p \leq_P p_j$. Let $F = \{p_{i+1}, \ldots, p_{j-1}\}$ and let C be the set of vertices of the connected component of $\mathrm{sym}(H) - \{p_{i+1}, p_{i+2}, p_{j-2}, p_{j-1}\}$ containing p_{i+3}. If $j - i \geq 5$, then $H[C \cup \{p_{i+1}, p_{i+2}, p_{j-2}, p_{j-1}\}]$ is a ladder and furthermore, $p_{i+1}, p_{i+2}, p_{j-2}$, and p_{j-1} are the only vertices with an H-neighbor outside the ladder.*

▶ **Lemma 3.14** (⋆). *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R) such that H is connected and every non-terminal vertex in H has at least three neighbors. Moreover, assume that for every $s, t \in T$ with $s \neq t$ that $st \in A(R)$ if and only if there is a T-avoiding s-t path in H. Let a, b, c, d be four vertices of H and $F \subseteq V(H)$ such that $a = b$ or $ab \in A(H)$, $c = d$ or $cd \in A(H)$, $F \cap T = \emptyset$, $H[F]$ is a connected component of $H - \{a, b, c, d\}$, and $H[F \cup \{a, b, c, d\}]$ is isomorphic to a ladder $G_{\eta,I}$. There exist a directed graph H' and a set $F' \subseteq V(H')$ such that:*
(1) *the genus of H' is at most the genus of H,*
(2) *$H' - F' = H - F$,*
(3) *$|F'| = O(1)$,*
(4) *$N_{H'}(F') = \{a, b, c, d\}$,*

(5) *H' is an inclusion-minimal solution to R,*
(6) *for every $k \geq 10$, if $\mathrm{sym}(H)$ contains $k \times k$ grid as a minor, then $\mathrm{sym}(H')$ contains $k \times k$ grid as a minor,*
(7) *H' is connected,*
(8) *every non-terminal vertex in H' has at least three neighbors, and*
(9) *for every $s, t \in T$ with $s \neq t$ we have $st \in A(R)$ if and only if there is a T-avoiding s-t path in H'.*

Proof sketch. From Lemma 3.12 it follows that $H[F \cup \{a, b, c, d\}]$ is a union of two directed paths P_1 from a to d and P_2 from c to b. We construct F' such that $H'[F' \cup \{a, b, c, d\}]$ is a ladder $G_{\eta', I'}$, where $I' \subseteq \{1, \eta'\}$ and $1 \in I'$ iff $a = b$ and $\eta' \in I'$ iff $c = d$. Even though it is a bit technical, it is rather straightforward to verify that if we replace F by another ladder, then H' will satisfy (5), (7), (8), and (9). If $\mathrm{sym}(H)$ does not contain any $k \times k$ grid for $k \geq 10$, then we just replace F with any constant size ladder and we are fine. Otherwise, we take a largest grid minor K of $\mathrm{sym}(H)$. Since $\mathrm{sym}(H)[F \cup \{a, b, c, d\}]$ has treewidth 2 and only 4 of its vertices have neighbors in the rest of H, one can show that $\mathrm{sym}(H)[F \cup \{a, b, c, d\}]$ contracts to at most ten vertices in K. Let K_F be the graph induced on these ten vertices. It is easy to see that if we replace $H[F \cup \{a, b, c, d\}]$ with any ladder whose underlying undirected graph has K_F as a minor which furthermore maps its boundaries onto K_F in the same way as $\mathrm{sym}(H)[F \cup \{a, b, c, d\}]$, then the underlying undirected graph of the resulting graph contains K as a minor as well. However, one can express by a constant-sized MSO formula that a boundaried graph is a ladder $G_{\eta', \emptyset}$ and has the boundaried graph K_F as a minor. It follows that this formula has a constant-sized model, whose suitable orientation is the sought replacement. ◀

▶ **Lemma 3.15 (⋆).** *Let (G, R) be an instance of DSN and H be an inclusion-minimal solution to (G, R) such that H is connected and every non-terminal vertex in H has at least three neighbors. Moreover, assume that for every $s, t \in T$ with $s \neq t$ that $st \in A(R)$ if and only if there is a T-avoiding s-t path in H. There exists a directed graph H' such that*

- *(H, H') is a $\left(20^{4(2g+3)}\right)$-admissible pair,*
- *$T \subseteq V(H')$,*
- *for all $s, t \in T$, there is a directed s-t path in $H - (T \setminus \{s, t\})$ if and only if there is a directed path from s to t in $H' - (T \setminus \{s, t\})$,*
- *H' is an inclusion-minimal solution to R,*
- *H' is connected,*
- *every non-terminal vertex in H' has at least three neighbors, and*
- *for any arc $st \in A(R)$, there is a T-avoiding directed path P from s to t in H' with length at most $O(|I_P|)$.*

Proof sketch. We obtain H' by recursively applying Lemma 3.14 until there is no ladder with the boundary of size at most 4 that can be shortened by applying Lemma 3.14. By Lemma 3.13 the distance between any two consecutive $p_i, p_j \in Q_P \cup I_P$ is constant. Since the genus of $\mathrm{sym}(H)$ is at most g, it follows from Proposition 2.3 that $\mathrm{sym}(H)$ is $K_{3,2g+3}$-minor-free. Hence, due to Proposition 2.2, the treewidth of $\mathrm{sym}(H)$ is at most $20^{4(2g+3)}\ell$, where ℓ is the size of the largest grid minor of $\mathrm{sym}(H)$ which is the same as of $\mathrm{sym}(H')$ by Lemma 3.14. ◀

Proof of Theorem 3.1. Let H_1 be any connected component of H, $T_1 = V(H_1) \cap T$, and $R_1 = R[T_1]$. By Lemma 3.2, there is R_2 such that for every $s, t \in T_1$ with $s \neq t$ we have $st \in A(R_2)$ if and only if there is a T_1-avoiding s-t path in H_1. By Lemma 3.4, there is a

directed graph H_2, such that H_2 is an inclusion-minimal solution to R_2, H_2 is connected, for every $s, t \in T_1$ with $s \neq t$ we have $st \in A(R_2)$ if and only if there is a T_1-avoiding s-t path in H_2, every non-terminal vertex in H_2 has at least three neighbors in H_2 and the genus of H_2 is at most the genus of H_1. By Lemma 3.15, there exists a directed graph H' such that H' is an inclusion-minimal solution to R_2, H' is connected, $\mathrm{tw}(\mathrm{sym}(H_2)) \leq 20^{4(2g+3)}\mathrm{tw}(\mathrm{sym}(H'))$, and for each arc $st \in A(R_2)$, there is a directed path from s to t of length at most $O(|I_P|)$ in H'. Furthermore, all the vertices of H' are on some path of length at most $O(|I_P|)$ between two terminals in H'. By Lemma 3.8, it follows that there is a path of length at most $O(q)$ between each pair of terminals in $\mathrm{sym}(H')$ and hence the diameter of $\mathrm{sym}(H')$ is also at most $O(q)$. Finally, by Proposition 2.4, it follows that $\mathrm{sym}(H')$ has treewidth $O(g'q)$, where g' is the genus of $\mathrm{sym}(H')$. Since the genus of $\mathrm{sym}(H')$ is at most the genus of $\mathrm{sym}(H_2)$, which in turn is at most the genus of $\mathrm{sym}(H_1)$, which in turn is at most the genus of $\mathrm{sym}(H)$, which is at most g, the genus of G, the theorem follows. ◀

4 Improved ETH-based Lower Bound for General Graphs

Our proof is based on a reduction from (a special case of) the following problem:

Partitioned Subgraph Isomorphism (PSI)	
Input:	Two undirected graphs G and H with $\lvert V(H)\rvert \leq \lvert V(G)\rvert$ (H is *smaller*) and a mapping $\psi\colon V(G) \to V(H)$.
Question:	Is H isomorphic to a subgraph of G? I.e., is there an injective mapping $\phi\colon V(H) \to V(G)$ such that $\{\phi(u), \phi(v)\} \in E(G)$ for each $\{u, v\} \in E(H)$ and $\psi \circ \phi$ is the identity?

▶ **Theorem 4.1** (Marx [32, Corollary 6.1]). *If there exist a recursively enumerable class $\mathcal{H}$ of graphs with unbounded treewidth, an algorithm $\mathbb{A}$, and an arbitrary function f such that $\mathbb{A}$ correctly decides every instance of* Partitioned Subgraph Isomorphism *with the smaller graph H in $\mathcal{H}$ in time $f(H)n^{o(\mathrm{tw}(H)/\log \mathrm{tw}(H))}$, then ETH fails.*

It is known that there are infinitely many 3-regular graphs such that each such graph H has treewidth $\Theta(|V(H)|)$ (see [23, Proposition 1, Theorem 5]). Using the class of 3-regular graphs as $\mathcal{H}$ in the above theorem, we arrive at the following corollary.

▶ **Corollary 4.2.** *If there is an algorithm $\mathbb{A}$ and an arbitrary function f such that $\mathbb{A}$ correctly decides every instance of* Partitioned Subgraph Isomorphism *with the smaller graph H being 3-regular in time $f(|V(H)|)n^{o(|V(H)|/\log |V(H)|)}$, then ETH fails.*

Our plan is to use this corollary. To this end, we transform the (special) instances of PSI to instances of DSN.

▶ **Construction 1.** Let (G, H, ψ) be an instance of PSI with H 3-regular and denote $k = |V(H)|$. Note that then $|E(H)| = O(k)$. We let $r = \lceil\sqrt{k}\rceil$. We first compute labelings $\alpha\colon V(H) \to X$, $\beta\colon V(H) \to Y$, and $\gamma\colon E(H) \to Z$, where $X = \{x_1, \ldots, x_{\max}\}$, $Y = \{y_1, \ldots, y_{\max}\}$, and $Z = \{z_1, \ldots, z_{\max}\}$ are three new sets. We want the sets X, Y, Z to be of size $O(r)$ while fulfilling the following constraints:

(i) $\forall u, v \in V(H) : (\alpha(u) \neq \alpha(v)) \vee (\beta(u) \neq \beta(v))$,
(ii) $\forall \{u, v\} \in E(H) : (\alpha(u) \neq \alpha(v)) \wedge (\beta(u) \neq \beta(v))$,
(iii) $\forall e, f \in E(H), \forall u, v \in V(H) : ((u \in e) \wedge (v \in f) \wedge (\alpha(u) = \alpha(v))) \implies (\gamma(e) \neq \gamma(f))$.

In other words, the pair $(\alpha(u), \beta(u))$ uniquely identifies vertex u, adjacent vertices share no labels and both pairs $(\alpha(u), \gamma(\{u, v\}))$ and $(\alpha(v), \gamma(\{u, v\}))$ uniquely identify edge $\{u, v\}$.

To obtain such labeling, first color the vertices of H greedily with colors $1, \ldots, 4$, denote μ the coloring and $A_1, \ldots, A_4$ the set of vertices of color $1, \ldots, 4$, respectively. For every $i \in [4]$, we split the set A_i into sets $A_{i,1}, \ldots, A_{i,a_i}$ such that for every $j \in [a_i - 1]$ the set $A_{i,j}$ is of the size r and the set A_{i,a_i} is of the size at most r. Since $r = \left\lceil \sqrt{k} \right\rceil$ we know that there will be at most r sets of the size r and, thus, at most $r + 4$ sets in total. We assign to each nonempty set $A_{i,j}$ a unique label x_ℓ and let $\alpha(u) = x_\ell$ for every $u \in A_{i,j}$. Note that $|X| \leq r + 4$.

Next we construct a graph H' from H by turning each $A_{i,j}$ into a clique. Since the degree of each vertex in H is 3 and the size of each $A_{i,j}$ is at most r, the degree of each vertex in H' is at most $r + 2$. Hence we can color the vertices of H' greedily with colors $y_1, \ldots, y_{r+3}$ and we let β be the coloring.

Finally, we construct a multigraph H'' from H' by contracting each clique $A_{i,j}$ to a single vertex. We keep multiple edges between two vertices if they are a result of the contraction, but we remove all loops. Note that the edges preserved are exactly the edges of H. Since the size of each $A_{i,j}$ is at most r and H is 3-regular, the maximum degree (counting the multiplicities of the edges) is at most $3r$. Therefore, the maximum degree in the line graph $L(H'')$ of H'' is at most $6r - 2$. Thus, we can color the edges of H'' greedily with colors $z_1, \ldots, z_{6r-1}$ and let γ be the coloring.

Let us check that the labelings fulfill the constraints. First, if $\alpha(u) = \alpha(v)$, then $\{u, v\} \in E(H')$ and, thus, $\beta(u) \neq \beta(v)$. If $\{u, v\} \in E(H)$, then $\{u, v\} \subseteq A_{i,j}$ would imply that u and v are colored by the same color by μ – a contradiction. Hence, $\alpha(u) \neq \alpha(v)$ and, since $E(H) \subseteq E(H')$, we also have $\beta(u) \neq \beta(v)$. Finally, if $e = \{u, v'\}$, $f = \{u', v\}$, and $\alpha(u) = \alpha(v)$, then the edges e and f share a vertex in H'' and, thus, $\gamma(e) \neq \gamma(f)$.

Note also that the labelings can be obtained in $O(|V(H)|^2)$ time.

Having the labelings at hand, we construct the instance (G', R) of DSN as follows (refer to Figure 4 for an overview of the construction). We let $V(G') = V \cup W \cup X \cup Y \cup Z$, where $V = V(G)$, $W = \{w_{uv} \mid \{u, v\} \in E(G)\}$, and X, Y, Z are the images of α, β, γ as defined previously. We let $T = V(R) = X \cup Y \cup Z$. Note that $q = O(r) = O(\sqrt{k})$. We let $A(G') = A_V \cup A_W$, where $A_V = \big\{\big(\alpha(\psi(u)), u\big), \big(u, \beta(\psi(u))\big) \mid u \in V\big\}$ and $A_W = \big\{(u, w_{uv}), (v, w_{uv}), \big(w_{uv}, \gamma(\{\psi(u), \psi(v)\})\big) \mid \{u, v\} \in E(G)\big\}$. We assign unit weights to all arcs of G'. Finally let $A(R) = A_Y \cup A_Z$, where $A_Y = \{(\alpha(u), \beta(u)) \mid u \in V(H)\}$ and $A_Z = \{(\alpha(u), \gamma(\{u, v\})), (\alpha(v), \gamma(\{u, v\})) \mid \{u, v\} \in E(H)\}$.

Let us stop here to discuss the size of $A(R)$. By Condition (i) on the labelings we have $|A_Y| = |V(H)|$. By Condition (ii) we have $(\alpha(u), \gamma(\{u, v\})) \neq (\alpha(v), \gamma(\{u, v\}))$ for any $\{u, v\} \in E(H)$. Hence, by Condition (iii) the size of A_Z is exactly $2|E(H)|$.

Next, we show that the construction transforms yes-instances of PSI to instances of DSN with bounded value of the optimum.

▶ **Lemma 4.3.** *If there is an injective mapping ϕ forming a solution to the instance (G, H, ψ) of PSI, then there is a subgraph P of G' forming a solution to the instance (G', R) of DSN with cost $|A(P)| \leq 2|V(H)| + 3|E(H)|$.*

Proof. Let ϕ be a solution to the instance (G, H, ψ). Since ϕ is a solution, we know that $\{\phi(u), \phi(v)\} \in E(G)$ whenever $\{u, v\} \in E(H)$. Consider the subgraph $P = G'[V_\phi]$ of G' induced by $V_\phi = X \cup Y \cup Z \cup V' \cup W'$, where $V' = \{\phi(v) \mid v \in V(H)\}$ and $W' = \big\{w_{\phi(u)\phi(v)} \mid \{u, v\} \in E(H)\big\}$. Obviously, $|V'| = |V(H)|$ and $|W'| = |E(H)|$.

Since each arc in A_W is incident to some vertex in W and each vertex in W is incident to exactly 3 such arcs, P contains at most $3|E(H)|$ arcs from A_W. Similarly, since each arc in A_V is incident to some vertex in V and each vertex in V is incident to exactly 2 such arcs, P contains at most $2|V(H)|$ arcs from A_V. Thus, P contains at most $2|V(H)| + 3|E(H)|$ arcs in total.

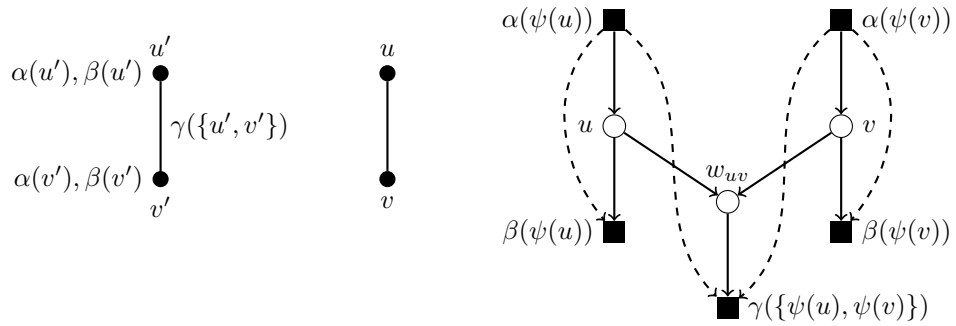


Figure 4 An illustration of Construction 1. Left is a pattern graph H, middle a host graph G, and right the produced graphs G' and R combined. We assume $\psi(u) = u'$ and $\psi(v) = v'$ here. On the right the terminals are depicted by full squares and non-terminals by empty circles. Arcs in G' are drawn solid, while the arcs of R are dashed.

We want to show for each $(s,t) \in A(R)$ that there is a directed path from s to t in P. Indeed, if $(x,y) \in A_Y$, then $x = \alpha(u)$ and $y = \beta(u)$ for some $u \in V(H)$ and $(\alpha(u), \phi(u), \beta(u)) = (\alpha(\psi(\phi(u))), \phi(u), \beta(\psi(\phi(u))))$ is a path of length 2 from x to y in P. If $(x,z) \in A_Z$, then $x = \alpha(u)$ and $z = \gamma(\{u,v\})$ for some $\{u,v\} \in E(H)$ and $(\alpha(u), \phi(u), w_{\phi(u)\phi(v)}, \gamma(\{u,v\}))$ is a path of length 3 from x to z in P. This finishes the proof. ◀

Next we show that the value of the optimum of the instances of DSN produced by the construction can be appropriately bounded only if we started with a yes-instance of PSI.

▶ **Lemma 4.4** (⋆). *If there is a subgraph P of G' forming a solution to the instance (G', R) of DSN with cost $|A(P)| \leq 2|V(H)| + 3|E(H)|$, then there is an injective mapping ϕ forming a solution to the instance (G, H, ψ) of PSI.*

Proof of Theorem 1.2. Let $\mathbb{A}$ be an algorithm that correctly solves DSN (on general graphs) in time $f(q)n^{o(q^2/\log q)}$ for some function f. Let us construct an algorithm $\mathbb{B}$ for PSI with the smaller graph H being 3-regular as follows: Let (G, H, ψ) be an instance of PSI with H 3-regular. We use Construction 1 to build the instance (G', R) of DSN. Then run $\mathbb{A}$ on (G', R) and return yes if and only if the cost of the obtained solution P is $|A(P)| \leq 2|V(H)|+3|E(H)|$. The answer of $\mathbb{B}$ is correct by Lemmata 4.3 and 4.4.

Let us analyze the running time of $\mathbb{B}$. Let us denote $k = |V(H)|$ and $n = |V(G)|$. We may assume that $k \leq n$, as otherwise we can immediately answer no. The labelings can be obtained in $O(k^2)$ time. Graph G has at most $O(n^2)$ edges and the graphs G' and R can be constructed in linear time in the number of vertices and edges of the graphs G and H, respectively. That is, Construction 1 can be performed in $O(n^2)$ time and, in particular, G' has $O(n^2)$ vertices. However, by the construction, the number q of vertices of graph R is $O(\sqrt{k})$. Now, $\mathbb{A}$ runs on (G', R) in time $f(q)|V(G')|^{o(q^2/\log q)} = f'(\sqrt{k})n^{o((\sqrt{k})^2/\log\sqrt{k})} = f''(k)n^{o(k/\log k)}$ for some functions f, f', and f''. But then the whole $\mathbb{B}$ runs in $f''(k)n^{o(k/\log k)}$ time and ETH fails by Corollary 4.2. ◀

5 Conclusions

Our results show that we can solve DSN in time $n^{O(q)}$ when the input directed graph is embeddable on a fixed surface However, for general graphs it is unlikely to obtain even an algorithm running in time $n^{o(q^2/\log(q))}$. It would be interesting to see what happens for the

graph classes that are somewhere in between. For example, it is not difficult to show that the graph H' that we obtain in Section 3 has at most $O(q^3)$ vertices and, hence, the largest grid minor of H' is of size $O(q^{3/2}) \times O(q^{3/2})$. Therefore, with a careful modification of our approach, one can show that there is an $n^{O(q^{3/2})}$ time algorithm for DSN when the input graph excludes a fixed minor. However, it remains open whether the running time $n^{O(q^{3/2})}$ is asymptotically optimal or whether it is possible to design an $n^{O(q)}$ time algorithm for DSN in this case.

References

1 Marshall W. Bern and Paul E. Plassmann. The Steiner Problem with Edge Lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176, 1989. doi:10.1016/0020-0190(89)90039-2.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th ACM Symposium on Theory of Computing, STOC 2007*, pages 67–74. ACM, 2007.

3 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *J. ACM*, 60(1):6:1–6:33, February 2013. doi:10.1145/2432622.2432628.

4 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation Algorithms for Directed Steiner Problems. *Journal of Algorithms*, 33(1):73–91, 1999. doi:10.1006/jagm.1999.1042.

5 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed Steiner network problem. *ACM Trans. Algorithms*, 7(2):18:1–18:17, 2011. doi:10.1145/1921659.1921664.

6 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized Approximation Algorithms for Bidirected Steiner Network Problems. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.20.

7 Rajesh Chitnis, Mohammadtaghi Hajiaghayi, and Daniel Marx. Tight Bounds for Planar Strongly Connected Steiner Subgraph with Fixed Number of Terminals (and Extensions). In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1782–1801. SIAM, 2014. doi:10.1137/1.9781611973402.129.

8 Dietmar Cieslik. *Steiner minimal trees*, volume 23 of *Nonconvex Optimization and Its Applications*. Springer Science & Business Media, 1998.

9 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems as Hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41, 2016. doi:10.1145/2925416.

10 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic Graph Minor Theory: Improved Grid Minor Bounds and Wagner's Contraction. *Algorithmica*, 54(2):142–180, 2009. doi:10.1007/s00453-007-9138-y.

11 Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2017.

12 Irit Dinur and Pasin Manurangsi. ETH-Hardness of Approximating 2-CSPs and Directed Steiner Network. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:20, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.36.

13 Yevgeniy Dodis and Sanjeev Khanna. Design Networks with Bounded Pairwise Distance. In *Proc. 31th STOC*, pages 750–759. ACM, 1999.

14 Stuart E. Dreyfus and Robert A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

15 David Eppstein. Diameter and Treewidth in Minor-Closed Graph Families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/s004530010020.

16 Ranel E. Erickson, Clyde L. Monma, and Arthur F. Veinott, Jr. Send-and-Split Method for Minimum-Concave-Cost Network Flows. *Mathematics of Operations Research*, 12(4):634–664, 1987. doi:10.1287/moor.12.4.634.

17 Jon Feldman and Matthias Ruhl. The Directed Steiner Network Problem Is Tractable for a Constant Number of Terminals. *SIAM Journal on Computing*, 36(2):543–561, 2006.

18 Andreas Emil Feldmann and Dániel Marx. The Complexity Landscape of Fixed-Parameter Directed Steiner Network Problems. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2016.27.

19 Andreas Emil Feldmann and Dániel Marx. The Complexity Landscape of Fixed-Parameter Directed Steiner Network Problems. *CoRR*, abs/1707.06808, 2017. arXiv:1707.06808.

20 Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized Single-Exponential Time Polynomial Space Algorithm for Steiner Tree. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Proceedings, Part I*, volume 9134 of *LNCS*, pages 494–505. Springer, 2015. doi:10.1007/978-3-662-47672-7_40.

21 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

22 Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic Programming for Minimum Steiner Trees. *Theory of Computing Systems*, 41(3):493–500, 2007. doi:10.1007/s00224-007-1324-4.

23 Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *J. Combin. Theory Ser. B*, 99(1):218–228, 2009. doi:10.1016/j.jctb.2008.06.004.

24 Jonathan L. Gross and Thomas W. Tucker. *Topological Graph Theory*. Wiley-Interscience, New York, NY, USA, 1987.

25 Jiong Guo, Rolf Niedermeier, and Ondřej Suchý. Parameterized Complexity of Arc-Weighted Directed Steiner Problems. *SIAM Journal on Discrete Mathematics*, 25(2):583–599, 2011.

26 Eran Halperin and Robert Krauthgamer. Polylogarithmic Inapproximability. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 585–594, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780628.

27 Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier, 1992. doi:10.1016/S0167-5060(08)70188-2.

28 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

29 Richard M. Karp. Reducibility among Combinatorial Problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Proceedings of a symposium on the Complexity of Computer Computations, 1972*, pages 85–103, Boston, MA, 1972. Springer US. doi:10.1007/978-1-4684-2001-2_9.

30 Anatolii Y. Levin. Algorithm for the shortest connection of a group of graph vertices. *Sov. Math. Dokl.*, 12:1477–1481, 1971.

31 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.

32 Dániel Marx. Can You Beat Treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

33 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001.

34 Jesper Nederlof. Fast Polynomial-Space Algorithms Using Inclusion-Exclusion. *Algorithmica*, 65(4):868–884, 2013. doi:10.1007/s00453-012-9630-x.

35 Hans Jürgen Prömel and Angelika Steger. *The Steiner Tree Problem; a Tour through Graphs, Algorithms, and Complexity*. Vieweg, 2002.

36 Ondřej Suchý. On Directed Steiner Trees with Multiple Roots. In Pinar Heggernes, editor, *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, volume 9941 of *Lecture Notes in Computer Science*, pages 257–268, 2016. doi:10.1007/978-3-662-53536-3_22.

Space Lower Bounds for the Signal Detection Problem

Faith Ellen
University of Toronto, Canada
faith@cs.toronto.edu

Rati Gelashvili
University of Toronto, Canada
gelash@cs.toronto.edu

Philipp Woelfel
University of Calgary, Canada
woelfel@cpsc.ucalgary.ca

Leqi Zhu
University of Toronto, Canada
lezhu@cs.toronto.edu

Abstract

Many shared memory algorithms have to deal with the problem of determining whether the value of a shared object has changed in between two successive accesses of that object by a process when the responses from both are the same. Motivated by this problem, we define the *signal detection problem*, which can be studied on a purely combinatorial level. Consider a system with $n + 1$ processes consisting of n readers and one signaller. The processes communicate through a shared *blackboard* that can store a value from a domain of size m. Processes are scheduled by an adversary. When scheduled, a process reads the blackboard, modifies its contents arbitrarily, and, provided it is a reader, returns a Boolean value. A reader must return *true* if the signaller has taken a step since the reader's preceding step; otherwise it must return *false*.

Intuitively, in a system with n processes, signal detection should require at least n bits of shared information, i.e., $m \geq 2^n$. But a proof of this conjecture remains elusive. We prove a lower bound of $m \geq n^2$, as well as a tight lower bound of $m \geq 2^n$ for two restricted versions of the problem, where the processes are oblivious or where the signaller always resets the blackboard to the same fixed value. We also consider a one-shot version of the problem, where each reader takes at most two steps. In this case, we prove that it is necessary and sufficient that the blackboard can store $m = n + 1$ values.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Signal detection, ABA problem, space complexity, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.26

Funding We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). This research was undertaken, in part, thanks to funding from the Canada Research Chairs program. Rati Gelashvili was supported by the University of Toronto Faculty of Arts & Science Postdoctoral Fellowship.

1 Introduction

1.1 The Signal Detection Problem

Consider a system consisting of $n + 1$ processes, one *signaller*, s, and n *readers*, $r_1, \ldots, r_n$, that communicate through a shared blackboard. The blackboard can contain one value from a domain of size m. Processes are scheduled to take steps one at a time by an adversarial

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 26; pp. 26:1–26:13

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

scheduler. Whenever a process takes a step, it atomically reads the blackboard and can modify its contents arbitrarily, i.e. without interruption from other processes.

In the *signal detection problem*, each time a reader, r_i, has taken a step, it must return a Boolean value. If r_i has no preceding step, it can return either *true* or *false*. Otherwise, it must return *true* if and only if the signaller has taken a step since r_i's preceding step. We are concerned with how large m has to be for this problem to be solvable.

1.2 Simple Signal Detection Algorithms

For large or even unbounded values of m, there are simple solutions to the signal detection problem. For example, the board could store an unbounded *signal counter* that is initially 0. Each time the signaller takes a step, it increments the counter. When a reader is scheduled, it simply memorizes the counter value, but does not change it. To detect whether a signal has occurred since its last step, a reader only needs to compare the current counter value with the one it read in its previous step. The number of values the blackboard needs to store grows with the number of signals that occur, which can be unbounded.

The following simple protocol works for all executions and needs only to store an n-bit string $(b_1, \ldots, b_n)$ on the blackboard. Initially, $b_1 = \cdots = b_n = 0$, and whenever the signaller takes a step, it sets all bits to 1. For each $j \in \{1, \ldots, n\}$, reader r_j resets bit b_j to 0, returns *false* if this is r_j's first step, and returns the old value of b_j otherwise.

1.3 ABA Detection

Signal detection is related to the fundamental *ABA detection* problem in asynchronous shared memory systems. In such systems, a process that observes the same value A in some shared object in two successive accesses cannot tell whether the value of the object remained unchanged between them. More formally, it cannot distinguish between an execution in which the shared object did not change and an execution in which the value of the object changed from A to some other value B and then back to A. Many shared memory algorithms have to deal with this problem.

A well-known example is the double-collect algorithm for performing an atomic *scan* of an array [1]: A process repeatedly performs a *collect* (reading all components of the array one by one) until the sequences of values read in two consecutive collects are the same. This algorithm is only correct (linearizable) if no ABAs occur, meaning that any two consecutive reads of the same array entry return the same value if and only if the value of the array entry was not changed between the two reads. This is because it can be shown that, provided no ABAs occur, the sequence returned by a scan must be the contents of the array at the end of its second last collect and the beginning of its last collect. However, in executions in which ABAs occur, this implementation might incorrectly return a sequence of values that was not the contents of the array at any point during the execution.

A standard approach to dealing with ABAs is tagging, as introduced by IBM [6], whereby a shared object gets augmented with a tag that changes with every write operation. If tags are never reused, the ABA problem can be avoided. From a theory perspective this solution is unsatisfactory: If there is no bound on the length of executions, then unbounded sized objects are required to accommodate ever increasing tag values. Even though, in many practical scenarios, a system may never run out of tags, it is often desirable or even necessary to use an entire word for data. In such scenarios, the tag associated with a data word could be stored in a subsequent memory location and double-width atomic instructions could be used. However, these are not supported by most of today's mainstream architectures [8].

In some cases, it is possible to store the tag in an unrelated memory location [7], but this requires technically difficult algorithms and tedious correctness proofs. As a result, algorithm designers often deal with ABAs in an ad-hoc way. For example, handshaking bits can be used to detect changes in the components of the array in a wait-free implementation of a snapshot object [1]. Such solutions are algorithm specific and require individual correctness proofs.

ABAs can also occur when using compare-and-swap (CAS) objects, which are provided by most existing multiprocessor systems and are much more powerful than read/write registers. Algorithms devised in theoretical research often use load-linked store-conditional (LL/SC) objects, which do not suffer from ABAs, and can easily replace CAS objects. Unfortunately, only a small number of multiprocessor systems provide LL/SC and they are weaker than the LL/SC specification used in theoretical research. Variants of LL/SC available in modern hardware restrict programmers severely in how the objects can be used [10], and "offer little or no help with preventing the ABA problem" [9].

To study the complexity of ABA detection, Aghazadeh and Woelfel [2] defined an *ABA detecting register*, which extends a read/write register with the ability to detect ABAs. It supports the operations `DWrite(`x`)`, which changes the value of the object to x, and `DRead()`, which returns the current value of the object together with a Boolean flag. The flag is true if and only if the process has previously performed `DRead()` and, since its last preceding `DRead()`, some process performed `DWrite()`. The authors proved space lower bounds and time-space-tradeoffs for linearizable implementations of ABA detecting registers in shared memory systems with n processors that provide bounded atomic base objects, such as read/write registers or CAS objects. For example, if only bounded read/write registers are available as base objects, then at least $n-1$ of them are needed to obtain an obstruction-free ABA detecting register. If bounded CAS objects are also available, then any implementation using m base objects has step-complexity $\Omega(n/m)$.

All the lower bound results in [2] are specific to the base objects provided by the system, and provide no insights for systems using different sets of base objects. But we conjecture that there is a large, general lower bound for the amount of information that needs to be stored in a system for processes to detect ABAs: Intuitively, the system state needs to keep track of whether the value of the object has changed since each process last accessed the object. This requires at least n bits of information. Hence, it seems believable that detecting ABAs in any system with arbitrarily powerful base objects requires at least n bits of information to be stored either in the base object or in the hardware implementing the base objects (for example, implementing LL/SC objects). Using the reasonable assumption that a single base object can store $O(\log n)$ bits of information, this would imply that $\Omega(n/\log n)$ base objects are required for implementing a single ABA detecting object.

The signal detection problem is a restricted version of the problem of detecting ABAs in asynchronous shared memory systems, stripped down to the essentials necessary for determining the information theoretic requirements. Its definition is self-contained, and the problem can be studied without any background knowledge on shared memory systems. If n processes can detect ABAs in a standard asynchronous shared memory system with arbitrarily strong primitives, then they can also solve signal detection. Therefore, if m^* is the smallest value of m (the number of values stored on the blackboard) for which signal detection can be solved, then $\log_2 m^*$ is a lower bound for the number of bits needed for ABA detection.

1.4 Results

We conjecture that any solution to the signal detection problem requires $m \geq 2^n$. This simply defined combinatorial problem does not seem to have a simple solution and a proof of the conjecture has eluded us so far. Even a proof of a polynomial lower bound is surprisingly non-trivial. We show the following.

▶ **Theorem 1.** *In any algorithm for the signal detection problem, the blackboard stores $m = \Omega(n^2)$ different values.*

To obtain better understanding, we consider several restricted versions of the signal detection problem and prove tight upper and lower bounds for them.

First, we consider a one-shot version of signal detection, where no reader takes more than two steps (but the signaller can take arbitrarily many steps). We show that this problem is strictly easier than the unrestricted version of the problem by showing that one-shot signal detection can be solved with $n + 1$ different blackboard values, which is optimal.

▶ **Theorem 2.** *The minimum number of different values that the blackboard stores in an algorithm that solves the one-shot signal detection problem is $m = n + 1$.*

Then we consider the case of *oblivious* processes. Here each process p is equipped with a fixed function $f_p : \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$. When taking a step it replaces the blackboard contents x with $f_p(x)$. Hence, what a process writes to the blackboard is independent of the process' internal state (but the return value of a reader's step may not be). In the simple algorithm above, which uses $m = 2^n$ blackboard values, processes are oblivious. In fact, what a reader returns also only depends on the contents of the blackboard and not on its internal state. We prove that when processes are oblivious, no better algorithm exists.

▶ **Theorem 3.** *In any algorithm for the signal detection problem with oblivious processes, the blackboard stores $m \geq 2^n$ different values.*

The signal detection problem with oblivious processes is similar to determining the minimum size of a dictionary in a sequential system. A *dictionary* supports three operations, *insert*(x), *query*(x), and *reset*(), where x is a parameter chosen from a domain of size n. A call to *query*(x) returns true if there has been an *insert*(x) operation since the last *reset*() operation or since the beginning of the execution, if there has been no *reset*(). Otherwise, it returns false. A dictionary implemented using $b(n)$ bits immediately yields a solution to the signal detection problem with oblivious processes as follows: A blackboard with $m = 2^{b(n)}$ possible values is used to store the dictionary. When a signaler takes a step, it simulates a *reset*() operation on the dictionary stored on the blackboard. Similarly, when reader r_i takes a step, it simulates *query*(i) followed by *insert(i)* on the dictionary and then returns the return value of its query operation. However, an arbitrary solution to the signal detection problem does not seem to yield an implementation of a dictionary. The difficulty is that the return value of a step by a reader r_i can depend on the state of the reader and, thus, its entire past execution. In contrast, the result of a *query*(i) operation is only a function of the state of the dictionary. Hence, the n-bit information theory lower bound for implementing a dictionary cannot be used to obtain Theorem 3.

We also consider signal detection with *identical signals*, where the signaller always resets the blackboard to the same value. Note that the simple algorithm above with $m = 2^n$ uses identical signals. Studying this restricted problem has another motivation: Consider a shared memory system, where shared memory objects may be reset to their initial states at arbitrary times. For example, this can happen due to power outages if volatile memory

is used. A solution to signal detection with identical signals corresponds to an algorithm where processes can detect that such faults have happened. This may allow them to start a recovery procedure. This is dual to the recently introduced notion of *recoverable* algorithms [5, 4, 3], which tolerate power outages when the local variables of processes are stored on volatile memory, but shared memory is non-volatile.

▶ **Theorem 4.** *In any algorithm for the signal detection problem with identical signals, the blackboard stores $m \geq 2^n$ different values.*

The lower bound proofs of $m \geq 2^n$ for signal detection with either oblivious readers or identical signals have one interesting aspect in common. We show that one can reach a configuration, C, from which 2^n different blackboard values result from the 2^n schedules that are sub-sequences of $(r_1, \ldots, r_n)$. For our simple algorithm, each execution that ends with the signaller taking a step results in a configuration with this property. We show that a lower bound proof for the unrestricted signal detection problem cannot rely on this property. In particular, we present an algorithm for two readers, r_1 and r_2, which uses a bounded number of blackboard values, such that every reachable configuration C satisfies the following: the schedules sr_1 and sr_2 performed starting from C result in configurations with the same blackboard contents. Hence, in contrast to our earlier intuition, it is not necessary for the blackboard to store information about which processes have taken steps since the signaller last took a step: Csr_1 and Csr_2 are indistinguishable to the signaller. This algorithm uses $m = 16$ blackboard values, so it does not contradict our conjecture. However, it has interesting implications for lower bound proof techniques - for example, the approach that we used to prove Theorem 4 does not apply to this particular algorithm.

2 Preliminaries

We consider a deterministic, asynchronous system in which $n + 1$ processes with unique IDs in $\{s, r_1, \ldots, r_n\}$ communicate with one another using a single shared *blackboard*. Each time a process takes a *step*, it atomically reads the blackboard, may change the value of the blackboard based on its state and the value it read, and updates its state.

A *configuration* C consists of a value, $\mathsf{v}(C)$, for the blackboard and a state for each process. An *execution* is an alternating sequence of configurations and steps. If C is a configuration and α is a finite execution starting from C, then $C\alpha$ denotes the configuration at the end of α. For any set of processes, P, a *P-only execution* is an execution in which only processes in P take steps in the execution. A solo execution is a P-only execution in which P contains only one process, i.e., all steps in the execution are by the same process.

A *schedule* is a sequence of processes (in which the same process can occur multiple times). For any (deterministic) algorithm and for any configuration C, a schedule α determines a unique execution starting from C in which the processes take steps in the order specified by the schedule. The configuration at the end of this execution is called $C\alpha$.

Two configurations, C and C', are *indistinguishable* to a set of processes, P, if $\mathsf{v}(C) = \mathsf{v}(C')$ and each process in P has the same state in C as it does in C'. If C and C' are indistinguishable to P and α is a finite P-only execution from C, then it is also an execution from C', and $C\alpha$ and $C'\alpha$ are also indistinguishable to P.

3 One-Shot Signal Detection

Recall that in the one-shot signal detection problem, no reader takes more than two steps, but the signaller can take arbitrarily many steps. Consider the following algorithm that solves this problem using $m = n + 1$ values:

- The blackboard initially has value 0.
- Whenever s takes a step, it resets the blackboard contents to 0.
- When r_i takes its first step, it changes the blackboard contents to i if it reads 0; otherwise it leaves the blackboard unchanged. In either case, r_i locally stores the value $v_i \neq 0$ of the blackboard immediately after its first step and returns *false*. Let $v_i \neq 0$ denote the value of the blackboard immediately after this step.
- When r_i takes its second step, it returns *false* if it reads v_i from the blackboard; otherwise it returns *true*. It does not change the value of the blackboard in either case.

Note that only the signaller changes the blackboard contents to 0 and a reader only changes the blackboard contents from 0. Thus, if the signaller does not take any steps between the two steps of reader r_i, then the value of the blackboard remains v_i during this interval and r_i returns *false*.

If the signaller does take a step between the two steps of reader r_i, then the blackboard is reset to 0. Consider the last step, S', by the signaller during this interval. If no reader takes its first step after S', but before the second step by r_i, then r_i will read 0 from the blackboard on its second step and return *true* Otherwise, consider the first step after S' in which a reader r_j takes its first step. It will change the blackboard contents to j. Note that $j \neq v_i$, since r_j is the only reader that can change the blackboard contents to j and r_j has not previously taken a step. In this case, r_i will read j from the blackboard on its second step and return *true*.

There is also a matching lower bound. In both of the following proofs, it is sufficient to restrict attention to executions in which each reader takes at most two steps.

▶ **Lemma 5.** *Let C be a configuration and let r be a reader. If α is a $(\{r_1, \ldots, r_n\} - \{r\})$-only execution from $C' = Cr$ and β is a $(\{s, r_1, \ldots, r_n\} - \{r\})$-only execution from $C'\alpha s$, then, for every configuration D in α and every configuration E in β, $\mathsf{v}(D) \neq \mathsf{v}(E)$.*

Proof. Suppose not. Then there is some configuration D in α and some configuration E in β such that $\mathsf{v}(D) = \mathsf{v}(E)$. Since r takes no steps in $\alpha s \beta$, D and E are indistinguishable to r. Note that r must return *false* if it takes a step in configuration D, because s has not taken any steps since r last took a step. However, r must return *true* if it takes a step in configuration E, because s has taken a step since r last took a step. This is impossible, because D and E are indistinguishable to r. ◀

We can now prove Theorem 2, restated for convenience:

▶ **Theorem 2.** *The minimum number of different values that the blackboard stores in an algorithm that solves the one-shot signal detection problem is $m = n + 1$.*

Proof. In the beginning of this section, we gave an algorithm for one-shot signal detection in which blackboard stores $n + 1$ values. In the following we show that, in any algorithm for one-shot signal detection, the blackboard stores at least $n + 1$ different values.

Let C_0 be the initial configuration. For $1 \leq j \leq n$, let $C_j = C_{j-1} s r_j$, and let $C_{n+1} = C_n s$.

For $1 \leq i < n$, consider the empty execution α from C_i and the execution β from $C_i s$ with schedule $r_{i+1} \cdots s r_n s$. By Lemma 5 with $C' = C_i$ and $r = r_i$, $\mathsf{v}(C_i) \neq \mathsf{v}(E)$ for all configurations E in β. In particular, $\mathsf{v}(C_i) \neq \mathsf{v}(C_j)$ for $i + 1 \leq j \leq n + 1$.

For $i = n$, consider the empty execution α from C_n and the empty execution β from $C_n s = C_{n+1}$ By Lemma 5 with $C' = C_n$ and $r = r_n$, $\mathsf{v}(C_n) \neq \mathsf{v}(C_{n+1})$.

Hence $|\{\mathsf{v}(C_1), \ldots, \mathsf{v}(C_n), \mathsf{v}(C_{n+1})\}| = n + 1$. ◀

There is a simple generalization of the algorithm for one-shot signal detection using $m = n + 1$ values to an algorithm for signal detection using $m = bn + 1$ values when each reader can perform at most $b + 1$ steps: When a reader r_i reads a 0 from the blackboard in its j'th step, for $1 \leq j \leq b$, it changes the blackboard contents to (i, j), instead of i, and stores the value of the blackboard in v_i. When r_i takes its first step, it always returns *false*. When r_i takes subsequent steps, it returns *false* if it reads v_i from the blackboard; otherwise it returns *true*.

4 Identical Signals

Suppose that the signaller always resets the contents of the blackboard to a fixed value, say 0. We show that the blackboard must be able to store at least 2^n values.

Given a set of readers, R, let $\vec{R}$ denote the schedule consisting of one occurrence of each reader in R, in order of their identifiers, and let $M(R)$ denote the set $\{r_i : i \leq j \text{ for some } r_j \in R\}$ of all readers whose identities are less than or equal to the largest identity of the readers in R. In particular, $M(\emptyset) = \emptyset$. For example, $M(\{r_1, r_4, r_8\}) = \{r_1, r_2, \ldots, r_8\}$. Notice that, for any two sets of readers R and R', either $M(R) \subseteq M(R')$ or $M(R') \subseteq M(R)$. There are $n + 1$ such sets, i.e., $|\{M(R) : R \subseteq \{r_1, \ldots, r_n\}\}| = n + 1$.

▶ **Lemma 6.** *If the blackboard can only store a finite number of different values, then it is possible to reach a configuration D such that, for every set of readers, T, there is a $(M(T) \cup \{s\})$-only execution β from $D\vec{T}s$ such that $\mathsf{v}(D\vec{T}s\beta) = \mathsf{v}(D\vec{T})$.*

Proof. Assume that, for all reachable configurations C, there is a set of readers, T, such that, for all $(M(T) \cup \{s\})$-only executions β from $C\vec{T}s$, $\mathsf{v}(C\vec{T}s\beta) \neq \mathsf{v}(C\vec{T})$. We define an infinite sequence $(C_i)_{i \geq 0}$ of reachable configurations as follows. Let C_0 be the initial configuration. For $j \geq 1$, let T_j be a set of readers such that for all $(M(T_j) \cup \{s\})$-only executions β from $C_{j-1}\vec{T}_j s$, $\mathsf{v}(C_{j-1}\vec{T}_j s\beta) \neq \mathsf{v}(C_{j-1}\vec{T}_j)$. The existence of T_j follows from the assumption, since C_{j-1} is reachable. Let $C_j = C_{j-1}\vec{T}_j s$.

Consider the infinite sequence $(M(T_j))_{j \geq 1}$. Since $|\{M(R) : R \subseteq \{r_1, \ldots, r_n\}\}| = n + 1$, some set of readers occurs in the sequence infinitely often. Let M be the largest such set, let $J = \{j \geq 1 : M(T_j) = M\}$, and let $k^* = \min\{k \geq 1 : M(T_j) \subseteq M \text{ for all } j \geq k\}$. Note that, for all $k, \ell \in J$ such that $k^* \leq k < \ell$, the schedule $\vec{T}_{k+1} s \vec{T}_{k+2} s \cdots \vec{T}_\ell$ is $(M \cup \{s\})$-only. Thus, by definition of T_k, $\mathsf{v}(C_k\vec{T}_k) \neq \mathsf{v}(C_k\vec{T}_k s\vec{T}_{k+1} s \cdots \vec{T}_\ell) = \mathsf{v}(C_\ell\vec{T}_\ell)$. Therefore, the blackboard can store an infinite number of values. ◀

This allows us to prove Theorem 4 (restated):

▶ **Theorem 4.** *In any algorithm for the signal detection problem with identical signals, the blackboard stores $m \geq 2^n$ different values.*

Proof. Suppose the blackboard can only store a finite number of values. Then, by Lemma 6, it is possible to reach a configuration D such that, for any set of readers T, there is a $(M(T) \cup \{s\})$-only execution β from $D\vec{T}s$ such that $\mathsf{v}(D\vec{T}s\beta) = \mathsf{v}(D\vec{T})$.

Suppose there exist two different sets of readers $R, R' \subseteq \{r_1, \ldots, r_n\}$ such that $\mathsf{v}(D\vec{R}) = \mathsf{v}(D\vec{R}')$. Without loss of generality, $\vec{R} = \vec{T}x\vec{X}$ and $\vec{R}' = \vec{T}\vec{X}'$, where $x \in R - R'$ and $\vec{T}$ is the longest common prefix of $\vec{R}$ and $\vec{R}'$. Note that $M(T) \cap (\{x\} \cup X \cup X') = \emptyset$ since $\vec{R}$ and $\vec{R}'$ are sorted. By definition of D, there is a $(M(T) \cup \{s\})$-only execution β from $D\vec{T}s$ such that $\mathsf{v}(D\vec{T}s\beta) = \mathsf{v}(D\vec{T})$. Consider the execution β' from $D\vec{T}xs$ which has the same schedule as β. Since $D\vec{T}s$ and $D\vec{T}xs$ are indistinguishable to $M(T)$ and s always sets the blackboard to 0, the corresponding configurations in β and β' are indistinguishable to $M(T)$. In particular,

$\mathsf{v}(D\vec{T}xs\beta) = \mathsf{v}(D\vec{T})$. Since $(M(T) \cup \{s, x\}) \cap X' = \emptyset$, configurations $D\vec{T}xs\beta$ and $D\vec{T}$ are indistinguishable to the set of readers X'. Thus $\mathsf{v}(D\vec{T}xs\beta\vec{X'}) = \mathsf{v}(D\vec{T}\vec{X'}) = \mathsf{v}(D\vec{R'}) = \mathsf{v}(D\vec{R}) = \mathsf{v}(D\vec{T}x\vec{X})$. Since $x \notin M(T) \cup X \cup X' \cup \{s\}$, it follows that $D\vec{T}xs\beta\vec{X'}$ and $D\vec{T}x\vec{X}$ are indistinguishable to x. Note that x must return *false* if it takes a step in configuration $D\vec{T}x\vec{X}$, because s has not taken any steps since x last took a step. However, x must return *true* if it takes a step in configuration $D\vec{T}xs\beta\vec{X'}$, because s has taken a step since x last took a step. This is impossible, because these two configurations are indistinguishable to x.

Hence, $\mathsf{v}(D\vec{R}) \neq \mathsf{v}(D\vec{R'})$ for all different sets of readers R and R', so $|\{\mathsf{v}(D\vec{R}) : R \subseteq \{r_1, \ldots, r_n\}\}| = 2^n$. ◀

If the signaller can only read from the blackboard and write to the blackboard, but cannot perform atomic read-modify-write operations, the blackboard must also store at least 2^n different values. The same proof works, provided the scheduler only lets the signaller write to the blackboard in solo executions that begin with a read of the blackboard. In such executions, the signaller writes a fixed sequence of values, that does not depend on the steps taken by the readers. This is all that is necessary to prove that the corresponding configurations in β and β' are indistinguishable to $M(T)$ and, therefore, $\mathsf{v}(D\vec{T}xs\beta) = \mathsf{v}(D\vec{T})$.

5 Oblivious Processes

Recall that a process is oblivious, if what it writes to the blackboard in a step only depends on the value of the blackboard at the beginning of that step. In this section we prove Theorem 3, which we restate for convenience:

▶ **Theorem 3.** *In any algorithm for the signal detection problem with oblivious processes, the blackboard stores $m \geq 2^n$ different values.*

Proof. Suppose the blackboard stores fewer than 2^n different values. For every (possibly empty) set of readers R and every positive integer i, consider the schedule $\rho_i(R)$, which consists of $s\vec{R}$ repeated i times. Because the blackboard stores fewer than 2^n different values, the blackboard contents will repeat when schedule $\rho_{2^n}(R)$ is applied starting from the initial configuration, C_0. Let $L(R) = \mathsf{v}(C_0\rho_\ell(R))$, where ℓ is the index of the first repetition in the sequence $\mathsf{v}(C_0\rho_i(R))_{i\geq 1}$.

Let R and R' be any two different sets of readers. Without loss of generality, suppose there is a reader $r_k \in R' \setminus R$. To obtain a contradiction, assume that $L(R) = L(R')$. Let $0 < i < j$ and $0 < i' < j'$ be such that $L(R) = \mathsf{v}(C_0\rho_i(R)) = \mathsf{v}(C_0\rho_j(R))$ and $L(R') = \mathsf{v}(C_0\rho_{i'}(R')) = \mathsf{v}(C_0\rho_{j'}(R'))$. Since processes are oblivious, and $\mathsf{v}(C_0\rho_{i'}(R')) = L(R') = L(R) = \mathsf{v}(C_0\rho_i(R))$, it follows that $\mathsf{v}(C_0\rho_{i'}(R')\rho_{j-i}(R)) = \mathsf{v}(C_0\rho_i(R)\rho_{j-i}(R)) = \mathsf{v}(C_0\rho_j(R)) = \mathsf{v}(C_0\rho_{i'}(R'))$. Since r_k takes no steps in $\rho_{j-i}(R)$, configurations $C_0\rho_{i'}(R')\rho_{j-i}(R)$ and $C_0\rho_{i'}(R')$ are indistinguishable to r_k. This is impossible, as the signaller has taken a step after r_k's last step in $C_0\rho_{i'}(R')\rho_{j-i}(R)$, but not in $C_0\rho_{i'}(R')$, so r_k would have to return different responses if it takes the next step. Thus, if $R \neq R'$, then $L(R) \neq L(R')$.

However, since there are 2^n different sets of readers and the blackboard stores fewer than 2^n different values, this contradicts the pigeon-hole principle. ◀

6 The General Setting

Let $\mathcal{M} = \{M(R) : R \subseteq \{r_1, \ldots, r_n\}\}$, and recall that $|\mathcal{M}| = n + 1$. For any execution α, let $M(\alpha)$ denote $M(R)$, where R is the set of readers that take steps in α.

▶ **Lemma 7.** *If the blackboard can only store a finite number of different values, then, from any configuration, it is possible to reach a configuration D such that, for any pair of executions α and β from D, there exists an $(M(\alpha) \cup M(\beta) \cup \{s\})$-only execution γ from $D\alpha$ such that $\mathsf{v}(D\alpha\gamma) = \mathsf{v}(D\beta)$.*

Proof. Let C_0 be an arbitrary configuration. To obtain a contradiction, suppose that, for all configurations C reachable from C_0, there are two executions, α and β from C such that for all $(M(\alpha) \cup M(\beta) \cup \{s\})$-only executions γ from $C\alpha$, $\mathsf{v}(C\alpha\gamma) \neq \mathsf{v}(C\beta)$.

We inductively define an infinite execution δ starting from C_0 and an infinite sequence of configurations C_j, for $j \geq 0$, in this execution such that C_j precedes C_{j+1}. In particular, C_j is reachable from C_0, so there exist two executions, α_{j+1} and β_{j+1} from C_j such that for all $(M(\alpha_{j+1}) \cup M(\beta_{j+1}) \cup \{s\})$-only executions γ from $C_j\alpha_{j+1}$, $\mathsf{v}(C_j\alpha_{j+1}\gamma) \neq \mathsf{v}(C_j\beta_{j+1})$. Let $C_{j+1} = C_j\alpha_{j+1}$ and let $\delta = \alpha_1\alpha_2\cdots$.

For $j \geq 1$, let $M_j = M(\alpha_j) \cup M(\beta_j) \in \mathcal{M}$. Since $\mathcal{M}$ is finite, there exists at least one set in $\mathcal{M}$ that occurs an infinite number of times in $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \ldots$. Let M' denote the largest such set and let $J = \{j \geq 1 : M_j = M'\}$ be the set of indices of the occurrences of M'. Let $k^* = \min\{k \geq 1 : M_j \subseteq M' \text{ for all } j \geq k\}$ be the first such index after which no set larger than M' occurs. Note that, if $k^* \leq k < \ell$ then $\gamma = \alpha_{k+1}\cdots\alpha_{\ell-1}\beta_\ell$ is an $(M' \cup \{s\})$-only execution from $C_{k-1}\alpha_k$. Hence, if $k, \ell \in J$, then $\mathsf{v}(C_{k-1}\beta_k) \neq \mathsf{v}(C_{k-1}\alpha_k\gamma) = \mathsf{v}(C_{\ell-1}\beta_\ell)$. Thus $\{\mathsf{v}(C_{k-1}\beta_k) : k \geq k^* \text{ and } k \in J\}$ is an infinite set of values that can appear on the blackboard. This contradicts the assumption that the blackboard can only store finite number of different values. ◀

Let D be a configuration such that, for any pair of executions α and β from D, there exists an $(M(\alpha) \cup M(\beta) \cup \{s\})$-only execution γ from $D\alpha$ such that $\mathsf{v}(D\alpha\gamma) = \mathsf{v}(D\beta)$. For $0 \leq i < j \leq n$, let $\delta(i, j)$ denote the schedule $r_1 s r_2 s \ldots r_i s r_{i+1} r_{i+2} \ldots r_j$. For example, $\delta(0, 3) = r_1 r_2 r_3$ and $\delta(2, 3) = r_1 s r_2 s r_3$.

▶ **Lemma 8.** *If $0 \leq i < j \leq n$, $0 \leq i' < j' \leq n$, and either $i \neq i'$ or $j \neq j'$, then $\mathsf{v}(D\delta(i, j)) \neq \mathsf{v}(D\delta(i', j'))$.*

Proof. First consider the case when $i \neq i'$. Without loss of generality, suppose that $i < i'$. The state of reader r_{i+1} is the same in configurations $D\delta(i, j)$ and $D\delta(i', j')$. In configuration $D\delta(i, j)$, if r_{i+1} takes a step, it must return *false*, because s has not taken any steps since r_{i+1} last took a step. In configuration $D\delta(i', j')$, if r_{i+1} takes a step, it must return *true*, because s has taken $i' - i$ steps since r_{i+1} last took a step. If $\mathsf{v}(D\delta(i, j)) = \mathsf{v}(D\delta(i', j'))$, then configurations $D\delta(i, j)$ and $D\delta(i', j')$ are indistinguishable to r_{i+1}, which is impossible. Thus $\mathsf{v}(D\delta(i, j)) \neq \mathsf{v}(D\delta(i', j'))$.

Now consider the case when $i = i'$ and $j \neq j'$. Without loss of generality, suppose that $j < j'$. Let $\delta' = r_{j+1} \cdots r_{j'}$, so $\delta(i', j') = \delta(i, j)\delta'$. By Lemma 7, where α is the execution of schedule $\delta(i, j)s$ starting from D and β is the execution of schedule $\delta(i, j)$ starting from D, there exists an $\{r_1, \ldots, r_j, s\}$-only execution γ such that $\mathsf{v}(D\delta(i, j)s\gamma) = \mathsf{v}(D\delta(i, j))$.

To obtain a contradiction, suppose that $\mathsf{v}(D\delta(i', j')) = \mathsf{v}(D\delta(i, j))$. Configurations $D\delta(i', j')$ and $D\delta(i, j)$ are indistinguishable to $r_1, \ldots, r_j$, and s, since the signaller and these readers take no steps in δ'. Let g be the schedule of execution γ. Then $\mathsf{v}(D\delta(i', j')sg) = \mathsf{v}(D\delta(i, j)sg) = \mathsf{v}(D\delta(i, j)s\gamma) = \mathsf{v}(D\delta(i, j)) = \mathsf{v}(D\delta(i', j'))$. Since r_{j+1} does not appear in sg, configurations $D\delta(i', j')sg$ and $D\delta(i', j')$ are indistinguishable to r_{j+1}. Note that r_{j+1} must return *false* if it takes a step in configuration $D\delta(i', j')$, because s has not taken any steps since r_{j+1} last took a step. However, r_{j+1} must return *true* if it takes a step in configuration $D\delta(i', j')sg$, because s has taken a step since r_{j+1} last took a step. This is impossible, because $D\delta(i', j')sg$ and $D\delta(i', j')$ are indistinguishable to r_{j+1}. ◀

Using this lemma, we obtain Theorem 1 (restated for convenience):

▶ **Theorem 1.** *In any algorithm for the signal detection problem, the blackboard stores $m = \Omega(n^2)$ different values.*

Proof. Consider any algorithm for signal detection in which the blackboard stores a finite number of different values. By Lemma 7, there is a reachable configuration D such that, for any pair of executions α and β from D, there exists an $(M(\alpha) \cup M(\beta) \cup \{s\})$-only execution γ from $D\alpha$ such that $\mathsf{v}(D\alpha\gamma) = \mathsf{v}(D\beta)$. By Lemma 8, for all different choices of $0 \le i < j \le n$, the value of the blackboard in configuration $D\delta(i, j)$ is different. There are $n(n+1)/2 \in \Omega(n^2)$ such choices. ◀

7 Two Process Algorithm

We describe an algorithm for signal detection among $n = 2$ readers, r_1 and r_2, using $m = 16$ values. The algorithm has the property that, for every reachable configuration C, $\mathsf{v}(Csr_1) = \mathsf{v}(Csr_2)$. This will allow us to show that, from any reachable configuration C, the number of different blackboard values that can be reached from C using $\{r_1, r_2\}$-only executions is at most 3. Thus, in order to show the existence of 4 different blackboard values from some configuration C, the signaller must also take steps. Note that our proof of the reset case does not do this, so it is unlikely to be generalized.

At all times, the contents of the blackboard is a quadruple $(track, position, both, flag) \in \{0, 1\}^4$. Initially, the blackboard has value $(0, 0, 1, 1)$. The $flag$ is used to indicate whether the last step was taken by the signaller. In particular, the signaller always sets $flag$ to 1 and the readers always set $flag$ to 0. Each reader r_i has 3 local variables, t_i, p_i and $jump_i$. Initially, $(t_i, p_i) = (0, 0)$ and $jump_i = false$. Variables t_i and p_i represent the last values that r_i wrote to the *track* and *position* fields of the blackboard, even if it didn't change their values. Readers only change these fields when the signaller sets $flag$ to 1. If $t_1 = t_2$ and $p_1 = p_2$ in some configuration C, then $both = 1$ in C. Otherwise, it is 0. If variable $jump_i$ is $true$, then when r_i takes its next step, it will change the track, provided it sees $track = t_i$, $position = p_i$, $both = 0$, and $flag = 1$ on the blackboard.

Suppose r_i reads (t, p, b, f) from the blackboard. Then, in its next step, r_i does the following:

1. If $f = 1$ and $b = 1$, then r_i changes *track*, sets *position* to 0, sets *both* to 0, and sets $jump_i$ to *false*.
2. If $f = 1$, $b = 0$, $t = t_i$, $p = p_i$ and $jump_i = false$, then r_i only changes *position*.
3. If $f = 1$, $b = 0$, $t = t_i$, $p = p_i$, and $jump_i = true$, then r_i changes *track*, sets *position* to 0, and sets $jump_i$ to *false*.
4. If $f = 1$, $b = 0$, $t = t_i$, and $p \neq p_i$, then r_i changes *track* and sets $jump_i$ to *false*.
5. If $f = 1$, $b = 0$, and $t \neq t_i$, then r_i changes *position* and sets $jump_i$ to *true*.
6. If $f = 0$ and $t \neq t_i$ or $p \neq p_i$, then r_i sets *both* to 1 and sets $jump_i$ to *false*.
7. If $f = 0$, $t = t_i$, and $p = p_i$, then r_i doesn't change anything.

In the first 6 cases, r_i returns *true*. In case 7, r_i returns *false*. Pseudocode appears in Algorithm 1.

Note that consecutive steps by a process do not change its state or the blackboard. Moreover, if s takes a step followed by an $\{r_1, r_2\}$-only execution in which they each take at least one step, then, in the resulting configuration, their t_i and p_i variables will be equal to track and position on the blackboard. From then on, r_1 and r_2 will not change their local variables or the blackboard until the next signaller step. Therefore, we may restrict attention

to schedules of the form $\alpha_1 s\alpha_2 \cdots s\alpha_\ell$ and $s\alpha_1 s\alpha_2 \cdots s\alpha_\ell$, where each α_k is an $\{r_1, r_2\}$-only schedule in which r_1 and r_2 each occur at most once and α_k is non-empty for $k < \ell$. Since *flag* is initially 1, a step by the signaller does not change the value of the blackboard. Hence we may assume α begins with s.

Algorithm 1: Pseudocode for reader r_i.

```
1  (track, position, both, flag) ← read from blackboard
2  if (flag = 0) ∧ ((track, position) = (t_i, p_i)) then
3      return false
4  if (flag = 0) then
5      write (track, position, 1, 0) to blackboard
6  else if (both = 0) ∧ ((track ≠ t_i) ∨ ((position = p_i) ∧ ¬jump_i)) then
7      write (track, 1 − position, 0, 0) to blackboard
8  else
9      write (1 − track, 0, 0, 0) to blackboard
10 jump_i ← (flag = 1) ∧ (both = 0) ∧ (track ≠ t_i)
11 (t_i, p_i) ← last track and position written
12 return true
```

Given a reachable configuration C, let $t_i(C)$, $p_i(C)$, and $jump_i(C)$ denote the value of reader r_i's local variables t_i, p_i, and $jump_i$, respectively, in C, and let $track(C)$, $position(C)$, $both(C)$, and $flag(C)$ denote the values of the *track*, *position*, *both*, and *flag* fields, respectively, on the blackboard in C.

▶ **Lemma 9.** *For every reachable configuration C and every $i \in \{1, 2\}$,*

$$(t_i(C), p_i(C)) \neq (track(Csr_{3-i}), position(Csr_{3-i}))\,.$$

Proof. Suppose, for a contradiction, that this is not the case. Consider a shortest schedule α such that, in configuration $C = C_0\alpha$, $(t_i(C), p_i(C)) = (track(Csr_{3-i}), position(Csr_{3-i}))$, for some $i \in \{1, 2\}$. As discussed above, $\alpha = s\alpha_1 \cdots s\alpha_\ell$, where each α_k is an $\{r_1, r_2\}$-only schedule in which r_1 and r_2 each occur at most once and α_k is non-empty for $k < \ell$.

Suppose r_i does not occur in α. Then $t_i(C) = t_i(C_0) = 0$. Since $flag(C_0) = 1$, $both(C_0) = 1$, and $track(C_0) = 0$, $track(C_0 sr_{3-i}) = 1$. If only s and r_{3-i} take steps from $C_0 sr_{3-i}$, then neither changes *track*. Since α is a $\{s, r_{3-i}\}$-only schedule, $track(Csr_{3-i}) = 1$. Therefore, $t_i(C) = 0 \neq 1 = track(Csr_{3-i})$, which contradicts the fact that $t_i(C) = track(Csr_{3-i})$.

Now suppose that r_i occurs at least once in α. Suppose it last occurs in α_k. Let $C' = C_0 s\alpha_1 \cdots s\alpha_k$. Since r_i does not occur in the remainder of α, $(t_i(C), p_i(C)) = (t_i(C'), p_i(C')) = (track(C'), position(C'))$. There are 3 cases:

Case 1: $k = \ell$. Then $C = C'$. If r_{3-i} takes a step from Cs, it either changes *track* or *position*. Hence, either $t_i(C) \neq track(Csr_{3-i})$ or $p_i(C) \neq position(Csr_{3-i})$. This contradicts the fact that $(t_i(C), p_i(C)) = (track(Csr_{3-i}), position(Csr_{3-i}))$.

Case 2: $k < \ell$ and α_k contains both r_1 and r_2. Then $1 = both(C') = both(C's)$. Hence, if r_{3-i} takes a step from $C's$, it changes *track* to $1 - track(C')$. As r_i does not take any more steps, r_{3-i} does not subsequently change *track*. Thus, $t_i(C) = track(C') \neq 1 - track(C') = track(Csr_{3-i})$. This contradicts the fact that $t_i(C) = track(Csr_{3-i})$.

Case 3: $k < \ell$ and $\alpha_k = r_i$. Since $s\alpha_1 \cdots s\alpha_k$ is strictly shorter than $s\alpha_1 \cdots s\alpha_\ell$, $(t_i(C'), p_i(C')) \neq (track(C'sr_{3-i}), position(C'sr_{3-i}))$. Moreover, if $\alpha = s\alpha_1 \cdots s\alpha_k s$,

then $C = C's$, so $(t_i(C), p_i(C)) \neq (track(Csr_{3-i}), position(Csr_{3-i}))$, contrary to the assumption. Hence $\alpha_{k+1} = r_{3-i}$.

Suppose $track(C'sr_{3-i}) \neq t_i(C')$ or $jump_{3-i}(C'sr_{3-i})$ is *true*. In the first case, r_{3-i} changed from $t_i(C') = track(C')$ to $1 - track(C')$ and it stays on this track. In the second case, $track(C'sr_{3-i}sr_{3-i}) = 1 - track(C')$ and it stays on this track. In either case, this implies that $track(Csr_{3-i}) = 1 - track(C')$. Therefore, $t_i(C) = track(C') \neq track(Csr_{3-i})$, which is a contradiction.

Now suppose that $track(C'sr_{3-i}) = t_i(C')$ and $jump_{3-i}(C'sr_{3-i})$ is false. Since r_{3-i} did not change the track and $jump_{3-i}$ is false, $(t_{3-i}(C'), p_{3-i}(C')) = (track(C'), position(C'))$. Let $C'' = C_0 s\alpha_1 \cdots s\alpha_{k-1}$. Since $\alpha_k = r_i$, $(t_{3-i}(C''), t_{3-i}(C'')) = (t_{3-i}(C'), t_{3-i}(C'))$. Since the latter is $(track(C'), position(C')) = (track(C''sr_i), position(C''sr_i))$, this contradicts the minimality of α.

Therefore, in all cases, we reach the desired contradiction. ◀

The next lemma proves that the algorithm has the desired property.

▶ **Lemma 10.** *For every reachable configuration C, $\mathsf{v}(Csr_1) = \mathsf{v}(Csr_2)$.*

Proof. Suppose, for a contradiction, that this is not the case. Consider a shortest α from the initial configuration C_0 such that $\mathsf{v}(C_0\alpha sr_1) \neq \mathsf{v}(C_0\alpha sr_2)$. Let $C = C_0\alpha$ and write $\alpha = s\alpha_1 \cdots s\alpha_\ell$. Notice that α_ℓ is non-empty as, otherwise, $\alpha' = s\alpha_1 \cdots s\alpha_{\ell-1}$ is a shorter execution such that $\mathsf{v}(C_0\alpha' sr_1) \neq \mathsf{v}(C_0\alpha' sr_2)$. Moreover, $\alpha_\ell \notin \{r_1r_2, r_2r_1\}$ since, otherwise, $both(Cs) = 1$ and both r_1 and r_2 set the blackboard to $(1 - track(C), 0, 0, 0)$ from Cs. Hence, $\alpha_\ell = r_i$, for some $i \in \{1, 2\}$. Then $(t_i(C), p_i(C)) = (track(C), position(C))$. Let $C' = C_0 s\alpha_1 \cdots s\alpha_{\ell-1}$. There are two cases to consider.

Case 1: $jump_i(C)$ is true. By the pseudocode, since $jump_i(C)$ is true, r_i saw that $both(C's) = 0$, $track(C's) \neq t_i(C's)$, and it wrote $t_i(C) = track(C')$ and $p_i(C) = 1 - position(C')$. Since $track(C') \neq t_i(C's)$, it must be that $track(C's) = t_{3-i}(C's)$. It follows that the next step of r_{3-i} in Cs is to write $(1 - track(C), 0, 0, 0)$. However, this is precisely what r_i does in its next step from Cs as well since $jump_i$ is true and $(t_i(C), p_i(C)) = (track(C), position(C))$. This is a contradiction.

Case 2: $jump_i(C)$ is false. Then $both(C's) = 0$ and $track(C's) = t_i(C's)$. There are two subcases to consider:

- Suppose that either $position(C's) \neq p_i(C's)$ or $jump_i(C's)$ is true. Then r_{3-i} was the last reader to take a step before C', $t_{3-i}(C's) = track(C's)$, and $p_{3-i}(C's) = position(C's)$. Moreover, the next step of r_i from $C's$ is to write $(1-track(C's), 0, 0, 0)$. Thus, $track(C) \neq track(C')$. It follows that the next step of r_{3-i} from Cs is to write $(track(C), 1 - position(C), 0, 0)$. This is a contradiction.
- Suppose that $position(C's) = p_i(C's)$ and $jump_i(C's)$ is false. Since $both(C's) = 0$, C' cannot be the initial configuration and, thus, r_i was the last reader to take a step before C' and $\alpha_{\ell-1} = r_i$. By Lemma 9, $(t_{3-i}(C'), p_{3-i}(C')) \neq (track(C'), position(C'))$. By definition of α, $\mathsf{v}(C'sr_i) = \mathsf{v}(C'sr_{3-i})$. Thus, as r_i did not change the track, it must be that $track(C's) \neq t_{3-i}(C's)$ or ($position(C's) = p_{3-i}(C's)$ and $jump_{3-i}(C's)$ is false). Since $(t_{3-i}(C'), p_{3-i}(C')) \neq (track(C'), position(C'))$, it must be the former. Thus, r_{3-i}'s next step from Cs is to write $(track(C), 1 - position(C), 0, 0)$, which is exactly what r_i does. This is a contradiction.

In all cases, we reach the desired contradiction. ◀

▶ **Lemma 11.** *For any configuration C, $|\{\mathsf{v}(C\alpha) : \alpha$ is a $\{r_1, r_2\}$-only execution$\}| \leq 3$.*

Proof. For any configuration C, let α' be the longest $\{r_1, r_2\}$-only execution such that $C = C'\alpha'$. Notice that it suffices to prove that the claim holds for configuration C'. By our earlier assumption that s takes a step immediately after the initial configuration, we may assume that s took the last step before C'.

By Lemma 10, $\mathsf{v}(C'r_1) = \mathsf{v}(C'r_2)$. Subsequently, from $C'r_i$, the steps by r_i do not change the blackboard value. If r_{3-i} takes a step, then it sets *both* to 1 and $\mathsf{v}(C'r_1r_2) = \mathsf{v}(C'r_2r_1)$. After this, neither r_i or r_{3-i} can change the blackboard. Hence $\{\mathsf{v}(C'\alpha) : \alpha \text{ is a } \{r_1, r_2\}\text{-only execution}\} = \{\mathsf{v}(C'), \mathsf{v}(C'r_1), \mathsf{v}(C'r_1r_2)\}$. ◀

Finally, we prove that the algorithm is correct.

▶ **Lemma 12.** *In every execution, each reader returns the correct responses.*

Proof. Suppose not, so that there is some execution α from the initial configuration C_0 such that, in $C = C_0\alpha$, some r_i returns an incorrect response in its next step from C. Write $\alpha = \alpha' r_i \alpha''$, where α'' is a $\{s, r_{3-i}\}$-only execution. If s does not take any steps in α'', then r_i returns $false$, which is correct. So, s takes at least one step in α'' and r_i returns $false$ in its next step from C. By the pseudocode, this implies that $flag(C) = 0$ and $(t_i(C), p_i(C)) = (track(C), position(C))$. It follows that we may write $\alpha'' = \alpha'''sr_{3-i}$. However, this contradicts Lemma 9 from $C' = C_0\alpha'''$ with r_i. In particular, r_i returns $true$ in its next step from $C = C'sr_{3-i}$. ◀

References

1. Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic Snapshots of Shared Memory. *Journal of the ACM*, 40(4):873–890, 1993.
2. Zahra Aghazadeh and Philipp Woelfel. On the time and space complexity of ABA prevention and detection. In *Proceedings of the 34th SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 193–202, 2015.
3. Wojciech Golab. Recoverable Consensus in Shared Memory. arXiv preprint, 2018. `arXiv:1804.10597`.
4. Wojciech Golab and Danny Hendler. Recoverable Mutual Exclusion in Sub-logarithmic Time. In *Proceedings of the 36th SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 211–220, 2017. `doi:10.1145/3087801.3087819`.
5. Wojciech Golab and Aditya Ramaraju. Recoverable mutual exclusion. In *Proceedings of the 35th SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 65–74, 2016.
6. IBM. IBM system/370 extended architecture, principles of operation, 1983. Publication No. SA22-7085.
7. Prasad Jayanti and Srdjan Petrovic. Efficient and practical constructions of LL/SC variables. In *Proceedings of the 2nd SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 285–294, 2003.
8. Maged Michael. ABA prevention using single-word instructions. Technical report, IBM T. J. Watson Research Center, 2004.
9. Maged Michael. Practical Lock-Free and Wait-Free LL/SC/VL Implementations Using 64-Bit CAS. In Rachid Guerraoui, editor, *Proceedings of the 18th International Symposium on Distributed Computing (DISC)*, pages 144–158, 2004.
10. Mark Moir. Practical Implementations of Non-Blocking Synchronization Primitives. In *Proceedings of the 16th SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 219–228, 1997.

Progressive Algorithms for Domination and Independence

Grzegorz Fabiański
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland
grzegorz.fabianski@students.mimuw.edu.pl

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Sebastian Siebertz
Institute of Informatics, Humboldt-Universität zu Berlin, Germany
sebastian.siebertz@hu-berlin.de

Szymon Toruńczyk
Institute of Informatics, University of Warsaw, Poland
szymtor@mimuw.edu.pl

Abstract

We consider a generic algorithmic paradigm that we call *progressive exploration*, which can be used to develop simple and efficient parameterized graph algorithms. We identify two model-theoretic properties that lead to efficient progressive algorithms, namely variants of the *Helly property* and *stability*. We demonstrate our approach by giving linear-time fixed-parameter algorithms for the Distance-r Dominating Set problem (parameterized by the solution size) in a wide variety of restricted graph classes, such as powers of nowhere dense classes, map graphs, and (for $r = 1$) biclique-free graphs. Similarly, for the Distance-r Independent Set problem the technique can be used to give a linear-time fixed-parameter algorithm on any nowhere dense class. Despite the simplicity of the method, in several cases our results extend known boundaries of tractability for the considered problems and improve the best known running times.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability; Theory of computation → Finite Model Theory

Keywords and phrases Dominating Set, Independent Set, nowhere denseness, stability, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.27

Related Version A full version of the paper is available at [10], `https://arxiv.org/abs/1811.06799`.

Funding The work of M. Pilipczuk and S. Siebertz is supported by the National Science Centre of Poland via POLONEZ grant agreement UMO-2015/19/P/ST6/03998, which has received funding from the European Union's Horizon 2020 research and innovation programme (Marie Skłodowska-Curie grant agreement No. 665778). G. Fabiański and Sz. Toruńczyk are supported by the NCN grant 2016/21/D/ST6/01485.

Acknowledgements We thank Ehud Hrushovski for pointing us to the nfcp property.

1 Introduction

It is widely believed that many important algorithmic graph problems cannot be solved efficiently on general graphs. Consequently, a natural question is to identify the most general classes of graphs on which a given problem can be solved efficiently. Structural graph theory offers a wealth of concepts that can be used to design efficient algorithms for generally

intractable problems on restricted graph classes. An important result in this area states that every property of graphs expressible in monadic second-order logic can be tested in linear time on every class of bounded treewidth [3]. Similarly, every property expressible in first-order logic can be tested in almost linear time on every nowhere dense graph class [11].

Nowhere denseness is an abstract notion of uniform sparseness in graphs, which is the foundational definition of the theory of sparse graphs; see the monograph of Nešetřil and Ossona de Mendez [14] for an overview. Formally, a graph class $\mathscr{C}$ is *nowhere dense* if for every $r \in \mathbb{N}$, one cannot obtain arbitrary large cliques by contracting disjoint connected subgraphs of radius at most r in graphs from $\mathscr{C}$. Many well-studied classes of sparse graphs are nowhere dense, for instance the class of planar graphs, any class of graphs with a fixed bound on the maximum degree, or any class of graphs excluding a fixed (topological) minor, are nowhere dense. Furthermore, under certain closure conditions, nowhere denseness constitutes the frontier of parameterized tractability for natural classes of problems. For instance, while the first-order model-checking problem is fixed-parameter tractable on every nowhere dense class $\mathscr{C}$ [11], on every subgraph-closed class $\mathscr{D}$ that is not nowhere dense, it is as hard as on the class of all graphs [5, 8]. Similar lower bounds are known for many individual problems, e.g. for the Distance-r Independent Set problem and the Distance-r Dominating Set problem, on subgraph-closed classes which are not nowhere dense [7, 16].

Towards the goal of extending the border of algorithmic tractability for the above mentioned problems beyond graph classes that are closed under taking subgraphs, we study a very simple and generic algorithmic paradigm that we call *progressive exploration*, described below.

This general idea can be applied to a *parameterized subset problem*, in which we are given a graph G and parameter k, and the goal is to determine if G satisfies a property of the form *"there exists a candidate of size k which agrees with every witness"*. Here, the notions of *candidates, witnesses,* and *agreeing* depend on the problem under consideration. For example, when considering the existence of a distance-r dominating set of size k, candidates are sets S of size k, witnesses are single vertices, and a candidate S *agrees* with every vertex in distance at most r from a vertex in S.

Another way of viewing a problem as above is to consider the bipartite "agreement" graph, whose left part consists of candidates, right part consists of witnesses, and adjacency denotes agreeing. Then the problem is to determine if the right part of this bipartite graph has some common neighbor. Note that the size of the bipartite graph is usually polynomial in terms of the size of the input graph G, where the exponent of the polynomial is the parameter k that we are interested in. In particular, if we are aiming at fixed-parameter tractability, we cannot even afford constructing the entire bipartite graph.

To solve a problem as above, a progressive exploration algorithm proceeds in rounds $i = 1, 2, \ldots$, where each round i finishes with constructing a candidate S_i and a set of witnesses W_i. In round i, we seek a candidate S_i that agrees with every vertex in the union of the previously constructed witness sets $W_1, W_2, \ldots, W_{i-1}$. If no such S_i exists, we can terminate and answer that there is no solution. On the other hand, if the found candidate S_i agrees with every witness in the graph, we also terminate and return it as a solution. Otherwise, we find another set W_i of witnesses, such that each of the candidates $S_1, \ldots, S_{i-1}, S_i$ found so far does not agree with one of the witnesses in W_i, and proceed to the next round. In this way, we progressively explore the whole solution space, while constructing more and more problematic witness sets that the future candidates must agree with, until we either find a solution or enough witnesses to certify that no solution exists.

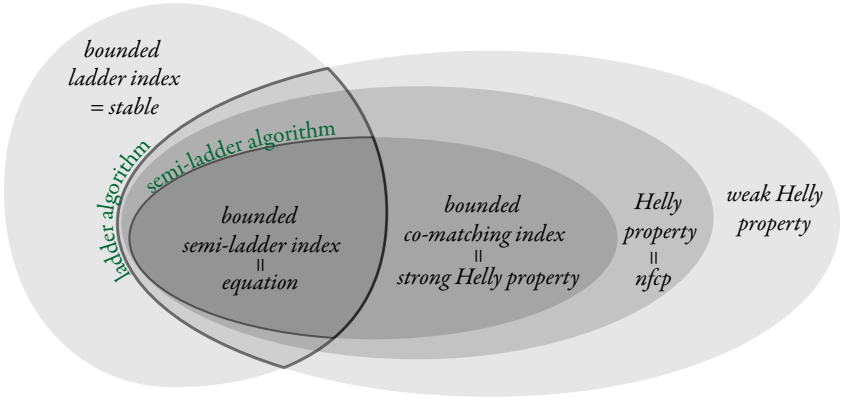


Figure 1 The figure depicts various properties of classes of bipartite graphs, which are introduced in Section 2. Domination- and independence-type problems studied in Section 3 reduce to the problem of determining whether the right part of a given bipartite graph has a common neighbor. In Section 4 two algorithms for the latter problem are devised, and their domains of applicability are marked above. The *ladder algorithm* has a larger domain, but requires a more powerful access oracle and has higher running time. Finally, we apply these algorithms to specific graph classes, yielding new fixed-parameter tractability results for domination- and independence-type problems.

Progressive graph exploration is a generic approach to solving graph problems, which so far rather resembles a wishful-thinking heuristic than a viable algorithmic methodology. Such algorithms can be applied to any input graph, however, a priori there are multiple problem-dependent details to be filled.

First, in order to implement the iteration, we should efficiently compute candidates S_i and small witness sets W_i in every round. Second, to analyze the running time we need to give an upper bound on the number of rounds in which the algorithm terminates. If we can guarantee that each round can be implemented efficiently and that the number of rounds is bounded in the parameter k, we immediately obtain a fixed-parameter algorithm for the problem under consideration.

In this work we study properties of algorithmic problems and graphs that ensure these desired features. Inspired by notions from model theory, we identify combinatorial properties of the arising bipartite graphs which lead to efficient progressive exploration algorithms. The properties that guarantee the existence of small witness sets are variants of the *Helly property*, called *nfcp* in model theory. The property that guarantees that the progressive exploration algorithms stop after a bounded number of rounds is the model-theoretic notion of *stability*. Under these conditions, for problems formulated using short distances in the graph, we are able to implement progressive exploration efficiently, yielding fast and simple fixed-parameter algorithms. See the caption in Figure 1 for an overview of the paper.

We demonstrate our approach by applying it to the Distance-r Dominating Set problem and the Distance-r Independent Set problem on a variety of restricted graph classes. More precisely, we prove that:

- For every $r \in \mathbb{N}$ and graph class $\mathscr{C}$ that is either nowhere dense, or is a power of a nowhere dense class, or is the class of map graphs, the Distance-r Dominating Set problem on any graph $G \in \mathscr{C}$ can be solved in time $2^{\mathcal{O}(k \log k)} \cdot \|G\|$. Here and throughout the paper, $\|G\|$ denotes the total number of vertices and edges in a graph G.
- For every $t \in \mathbb{N}$, the Dominating Set problem on any $K_{t,t}$-free graph G can be solved in time $2^{\mathcal{O}(k \log k)} \cdot \|G\|$; here, a graph is $K_{t,t}$-free if it does not contain the complete bipartite graph $K_{t,t}$ as a subgraph.

- For every $r \in \mathbb{N}$ and nowhere dense graph class $\mathscr{C}$, the Distance-r Independent Set problem on any graph $G \in \mathscr{C}$ can be solved in time $f(k) \cdot \|G\|$, for some function f.

Actually, for the last result, we also give a different algorithm with running time $2^{\mathcal{O}(k \log k)} \cdot \|G\|$, which however does not rely on the concept of progressive exploration and uses some black-boxes from the theory of sparsity.

Our results extend the limits of tractability for Distance-r Dominating Set and Distance-r Independent Set and, in some cases, improve the best known running times. We include a comprehensive comparison with the existing literature at the end of Section 4. However, let us stress here a key point: all our algorithms are derived in a generic way using the idea of progressive exploration, hence they are very easy to implement and they do not use any algorithmic black-boxes that depend on the class from which the input is drawn. In fact, properties of the considered classes are used only when analyzing the running time.

2 Complexity-measures for bipartite graphs

In this section, we define the basic notions used in this paper, related to various complexity measures associated with bipartite graphs. We work with the following notion of bipartite graphs that is not standard in graph theory but suits the model theoretic context very well.

A *bipartite graph* is a triple $G = (L, R, E)$, where L and R are two sets of vertices, called the *left part* and *right part*, respectively, and $E \subseteq L \times R$ is a binary relation whose elements are called *edges*. Hence, bipartite graphs with parts L, R correspond bijectively to binary relations with domain L and codomain R. Note that each bipartite graph has a uniquely determined left and right part. Also, those parts are not necessarily disjoint.

Ladders, semi-ladders, and co-matchings. We now define various complexity measures for bipartite graphs, based on the size of a largest "obstruction" found in a given bipartite graph. There are several types of obstructions, leading to different complexity measures. We start with defining the various types of obstructions. Let $G = (L, R, E)$ be a bipartite graph. Two sequences, $a_1, \ldots, a_n \in L$ and $b_1, \ldots, b_n \in R$, form:

- a *co-matching* of order n in G if we have $(a_i, b_j) \in E \iff i \neq j$, for all $i, j \in \{1, \ldots, n\}$;
- a *ladder* of order n in G if we have $(a_i, b_j) \in E \iff i > j$, for all $i, j \in \{1, \ldots, n\}$; and
- a *semi-ladder* of order n in G if $(a_i, b_j) \in E$ for all $i, j \in \{1, \ldots, n\}$ with $i > j$, and $(a_i, b_i) \notin E$ for all $i \in \{1, \ldots, n\}$.

Note that in case of a semi-ladder we do not impose any condition for $i < j$. Observe that any ladder of order n and any co-matching of order n are also semi-ladders of order n.

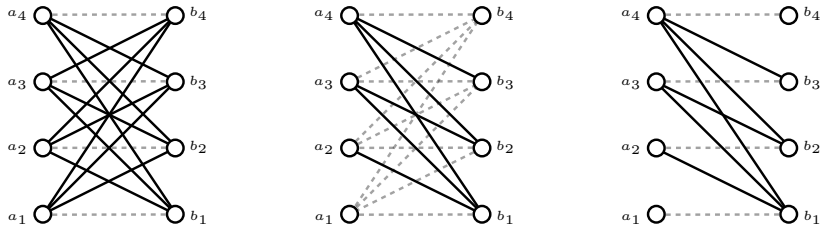


Figure 2 A co-matching, a ladder, and a semi-ladder of order 4, respectively. Dashed gray lines represent non-edges.

The *co-matching index* of a bipartite graph is the maximum order of a co-matching that it contains. A class of bipartite graphs has *bounded* co-matching index if the supremum of the co-matching indices of its members is finite. We define analogous notions for the *ladder index* and the *semi-ladder index*, in the expected way.

In this paper, we will often not care about the precise bounds on the indices of graphs, and it will only matter whether the respective index is bounded in a given class. Bounded ladder and semi-ladder indices correspond notions from model theory (see paragraph on formulas below). We will later relate the property of having a bounded co-matching index to a variant of the *Helly property*. Using a simple Ramsey argument, we now observe that boundedness of the semi-ladder index is equivalent to boundedness of both the co-matching and the ladder index. Let us first state Ramsey's theorem in the form used in this paper.

▶ **Theorem 1** (Ramsey's theorem). *For all $c, \ell \in \mathbb{N}$ there exists a number $R^c(\ell)$ with the following property. If the edges of a complete graph on $R^c(\ell)$ vertices are colored using c colors, then there is a set of ℓ vertices which is* monochromatic*, that is, all edges with both endpoints in this set are of the same color.*

The standard proof of Ramsey's theorem yields an upper bound of $R^c(\ell) \leqslant c^{c\ell-1}$ for $c \geqslant 2$. From now on, we adopt the notation $R^c(\ell)$ for the multicolored Ramsey numbers as described in Theorem 1. The proofs of all statements marked with ♠ can be found in the full version of the paper [10].

▶ **Lemma 2** (♠). *A class of bipartite graphs has finite semi-ladder index if and only if both its ladder index and its co-matching index are finite.*

Helly property and its variants. Let $p \in \mathbb{N}$ and let $G = (L, R, E)$ be a bipartite graph. We say that a subset $B \subseteq R$ is *covered* by a subset $A \subseteq L$ if there exists a vertex $a \in A$ which is adjacent to all the vertices of B. Then subsets A and B have the *p-Helly property* if either B is covered by A, or B contains a subset of size at most p that is not covered by A. We shall say that G has:

- the *weak p-Helly property* if L and R have the p-Helly property;
- the *p-Helly property* if L and B have the p-Helly property, for all $B \subseteq R$; and
- the *strong p-Helly property* if all $A \subseteq L$ and $B \subseteq R$ have the p-Helly property.

We say that a class $\mathscr{C}$ of bipartite graphs has the (weak/strong) Helly property if there is some $p \in \mathbb{N}$ such that all graphs in $\mathscr{C}$ have the (weak/strong) p-Helly property.

It turns out that the strong p-Helly property corresponds precisely to having co-matching index at most p.

▶ **Lemma 3** (♠). *Let $p \in \mathbb{N}$ and G be a bipartite graph. Then G has the strong p-Helly property if and only if it has co-matching index at most p.*

In the following paragraphs we will see specific examples of classes of bipartite graphs satisfying variants of the (weak/strong) Helly property.

Bipartite graphs defined by formulas. We construct bipartite graphs using logical formulas. In principle, we could consider formulas of any logic, but in this paper we only consider first-order logic in the vocabulary of graphs, i.e., using the binary relation symbol E representing edges, the binary relation symbol $=$ representing equality, and logical constructs $\vee, \wedge, \neg, \rightarrow, \forall, \exists$. E.g, the property $\mathrm{dist}(x, y) \leqslant 5$, expressing that x and y are at distance at most 5 in a graph G, can be expressed by a first-order formula using four existential quantifiers.

We write $\bar{x}$ to represent a tuple of variables (with every variable appearing only once). If V is a set, then $V^{\bar{x}}$ denotes the set of all assignments mapping variables in $\bar{x}$ to V. Let $\varphi(\bar{x}; \bar{y})$ be a formula with free variables partitioned into two disjoint tuples, $\bar{x}$ and $\bar{y}$. Given any graph G with vertex set V, the formula φ induces a bipartite graph $\varphi(G)$ with left

part $V^{\bar{x}}$, right part $V^{\bar{y}}$, and where $\bar{a} \in V^{\bar{x}}$ and $\bar{b} \in V^{\bar{y}}$ are adjacent if and only if $\varphi(\bar{a}; \bar{b})$ holds in G. If $\mathscr{C}$ is a class of graphs and $\varphi(\bar{x}; \bar{y})$ is a formula, then by $\varphi(\mathscr{C})$ we denote the class of all bipartite graphs $\varphi(G)$, for $G \in \mathscr{C}$. We say that φ has *bounded ladder index* on a class $\mathscr{C}$ if the class $\varphi(\mathscr{C})$ has bounded ladder index; similarly for the co-matching and the semi-ladder index. The same definitions apply if instead of graphs we consider logical structures over some fixed signature, and $\varphi(\bar{x}; \bar{y})$ is a formula over that signature. For simplicity, we consider only graphs in this paper.

We note that the various indices are preserved by adding spurious free variables to formulas. Precisely, let $\varphi(\bar{x}; \bar{y})$ be a first-order formula and let $\varphi'(\bar{x}'; \bar{y}')$ be the same formula, but having extra free variables, i.e., $\bar{x}$ is a subtuple of $\bar{x}'$ and $\bar{y}$ is a subtuple of $\bar{y}'$. Then, for any graph G, the ladder index of $\varphi(G)$ is equal to the ladder index of $\varphi'(G)$, although the two bipartite graphs $\varphi(G)$ and $\varphi'(G)$ may differ. The same holds for all the other properties studied in this paper: co-matching index, semi-ladder index, (weak/strong) Helly property.

The next lemma shows that a positive boolean combination (a boolean combination in which no negations are used) of formulas with bounded semi-ladder index also has bounded semi-ladder index. The proof uses a Ramsey argument.

▶ **Lemma 4 (♠).** *Let $\varphi_1(\bar{x}; \bar{y}), \ldots, \varphi_k(\bar{x}; \bar{y})$ be formulas and let $\psi(\bar{x}; \bar{y})$ be a positive boolean combination of $\varphi_1, \ldots, \varphi_k$. Suppose G is a graph such that $\varphi_1(G), \ldots, \varphi_k(G)$ have semi-ladder index smaller than ℓ. Then $\psi(G)$ has semi-ladder index smaller than $R^k(\ell)$.*

We remark that the property of having bounded ladder index is preserved by taking arbitrary boolean combinations, not just positive ones. Finally, the analogue of Lemma 4 fails for the co-matching index if positive boolean combinations are considered, but still holds if we restrict attention to conjunctions of atomic formulas.

We will later need the following variant of Lemma 4, which provides a sharper bound for formulas of a special form. As usual in this work, the proof relies on a Ramsey-like argument.

▶ **Lemma 5 (♠).** *Let $\psi(\bar{x}; \bar{y}) = \bigvee_{j=1}^{k} \varphi(\bar{x}^j; \bar{y})$ for some $k \geqslant 2$, where $\varphi(\bar{x}; \bar{y})$ is a formula and $\bar{x}^1, \ldots, \bar{x}^k$ are permutations of $\bar{x}$. Suppose G is a graph such that $\varphi(G)$ has semi-ladder index smaller than ℓ. Then $\psi(G)$ has semi-ladder index smaller than $k^{\ell-1}$.*

Stability theory. Many of the combinatorial properties of bipartite graphs introduced above correspond to properties of formulas studied in model theory, specifically, in its modern branch called *stability theory*. Very roughly, stability theory studies how various combinatorial obstructions affect the logical complexity of the considered first-order theory. Stability theory will not be used in this paper, but for bibliographic completeness, we present a dictionary relating the notions introduced above to the notions studied there. Fix a class of structures $\mathscr{C}$; in model theory; usually $\mathscr{C}$ is the class of all models of a fixed first-order theory. A first-order formula $\varphi(\bar{x}; \bar{y})$ is called *stable* (with respect to $\mathscr{C}$) if $\varphi(\mathscr{C})$ has bounded ladder index [21, Chapter I.2]. It is called *nfcp* if $\varphi(\mathscr{C})$ has the Helly property [21, Chapter II.4]. It is called an *equation* if $\varphi(\mathscr{C})$ has bounded semi-ladder index [19, 15].

We also remark that the properties of bipartite graphs that we consider can be viewed as properties of set systems: a bipartite graph $G = (L, R, E)$ gives rise to a family $\mathcal{F}$ of subsets of R, consisting of neighborhoods of elements of L. The p-Helly property is usually formulated for set systems $\mathcal{F}$, requiring that every minimal subfamily of $\mathcal{F}$ with an empty intersection has at most p sets in it. The semi-ladder index corresponds to the maximal length of an inclusion-chain in the family of intersections of the sets in $\mathcal{F}$.

Stability in nowhere dense graph classes. The classes of bipartite graphs with bounded ladder index that are relevant to this paper are provided by the following result, due to Podewski and Ziegler [20].

▶ **Theorem 6** ([20], cf. [1, 18]). *Let $\mathscr{C}$ be a nowhere dense class of graphs and let $\varphi(\bar{x};\bar{y})$ be a first-order formula. Then φ has bounded ladder index on $\mathscr{C}$.*

The above result was originally stated for superflat graphs, and using the notion of stability. The proof relied on non-constructive model-theoretic methods, such as the compactness theorem. The connection with nowhere denseness was observed by Adler and Adler [1], and a proof providing explicit bounds was given by the last three authors in [18]. The observation of Adler and Adler is the starting point of this work, as it brings to light the connection between model theory and computer science which is further studied in this paper.

3 Domination and independence problems

We consider subset problems, where in a given graph we look for a solution S of size k that satisfies some property, whose dissatisfaction can be witnessed by a small subset of vertices W. Moreover, checking a candidate solution S against a witness W can be expressed in first-order logic. Thus, a problem of interest can be expressed by a sentence of the form $\exists_{\bar{x}}\forall_{\bar{y}}\varphi(\bar{x};\bar{y})$, for a suitable formula $\varphi(\bar{x};\bar{y})$, where $\bar{x}$ is a tuple of k variables that represent a candidate S, while $\bar{y}$ is a tuple of ℓ variables that represent a witness W.

▶ **Example 7.** The Distance-r Dominating Set problem for parameter k can be expressed as above using the formula $\delta_r^k(\bar{x};y) = \bigvee_{i=1}^{k} \delta_r(x_i,y)$, where $\delta_r(x,y)$ is a formula that checks whether $\mathrm{dist}(x,y) \leqslant r$, and $\bar{x} = (x_1,\ldots,x_k)$. Similarly, the Distance-r Independent Set problem for parameter k can be expressed using the formula $\eta_r^k(\bar{x};y) = \bigwedge_{1\leqslant i<j\leqslant k} \eta_r(x_i,x_j,y)$, where $\eta_r(x,x',y)$ is a formula that checks whether $\mathrm{dist}(x,y)+\mathrm{dist}(x',y) > r$.

Observe that a graph G satisfies the sentence $\exists_{\bar{x}}\forall_{\bar{y}}\varphi(\bar{x};\bar{y})$ if and only if the right part of the bipartite graph $H = \varphi(G)$ is covered by the left side, i.e., all vertices in the right part (witnesses) have a common neighbor (solution). We call this abstract problem – checking whether the right part of a given bipartite graph H is covered by the left side – the Coverage problem.

Note that the size of the bipartite graph $\varphi(G)$ is polynomial in the size of G, where the exponent depends on the number of free variables in φ, which is usually the parameter we are interested in. As we are aiming at fixed-parameter algorithms, we cannot afford to even construct the whole bipartite graph $\varphi(G)$. Therefore, we will design algorithms that solve the Coverage problem using an oracle access to the bipartite graph graph $H = \varphi(G)$, where the oracle calls will be implemented using subroutines on the original graph G. The running time of these algorithms, expressed in terms of the number of oracle calls, will be bounded only in terms of quantities (ladder indices, numbers governing Helly property, etc.) related to the class of bipartite graphs $\varphi(\mathscr{C})$, where $\mathscr{C}$ is the considered class of input graphs.

Therefore, to obtain an algorithm for solving the initial problem on a given graph class $\mathscr{C}$ we proceed in two steps:

- Prove that the class $\varphi(\mathscr{C})$ has a suitable Helly-type property and bounded ladder index.
- Design an algorithm for Coverage, for input bipartite graphs with suitable Helly-type properties and bounded ladder index, that uses only a bounded number of oracle calls.

In Section 4 we give two such algorithms solving Coverage: the *Semi-ladder Algorithm*, and the *Ladder Algorithm*. The Semi-ladder Algorithm requires that H has bounded semi-ladder index, whereas the Ladder Algorithm requires that H has bounded ladder index and the

weak p-Helly property, for some fixed p. Note that by Lemmas 2 and 3, boundedness of the semi-ladder index is equivalent to boundedness of the ladder index and having the strong p-Helly property, for some fixed p, so the prerequisites for the Semi-ladder Algorithm are stronger than for the Ladder Algorithm. See Figure 1 for an overview.

We postpone the discussion of the algorithms to Section 4, and for now we focus on exhibiting the suitable properties for various classes of bipartite graphs. Slightly more precisely, we prove that on certain graph classes, formulas corresponding to domination-type problems have bounded semi-ladder index, while those corresponding to independence-type problems have the weak Helly property and bounded ladder index. Hence, in the first case we will apply the Semi-ladder Algorithm, and in the second – the Ladder Algorithm.

Distance formulas and domination-type problems. We shall prove fixed-parameter tractability results not only for distance-r domination, but for a more general class of domination-type problems. Those can be expressed by suitable formulas, as explained next.

▶ **Definition 8.** *For $r \in \mathbb{N}$, let $\delta_r(x,y)$ be the formula checking whether $dist(x,y) \leqslant r$. A* distance formula *is a formula $\varphi(\bar{x};\bar{y})$ which is a boolean combination of* atoms *of the form $\delta_r(x,y)$, where the variable x occurs in $\bar{x}$, the variable y occurs in $\bar{y}$, and $r \in \mathbb{N}$ is any number. The* radius *of a distance formula is the maximal number r occurring in its atoms, whereas its* size *is the number of atoms occurring in it. A distance formula is* positive *if it is a positive boolean combination of atoms.*

A domination-type property *is a sentence ψ of the form $\exists_{\bar{x}}\forall_{\bar{y}}\,\varphi(\bar{x};\bar{y})$, where φ is a positive distance formula. A* domination-type problem *is the computational problem of determining whether a given graph G satisfies a given domination-type property.*

▶ **Example 9.** Fix $r \in \mathbb{N}$ and let $\bar{x} = (x_1, \ldots, x_k)$ be a k-tuple of variable. Then the formula $\delta_r^k(\bar{x};y)$ considered in Example 7 is a positive distance formula, hence the problem defined by the domination-type property $\exists_{\bar{x}}\forall_{\bar{y}}\,\delta_r^k(\bar{x};y)$ (aka Distance-r Dominating Set) is a domination-type problem. Similarly, formulas $\varphi(\bar{x};y)$ expressing the following properties give raise to natural domination-type problems:

- y is at distance at most r from at least two of the vertices $x_1, \ldots, x_k$; and
- the sum $\mathrm{dist}(x_1,y) + \mathrm{dist}(x_2,y) + \ldots + \mathrm{dist}(x_k,y)$ is at most r.

On the other hand, the formula $\eta_r^k(\bar{x};y)$ considered in Example 7 is a distance formula, but it is not positive, and hence it does not yield a domination-type property.

From Lemmas 4 and 5 and the remark about spurious variables not affecting the semi-ladder index, we immediately obtain the following.

▶ **Corollary 10.** *Let $\varphi(\bar{x};\bar{y})$ be a positive distance formula of radius r and size s. If G is a graph such that the semi-ladder index of $\delta_q(G)$ is smaller than ℓ for all $q \leqslant r$, then the semi-ladder index of $\varphi(G)$ is smaller than $R^s(\ell)$. Moreover, if $\varphi = \delta_r^k$ as defined in Example 7 and $k \geqslant 2$, then the semi-ladder index of $\varphi(G)$ is smaller than $k^{\ell-1}$.*

Domination problems. We first consider domination-type problems and prove that they have bounded semi-ladder indices on any nowhere dense class. This result can actually be extended beyond nowhere denseness: to powers of nowhere dense classes, to map graphs, and to $K_{t,t}$-free graphs for radius $r = 1$. We define the former two concepts next.

For a graph G and $s \in \mathbb{N}$, let G^s denote the graph with the same vertex set as G, where two vertices are adjacent if and only if their distance in G is at most s. If $\mathscr{D}$ is a graph class, then $\mathscr{D}^s$ denotes the class $\{G^s : G \in \mathscr{D}\}$. Note that a power of a nowhere dense class is not necessarily nowhere dense, e.g., the square of the class of stars is the class of complete graphs.

A graph G is a map graph if one can assign to each vertex of G a closed, arc-connected region in the plane so that the interiors of regions are pairwise disjoint and two vertices of G are adjacent if and only if their regions share at least one point on their boundaries. Note that map graphs are not necessarily planar and may contain arbitrarily large cliques, as four or more regions may share a single point on their boundaries.

The following result will be used in the next section to obtain fixed-parameter tractability of domination-type problems over graph classes described above.

▶ **Theorem 11** (♠). *For any $r \in \mathbb{N}$ and nowhere dense graph class $\mathscr{C}$, the formula $\delta_r(x, y)$ has bounded semi-ladder index on $\mathscr{C}$. The same holds also when $\mathscr{C} = \mathscr{D}^s$ for some nowhere dense class $\mathscr{D}$ and $s \in \mathbb{N}$, when $\mathscr{C}$ is the class of map graphs, and when $r = 1$ and $\mathscr{C}$ is the class of $K_{t,t}$-free graphs, for any fixed $t \in \mathbb{N}$.*

For the case when $\mathscr{C}$ is nowhere dense we utilize the well-known characterization of nowhere denseness via *uniform quasi-wideness* [13], which we recall below.

▶ **Definition 12.** *We say that a graph class $\mathscr{C}$ is* uniformly quasi-wide *if for all $r \in \mathbb{N}$, there are $s_r \in \mathbb{N}$ and $N_r : \mathbb{N} \to \mathbb{N}$ such that for all $k \in \mathbb{N}$, every graph $G \in \mathscr{C}$, and every vertex subset $W \subseteq V(G)$ of size larger than $N_r(k)$, there exist disjoint vertex subsets $S \subseteq V(G)$ and $A \subseteq W$ such that $|S| \leqslant s_r$, $|A| > k$, and A is distance-r independent in $G - S$.*

▶ **Theorem 13** ([12, 14]). *A graph class $\mathscr{C}$ is nowhere dense if and only if it is uniformly quasi-wide.*

The nowhere dense case of Theorem 11 is therefore encapsulated in the following lemma.

▶ **Lemma 14** (♠). *For every $r \in \mathbb{N}$ and uniformily quasi-wide class $\mathscr{C}$, the class $\delta_r(\mathscr{C})$ has bounded semi-ladder index.*

We now give a very rough sketch the proof of Lemma 14. If for some $G \in \mathscr{C}$ the graph $\delta_r(G)$ has semi-ladder index ℓ, then in G we have vertices $a_1, \ldots, a_\ell$ and $b_1, \ldots, b_\ell$ such that $\mathrm{dist}(a_i, b_j) \leqslant r$ for all $i > j$ and $\mathrm{dist}(a_i, b_i) > r$ for all i. Then provided ℓ is huge, by uniform quasi-wideness, we can find a large subset $A \subseteq \{a_1, \ldots, a_\ell\}$ of vertices that "communicate" with each other only through a set S of constant size – all paths of length at most $2r$ between vertices of A pass through S. Now the vertices from A have pairwise different distance-r neighborhoods within $\{b_1, \ldots, b_\ell\}$, but only a limited number of possible interactions with S (measured up to distance r). This quickly leads to a contradiction if A is large enough. The cases when $\mathscr{C}$ is a power of a nowhere dense class and when $\mathscr{C}$ is the class of map graphs follow as simple corollaries from the result for nowhere dense classes. The case when $\mathscr{C}$ is the class of $K_{t,t}$-free graphs is a simple observation: a large semi-ladder in $\delta_1(G)$ enforces a large biclique in G.

We remark that the above argument is similar to the reasoning that shows that graphs from a fixed nowhere dense class admit small *distance-r domination cores*: subsets of vertices whose distance-r domination forces distance-r domination of the whole graph. This property was first proved implicitly by Dawar and Kreutzer in their FPT algorithm for Distance-r Dominating Set on any nowhere dense class [4], also using uniform quasi-wideness. We refer the reader to [17, Chapter 3, Section 5] for an explicit exposition.

Having established boundedness of the semi-ladder index of $\delta_r(x,y)$ on a class $\mathscr{C}$, we can use Corollary 10 to extend this to any positive distance formula. Therefore, by Theorem 11, Corollary 10, and Lemma 3 we immediately obtain the following.

▶ **Corollary 15.** *Let $\mathscr{C}$ and r be as in Theorem 11 and let $\varphi(\bar{x};\bar{y})$ be a positive distance formula of radius at most r. Then the class $\varphi(\mathscr{C})$ has bounded semi-ladder index, so in particular it has the strong Helly property.*

In fact, Corollary 10 provides a better control of the semi-ladder index of $\varphi(\mathscr{C})$ in terms of the semi-ladder index of $\delta_r(\mathscr{C})$ and the size of φ. In the next section we will use these more refined bounds for a precise analysis of the running times.

Note that Corollary 15 does not generalize to arbitrary first-order formulas. Indeed, if $\mathscr{C}$ is the class of all edgeless graphs and $\varphi(x;y)$ is the formula $x \neq y$, then $\varphi(\mathscr{C})$ is the class of all complements of matchings, which does not even have the weak Helly property.

Independence problems. We now move to the Distance-r Independent Set problem: deciding whether a given graph contains a distance-r independent set of size k. This property is most naturally expressed using an existential sentence, and not as a sentence of the form $\exists_{\bar{x}}\forall_{\bar{y}}\varphi(\bar{x};\bar{y})$. However, in Example 7 we gave a suitable formula $\eta_r^k(\bar{x};y)$ that expresses the problem: the trick is to phrase the property that $x_1,\ldots,x_k$ are pairwise at distance more than r by saying that for every vertex y, for all $1 \leqslant i < j \leqslant k$ the sum of distances from y to x_i and x_j is larger than r. Thus, a vertex y that does not satisfy this condition may serve as a witness that a given tuple $\bar{x}$ does not form a distance-r independent set.

In the full version of the paper we prove the following.

▶ **Theorem 16 (♠).** *Let $\mathscr{C}$ be a nowhere dense class of graphs and let $k,r \in \mathbb{N}$. Then the class $\eta_r^k(\mathscr{C})$ has the weak p-Helly property, for some $p \in \mathbb{N}$ depending on k, r, and $\mathscr{C}$.*

It is easy to see that for any $k \geqslant 2$ and $r \geqslant 1$, the formula $\eta_r^k(x;y)$ does not have the strong Helly property on the class $\mathscr{C}$ of edgeless graphs. Thus, in general we cannot hope for boundedness of the semi-ladder index of $\eta_r^k(\mathscr{C})$ and use the Semi-Ladder Algorithm.

The proof of Theorem 16 is actually very different from the proof of Theorem 11, and presents a novel contribution of this work. Instead of uniform quasi-wideness, we use the characterization of nowhere denseness via the *Splitter game* [11]. The idea is that in case a graph $G \in \mathscr{C}$ does not have a distance-r independent set of size k, there is a small witness of this: a set W of size bounded in terms of k, r, and $\mathscr{C}$ such that for every vertex subset S of size k, some path of length at most r connecting two vertices of S crosses W. This exactly corresponds to the notion of witnessing expressed by η_r^k. Such a witness W is constructed recursively along Splitter's strategy tree in the Splitter game in G. We use the condition that G does not have a distance-r independent set of size k to prove that we can find a small (in terms of $k,r,\mathscr{C}$) set of "representative" moves of the Connector. Trimming the strategy tree to those moves bounds its size in terms of $k,r,\mathscr{C}$, yielding the desired upper bound on the witness size.

We remark that our proof of Theorem 16 can actually be turned into an algorithm for the Distance-r Independent Set problem on any nowhere dense class $\mathscr{C}$ with running time of $2^{\mathcal{O}(k\log k)} \cdot \|G\|$. However, this algorithm is much more complicated than the Ladder algorithm that we explain in the next section, and in particular it uses some black-box results from the theory of nowhere dense graph classes.

4 Algorithms

In this section, we present two algorithms solving the Coverage problem on a given bipartite graph. The bipartite graph can be only accessed via restricted access oracles; we start with presenting this model. We then describe the two algorithms. Then we show how the oracles can be efficiently implemented in the special cases relevant to this paper. This, in combination with the results from the previous section, will allow us to obtain the desired algorithmic statements about domination and independence problems on restricted graph classes.

The access oracles. We consider the following model of an algorithmic search for a solution in a bipartite graph representing the search space. Consider a bipartite graph $H = (L, R, E)$, where the left side L is the set of *candidates* and the right side R is the set of *witnesses*. An edge between a candidate $a \in L$ and a witness $b \in R$ is interpreted as that a and b *agree*: b agrees that a is a solution. Expressed in those terms, Coverage is the problem of finding a *solution*: a candidate which agrees with all witnesses. We will use the terminology of candidates, witnesses, solutions, and agreeing as explained above, as this facilitates the understanding of the algorithms for Coverage in terms of the original problems.

As we explained, the considered bipartite graph H will typically be of the form $\varphi(G)$ for some formula $\varphi(\bar{x}; \bar{y})$ expressing the considered problem. Thus, H shall represent the whole search space, so we allow our algorithms a restricted access to H via the following *oracles*.

Candidate Oracle: Given a set of witnesses $B \subseteq R$, the oracle either returns a candidate $a \in L$ that agrees with all witnesses of B, or concludes that no such candidate exists.

Weak Witness Oracle: Given a candidate $a \in L$, the oracle either concludes that a is a solution, or returns a witness $b \in R$ that does not agree with a.

Strong Witness Oracle: Given a set of candidates $A \subseteq L$ and a number $p \in \mathbb{N}$, the oracle either finds a set of witnesses $P \subseteq R$ such that $|P| \leqslant p$ and every candidate of A does not agree with some witness from P, or concludes that no such set P exists.

Note that the Weak Witness Oracle can be simulated by the Strong Witness Oracle applied to $A = \{a\}$. We now provide the two algorithms for Coverage announced in Section 3.

Semi-ladder Algorithm. The Semi-ladder Algorithm proceeds in a number of rounds, where each round consists of two steps: first the *Candidate Step*, and then the *Witness Step*. Also, the algorithm maintains a set B of witnesses gathered so far, initially set to be empty. The steps are defined as follows:

Candidate Step: Apply the Candidate Oracle to find a candidate $a \in L$ that agrees with all the witnesses in B. If no such candidate exists, terminate the algorithm returning that no solution exists. Otherwise, proceed to the Witness Step.

Witness Step: Apply the Weak Witness Oracle to find a witness $b \in R$ that does not agree with a. If there is no such witness, terminate the algorithm and return a as the solution. Otherwise, add b to B and proceed to the next round.

The correctness of the algorithm is obvious, while the running time can be bounded by the immediate observation that if the Semi-ladder Algorithm performs ℓ full rounds, then the candidates $a_1, \ldots, a_\ell \in L$ discovered in consecutive rounds, together with the witnesses $b_1, \ldots, b_\ell \in R$ added to B in consecutive rounds, form a semi-ladder in H.

▶ **Corollary 17.** *The Semi-ladder Algorithm applied to a graph H with semi-ladder index ℓ terminates after performing at most ℓ full rounds. Consequently, it uses at most $\ell + 1$ Candidate Oracle Calls, each involving a set of witnesses B with $|B| \leqslant \ell$, and at most ℓ Weak Witness Oracle Calls.*

Ladder algorithm. As before, the Ladder Algorithm maintains the set B of witnesses gathered so far, but also the set A of candidates found so far. The algorithm is also given a parameter $p \in \mathbb{N}$. Again, the algorithm proceeds in rounds, each consisting of the Candidate step and the Witness step, with the following description:

Candidate Step: Apply the Candidate Oracle to find a candidate $a \in L$ that agrees with all the witnesses in B. If no such candidate exists, terminate the algorithm returning that no solution exists. Otherwise, add a to A and proceed to the Witness step.

Witness Step: Apply the Strong Witness Oracle to set A and parameter p, yielding either a set of witnesses $P \subseteq R$ such that $|P| \leqslant p$ and every candidate from A does not agree with some witness from P, or a conclusion that no such set P exists. In the former case, add P to B and proceed to the next round. In the latter case, terminate the algorithm returning that a solution exists.

Note that the algorithm actually never finds a solution, but only may claim its existence in the Witness Step, and this claim is not substantiated by having a concrete solution in hand. However, the observation is that assuming the weak p-Helly property, the structure discovered by the algorithm is sufficient to deduce the existence of a solution.

▶ **Lemma 18 (♠).** *The Ladder Algorithm applied with parameter p in a bipartite graph with the weak p-Helly property is always correct.*

Finally, we show that if H has ladder index bounded by ℓ, then the algorithm terminates in a number of rounds bounded in terms of ℓ and p. For this we observe that during its execution, the algorithm in fact constructs a ladder in an auxiliary bipartite graph H' with candidates a on the left side and sets of witnesses P on the right side, and the ladder index of H' can be bounded in terms of p and the ladder index of H using a Ramsey argument.

▶ **Lemma 19 (♠).** *The Ladder Algorithm applied with parameter p to a bipartite graph H with ladder index smaller than ℓ terminates after performing less than $R^p(2\ell)$ full rounds.*

▶ **Corollary 20.** *The Ladder Algorithm applied with parameter p to a graph H with ladder index smaller than ℓ and the weak p-Helly property, always returns the correct answer and terminates after performing at most $q = R^p(\ell) - 1$ full rounds. Consequently, it uses at most $q+1$ Candidate Oracle Calls, each involving a set of witnesses B with $|B| \leqslant pq$, and at most q Strong Witness Oracle Calls, each involving a set of candidates A with $|A| \leqslant q$.*

Implementing the oracles. The last missing ingredient for obtaining our algorithmic results is an efficient implementation of the oracles for bipartite graphs of the form $\varphi(G)$, where G is the input graph and $\varphi(\bar{x}, \bar{y})$ is a formula expressing the considered problem. We describe such an implementation whenever φ is a distance formula.

We use the concept of *distance profiles* and *distance profile complexity*. Let G be a graph and let S be a set of its vertices. For a vertex v of G, the *distance-r profile* of v on S, denoted $\mathrm{profile}_r^{G,S}(v)$, is the function mapping S to $\{0, 1, \ldots, r, \infty\}$ such that for $s \in S$,

$$\mathrm{profile}_r^{G,S}(v)(s) = \begin{cases} \mathrm{dist}_G(v,s) & \text{if } \mathrm{dist}_G(v,s) \leqslant r, \\ \infty & \text{otherwise.} \end{cases}$$

The *distance-r profile complexity* of G is the function from $\mathbb{N}$ to $\mathbb{N}$ defined as

$$\nu_r^G(m) = \max_{S \subseteq V,\, |S| \leqslant m} |\{\mathrm{profile}_r^{G,S}(v) \colon v \in V(G)\}|.$$

That is, this is the maximum possible number of different functions from S to $\{0,1,\ldots,r,\infty\}$ realized as distance-r profiles on S of vertices of G, over all vertex subsets S of size at most m. For a graph class $\mathscr{C}$, we denote $\nu_r^{\mathscr{C}}(m) = \sup_{G\in\mathscr{C}} \nu_r^G(m)$.

Note that for any graph G and $r, m \in \mathbb{N}$ we have $\nu_r^G(m) \leqslant (r+2)^m$, as this is the total number of functions from a set of size m to $\{0,1,\ldots,r,\infty\}$. This bound is exponential in m, however it is known that on nowhere dense classes an almost linear bound holds.

▶ **Lemma 21** ([9]). *Let $\mathscr{C}$ be a nowhere dense class of graphs. Then for every $r \in \mathbb{N}$ and $\varepsilon > 0$ there exists a constant $c_{r,\varepsilon}$ such that $\nu_r^{\mathscr{C}}(m) \leqslant c_{r,\varepsilon} \cdot m^{1+\varepsilon}$ for all $m \in \mathbb{N}$.*

We remark that the conclusion of Lemma 21 still holds when $\mathscr{C}$ is any fixed power of a nowhere dense class, and when $\mathscr{C}$ is the class of map graphs. Moreover, when $\mathscr{C}$ is the class of $K_{t,t}$-free graphs for some $t \in \mathbb{N}$, then $\nu_1^{\mathscr{C}}(m) \leqslant \mathcal{O}(m^t)$.

We are ready to give implementations for the oracles. The main idea is that because we are working with a distance formula, when looking for, say, a candidate that agrees with all witnesses in a set B, the only information that is relevant about any vertex is its distance-r profile on the set S consisting of all vertices appearing in the tuples of B. Hence, there are only $\nu_r^G(|S|)$ different "types" of vertices, and instead of checking all k-tuples of vertices in the graph, we can check all k-tuples of types.

▶ **Lemma 22 (♠).** *Fix a distance formula $\varphi(\bar{x};\bar{y})$ of radius r and with $|\bar{x}| = c$ and $|\bar{y}| = d$. Then for an input graph $G = (V,E)$, there are implementations of oracle calls in $\varphi(G)$ that achieve the following running times:*

- *Candidate Oracle: time $\mathcal{O}(|B| \cdot \|G\| + |B| \cdot \nu_r^G(d|B|)^c)$ for a call to $B \subseteq V^{\bar{y}}$;*
- *Weak Witness Oracle: time $\mathcal{O}(\|G\| + \nu_r^G(c)^d)$ for a call to $\bar{a} \in V^{\bar{x}}$;*
- *Strong Witness Oracle: time $\mathcal{O}(|A|\cdot\|G\| + |A|\cdot\nu_r^G(c|A|)^{pd})$ for a call to $A \subseteq V^{\bar{x}}$ and $p \in \mathbb{N}$.*

Algorithmic consequences. We are ready to present our algorithmic corollaries, promised in Section 1. Throughout this section, when stating parameterized running times we use k to denote the target size of a solution (i.e., distance-r dominating or independent set). We start with the domination problems.

▶ **Theorem 23.** *Fix $r \in \mathbb{N}$ and let $\mathscr{C}$ be a class of graphs such that for each $q \leqslant r$, the class $\delta_q(\mathscr{C})$ has finite semi-ladder index. Then, for any positive distance formula $\varphi(\bar{x};\bar{y})$ of radius at most r and size k, the domination-type problem corresponding to φ can be solved on $\mathscr{C}$ in time $f(k)\cdot\|G\|$, for some function f.*

Proof. W.l.o.g. we can assume that $|\bar{x}|, |\bar{y}| \leqslant k$. Let $\ell \in \mathbb{N}$ be such that $\delta_q(\mathscr{C})$ has semi-ladder index smaller than ℓ, for all $q \leqslant r$. Given a graph G, we apply the Semi-Ladder Algorithm for the Coverage problem in the graph $\varphi(G)$ with implementations of oracles provided by Lemma 22. By Lemma 4 we conclude the semi-ladder index of $\varphi(\mathscr{C})$ is bounded by $R^k(\ell)$. Now the claimed running time follows immediately from Corollary 17 and Lemma 22. ◀

▶ Remark 24. By Corollary 17 and Lemma 22, the running time is actually $\mathcal{O}(p\cdot\nu_r^{\mathscr{C}}(p)^k\cdot\|G\|)$, where p is the semi-ladder index of $\varphi(G)$. By Lemma 4, we have that $p \leqslant R^k(\ell)$, which is upper-bounded by $k^{k\ell-1}$ for $k \geqslant 2$. Combining this with the trivial upper bound $\nu_r^{\mathscr{C}}(p) \leqslant (r+2)^p$ yields $f(k) \leqslant 2^{2^{\mathcal{O}(k\log k)}}$, where r and ℓ are considered fixed constants. However, if a priori we know for the graph class $\mathscr{C}$ that $\nu_r^{\mathscr{C}}(m)$ is polynomial in m, instead of exponential, then by the analysis above we obtain $f(k) \leqslant 2^{\mathcal{O}(k^2\log k)}$. Finally, by Lemma 5, for $\varphi = \delta_r^k$ – the formula corresponding to the Distance-r Dominating Set problem – we can use a sharper bound of $p \leqslant k^{\ell-1}$. Thus, for this case we obtain an upper bound of $f(k) \leqslant 2^{\mathrm{poly}(k)}$ in the general setting, and $f(k) \leqslant 2^{\mathcal{O}(k\log k)}$ when $\nu_r^{\mathscr{C}}(m)$ is polynomial in m.

Now, using Theorem 23 together with combinatorial results stated in Section 2 we immediately obtain the algorithmic results promised in Section 1. Note that the results hold not only for Distance-r Dominating Set, but even for every domination-type problem of fixed radius r and size k that is considered the parameter.

▶ **Theorem 25.** *Fix $r \in \mathbb{N}$. Then any domination-type problem defined by a positive distance formula of size k and radius at most r can be solved in time $2^{\mathcal{O}(k^2 \log k)} \cdot \|G\|$ on any graph class $\mathscr{C}$ such that either $\mathscr{C}$ is nowhere dense, or $\mathscr{C} = \mathscr{D}^s$ for a nowhere dense class $\mathscr{D}$ and some $s \in \mathbb{N}$, or $\mathscr{C}$ is the class of map graphs, or $r = 1$ and $\mathscr{C}$ is the class of $K_{t,t}$-free graphs for some fixed $t \in \mathbb{N}$. Moreover, if this domination-type problem is* Distance-r Dominating Set *for parameter k, then the running time can be improved to $2^{\mathcal{O}(k \log k)} \cdot \|G\|$.*

Proof. By Theorem 11, the class $\delta_r(\mathscr{C})$ has finite semi-ladder index. By Lemma 21 and its strengthenings (see the comment below Lemma 21) $\nu_r^{\mathscr{C}}(m)$ is bounded by a polynomial in m. Hence, we may apply Theorem 23; the claimed running times follow from the remark following it. ◀

We now move to the independence problems, for which we apply the Ladder algorithm.

▶ **Theorem 26.** *Let $r \in \mathbb{N}$ and let $\mathscr{C}$ be a class of graphs such that for any $k \in \mathbb{N}$, the class $\eta_r^k(\mathscr{C})$ has ladder index smaller than $\ell(k)$ and has the weak $p(k)$-Helly property, for some functions $\ell, p \colon \mathbb{N} \to \mathbb{N}$. Then the* Distance-$r$ Independent Set *problem on $\mathscr{C}$ can be solved in time $f(k) \cdot \|G\|$, for some function f.*

Proof. Given a graph G, we apply the Ladder Algorithm in the graph $\eta_r^k(\mathscr{C})$ with implementations of oracles provided by Lemma 22. The correctness of the algorithm and the running time bound follow directly from Corollary 20 and Lemma 22, where we may set $f(k) = \mathcal{O}(R^{p(k)}(2\ell(k)) \cdot \nu_r^{\mathscr{C}}(p(k) \cdot R^{p(k)}(2\ell(k))))^{p(k)\cdot k})$. ◀

▶ **Theorem 27.** *For any $r \in \mathbb{N}$ and nowhere dense class $\mathscr{C}$, the* Distance-r Independent Set *problem on $\mathscr{C}$ can be solved in time $f(k) \cdot \|G\|$, for some function f.*

Proof. By Theorems 6 and 16, for every $k \in \mathbb{N}$ there are constants $\ell, p \in \mathbb{N}$, depending on k, such that the class $\eta_r^k(\mathscr{C})$ has ladder index bounded by ℓ and has the weak p-Helly property. This allows us to apply Theorem 26. ◀

5 Discussion of related results

Fixed-parameter tractability of both Distance-r Dominating Set and Distance-r Independent Set on any nowhere dense class follows from the general model-checking result for first-order logic of Kreutzer et al. [11]. The algorithms derived in this manner have running time $f(k) \cdot n^{1+\varepsilon}$ for any fixed $\varepsilon > 0$ and some function f, where n is the number of vertices of the input graph. In fact, an algorithm with running time $f(k) \cdot n^{1+\varepsilon}$ for the Distance-r Independent Set problem is one of the intermediate results used in [11]. A close inspection of this algorithm reveals that the polynomial factor is in fact $\|G\|$, improving the claimed $n^{1+\varepsilon}$, however this is not explicit in [11]. For the Distance-r Dominating Set problem, its fixed-parameter tractability on any nowhere dense class was established earlier by Dawar and Kreutzer [4], but their algorithm had at least a quadratic polynomial factor in the running time bound.

As far as Distance-r Dominating Set on powers of nowhere dense classes is concerned, we remark that the result provided in Theorem 25 would *not* follow immediately from applying the algorithm on the graph before taking the power, for radius rs instead of r. The

reason is that the input consists only of the graph G^s, and it is completely unclear how to algorithmically find the preimage G if we are dealing with an arbitrary nowhere dense class $\mathscr{D}$. To the best of our knowledge, this result is a completely new contribution.

Regarding map graphs, the fixed-parameter tractability of the Distance-r Dominating Set problem on this class of graphs was established by Demaine et al. [6]. However, they use the recognition algorithm for map graphs of Thorup [23] to draw a map model of the graph; this algorithm has an estimated running time of at least $\mathcal{O}(n^{120})$ [2] and not all technical details have been published. Another way of obtaining a fixed-parameter algorithm would be to use the fact that map graphs have *locally bounded rankwidth*; however, again achieving linear running time would be difficult due to the need of computing branch decompositions with approximately optimum rankwidth, for which the best known algorithms have cubic running time. In contrast, as we have shown, the Semi-ladder Algorithm solves the problem in linear fixed-parameter time without the need of having a map model provided.

Finally, the fixed-parameter tractability of Dominating Set on $K_{t,t}$-free graphs, where both k and t are considered parameters, was established by Telle and Villanger [22]. Thus, Theorem 25 reproves this result and also improves upon the running time: from $2^{\mathcal{O}(k^{t+2})} \cdot \|G\|$ of [22] to $2^{\mathcal{O}(k \log k)} \cdot \|G\|$.

References

1 Hans Adler and Isolde Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics*, 36:322–330, 2014.

2 Zhi-Zhong Chen. Approximation Algorithms for Independent Sets in Map Graphs. *J. Algorithms*, 41(1):20–40, 2001.

3 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

4 Anuj Dawar and Stephan Kreutzer. Domination Problems in Nowhere-Dense Classes. In *FSTTCS 2009*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009.

5 Anuj Dawar and Stephan Kreutzer. Parameterized complexity of first-order logic. In *Electronic Colloquium on Computational Complexity, TR09-131*, page 39, 2009.

6 Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

7 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and Sparseness: the Case of Dominating Set. In *STACS 2016*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. Full version available as arxiv preprint `https://arxiv.org/abs/1411.4575`. `doi:10.4230/LIPIcs.STACS.2016.31`.

8 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36, 2013.

9 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood Complexity and Kernelization for Nowhere Dense Classes of Graphs. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. Full version available as arxiv preprint `https://arxiv.org/abs/1612.08197`. `doi:10.4230/LIPIcs.ICALP.2017.63`.

10 Grzegorz Fabiański, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Progressive Algorithms for Domination and Independence. *arXiv preprint*, 2018. `arXiv:1811.06799`.

11 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM*, 64(3):17:1–17:32, 2017.

12 Jaroslav Nešetřil and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(03):868–887, 2010.

13 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

14 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

15 Allen O'Hara. An Introduction to Equations and Equational Theories, 2011.

16 MichałPilipczuk and Sebastian Siebertz. Kernelization and approximation of distance-r independent sets on nowhere dense graphs. *arXiv preprint*, 2018. URL: `1809.05675`.

17 Michał Pilipczuk and Sebastian Siebertz. Lecture notes for the course "Sparsity" given at Faculty of Mathematics, Informatics, and Mechanics of the University of Warsaw, Winter Semester 2017/18. Available at `https://www.mimuw.edu.pl/~mp248287/sparsity`.

18 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. On the number of types in sparse graphs. In *LICS 2018*, pages 799–808. ACM, 2018.

19 Anand Pillay and Gabriel Srour. Closed Sets and Chain Conditions in Stable Theories. *J. Symb. Log.*, 49:1350–1362, 1984.

20 Klaus-Peter Podewski and Martin Ziegler. Stable graphs. *Fundamenta Mathematicae*, 100(2):101–107, 1978.

21 Saharon Shelah. *Classification theory: and the number of non-isomorphic models*, volume 92. Elsevier, 1990.

22 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *ESA 2012*, volume 7501 of *LNCS*, pages 802–812. Springer, 2012.

23 Mikkel Thorup. Map Graphs in Polynomial Time. In *FOCS 1998*, pages 396–405. IEEE Computer Society, 1998.

Modification to Planarity is Fixed Parameter Tractable

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
fomin@ii.uib.no

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
petr.golovach@uib.no

Dimitrios M. Thilikos
AlGCo project-team, LIRMM, Université de Montpellier, CNRS, Montpellier, France
sedthilk@thilikos.info

Abstract

A *replacement action* is a function $\mathcal{L}$ that maps each k-vertex labeled graph to another k-vertex graph. We consider a general family of graph modification problems, called $\mathcal{L}$-Replacement to $\mathcal{C}$, where the input is a graph G and the question is whether it is possible to replace in G some k-vertex subgraph H of it by $\mathcal{L}(H)$ so that the new graph belongs to the graph class $\mathcal{C}$. $\mathcal{L}$-Replacement to $\mathcal{C}$ can simulate several modification operations such as edge addition, edge removal, edge editing, and diverse completion and superposition operations. In this paper, we prove that for any action $\mathcal{L}$, if $\mathcal{C}$ is the class of planar graphs, there is an algorithm that solves $\mathcal{L}$-Replacement to $\mathcal{C}$ in $O(|G|^2)$ steps. We also present several applications of our approach to related problems.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Modification problems, Planar graphs, Irrelevant vertex technique

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.28

Funding *Fedor V. Fomin*: Supported by the Research Council of Norway via the projects "CLASSIS" and "MULTIVAL". Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.
Petr A. Golovach: Supported by the Research Council of Norway via the projects "CLASSIS" and "MULTIVAL". Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.
Dimitrios M. Thilikos: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010). Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.

1 Introduction

The *irrelevant vertex technique* was proposed by Robertson and Seymour for the Disjoint Paths problem [24, 25], which is the central algorithmic result of their Graph Minors series of papers. A superficial description of the technique is

> *If the treewidth of the input graph is small, then standard techniques on graphs of bounded treewidth can be used to solve the problem we have on hands. Otherwise, the graph contains an irrelevant vertex, that is, the vertex whose removal does not change the problem.*

Of course, the devil is in details, and usually in order to make the irrelevant vertex technique work, highly non-trivial arguments are involved in proving the existence of an irrelevant vertex, see e.g. [1, 7, 12, 13, 14, 15, 17, 18, 19, 20, 22, 23]. We also refer to [6, Chapter 7] for a high-level overview of the irrelevant vertex technique.

There are a number of generic algorithmic results in the literature explaining why and when a certain algorithmic technique is successful. For example, we know that many problems can be solved by dynamic programming on graphs of bounded treewidth, and Courcelle's theorem explains why this happens [5]. However, we do not know any generic characterization of problems solvable by the irrelevant vertex technique and the quest for such a characterization is the main motivation for this paper.

We show that the irrelevant vertex approach can be used to establish fixed-parameter tractability of a very general class of graph transformation problems. The problem we consider is the following: For some integer k, is it possible to transform an input graph G into a planar graph by performing at most k allowed changes? The allowed changes are defined through the set of the following *replacement actions*. Suppose that for every labelled k-vertex graph H we have a list $L(H)$ of labelled k-vertex graphs. Then the replacement action selects a subset of k vertices X in the graph G and replaces the subgraph $G[X]$ induced by X by a graph F from the list $L(G[X])$). More precisely, the action selects a k-sized vertex subset X of G labelled by numbers $\{1,\ldots,k\}$ and, given that H is the labelled k-vertex graph obtained from $G[X]$, we select a labelled k-vertex graph F from $L(H)$ and replace H by F. Thus the vertex set of the new graph G' is $V(G)$ and it has the same adjacencies as in G except pairs of vertices from X. In the transformed graph, vertices $u,v \in X$ labelled by $i,j \in \{1,\ldots,k\}$ are adjacent in G' if and only if $\{i,j\}$ is an edge of F. Then the task is for a given graph G and the family of allowed replacement actions, to decide whether there is a replacement action transforming G into a planar graph.

The problem of replacements to a planar graph encompasses many interesting graph modification problems. For example, the simplest replacement action is defined by associating with every k-vertex graph H the list $L(H)$ consisting of an edgeless k-vertex graph. This encodes the problem of finding in graph G a set of k vertices X such that deleting all edges with both endpoints in X results in a planar graph. By selecting an appropriate set of replacements, one can encode many interesting graph transformation problems, with specified properties of the replaced subgraph. This also includes various structural properties of replaced subgraph H, like being a a matching, a clique, or a cycle. Similarly replacement actions can describe the structural properties of the replacement graph F. For example, the condition could be that F is the complement of H. Or it could be some quantitive property, like if we delete $k/100$ edges, we have to add a least $k/200$ edges, or that one graph is obtained from another by flipping k edges according to some specified rules, etc.

Our main result is an algorithm that for any choice of the replacement actions on k-vertex graphs, decides whether an n-vertex input graph G can be made planar by making use of replacement in time $\mathcal{O}(f(k)\cdot n^2)$, where f is some function of k only. In other words, the problem is fixed-parameter tractable (FPT) parameterized by k.

While, from the general perspective, the proof of our main result follows the path of all irrelevant vertex techniques papers, there are several significant differences compared with the previous works. The main difficulty we have to resolve is the following. The most common argument towards application of the irrelevant vertex technique is that if we find a large "flat wall" (a grid-like part of the graph which has a planar embedding), then the central part of the wall is irrelevant. This does not work for replacements – the reason is that replacements are *non-local* and may affect vertices that are anywhere in the graph. Thus even if a vertex is inside a huge wall, it still can be used in the action. To overcome this issue we have to define an equivalence relation between pieces of the wall expressing the fact that equivalent pieces are "interchangeable" with respect to any application of an action. Next we have to detect a sufficiently large set of "equivalent" pieces of the wall and

prove that at least one of these pieces can be considered untouched by the transformation. Identifying equivalent parts of the wall is the main technical challenge. While parts of the wall are of bounded treewidth, typical applications of equivalence relations for graphs of bounded treewidth used in the literature strongly exploit the property that the boundary of treewidth bounded graphs is also bounded. However, this is not true in our case because the parts of the wall may have huge boundaries. This requires a new way to define equivalence relations and an efficient procedure for handling new relations, which in turn allows us to simplify the input of large treewidth. We believe that the technical contribution of our work will be the starting point for dealing with other algorithmic applications on planar graphs.

Section 3 is devoted to a high-level description of our algorithm, while the formal definitions for some basic concepts of this description are presented in Section 4. In Section 5 we provide several examples of problems that can be reduced to the replacement framework and in Section 6 we conclude with some open problems and further research directions.

Related work. Graph planarization and, more generally, graph modification, is one of the main themes in Parameterized Algorithms. For example, Planar Vertex Deletion, where one asked to remove at most k vertices of the input graph to make it planar. This problem is fixed-parameter tractable parameterized by k by the generic result from the Graph Minors project of Robertson and Seymour [24], who showed that every minor-closed property of graphs can be checked in polynomial time. The result of Robertson and Seymour yields the existence of an algorithm for this problem but provides no way to construct such an algorithm. Constructive algorithms for Planar Vertex Deletion were considered in [16, 23]. The fastest known algorithm for this problem runs in time $2^{\mathcal{O}(k \log k)}n$ [15]. The problem of obtaining a planar graph by contracting edges was considered in [13]. More generally, the problem of modifying a graph to some graph class excluding some fixed minors, was considered in [9, 8, 21, 10, 11].

2 Definition of the problem and outline of the algorithm

Elementary definitions. We use $\mathbb{N}$ to denote the set of all non-negative numbers. Given a $k_1, k_2 \in \mathbb{N}$, we denote by $[k_1, k_2]$ the set $\{k_1, \ldots, k_2\}$ and given k we denote $[k] = [1, k]$. Given a function $\varphi : A \to B$ and a subset $X \subseteq A$ we naturally extend φ by using $\varphi(X)$ to denote $\{\varphi(x) \mid x \in X\}$. We also write $\varphi|_X$ to denote the restriction of φ to $X \subseteq A$. We denote by $\mathbf{inj}(A, B)$ the set of all injections from A to B. For $\varphi \in \mathbf{inj}(A, B)$, we denote by φ^{-1} the mapping of $\varphi(B)$ to A that is the reverse of φ.

All graphs in this paper are undirected, finite, simple, and without multiple edges. Given a graph G we denote by $V(G)$ and $V(E)$ the set of its vertices and edges, respectively. We also denote $|G| = |V(G)|$. If $S \subseteq V(G)$, then we denote by $G \setminus S$ the graph obtained by G after removing from it all vertices in S, together with the incident edges. We define the subgraph of G *induced* by S as the graph $G[S] = G \setminus (V(G) \setminus S)$. Given an induced subgraph G' of G, we define $G \setminus G' = (V(G), E(G) \setminus E(G'))$, i.e., $G \setminus G'$ is obtained from G if we remove the edges of G'. Given two graphs G_1 and G_2, we define their *union* as $G_1 \cup G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$.

2.1 Replacement actions

Replacements. A *k-numbered-graph* is any graph H where $V(H) = [k]$, i.e., the vertices of H are the numbers $\{1, \ldots, k\}$. We denote the set of all k-numbered graphs by $\mathcal{H}_k$ and we set $\mathcal{H} = \bigcup_{k \in \mathbb{N}} \mathcal{H}_k$. A *replacement action* (abbreviated as *R-action*) is any function $\mathcal{L} : \mathcal{H} \to \mathcal{H}$, where for every $H \in \mathcal{H}$, $|\mathcal{L}(H)| = |H|$, i.e., graphs in $\mathcal{H}$ are mapped to same-size graphs.

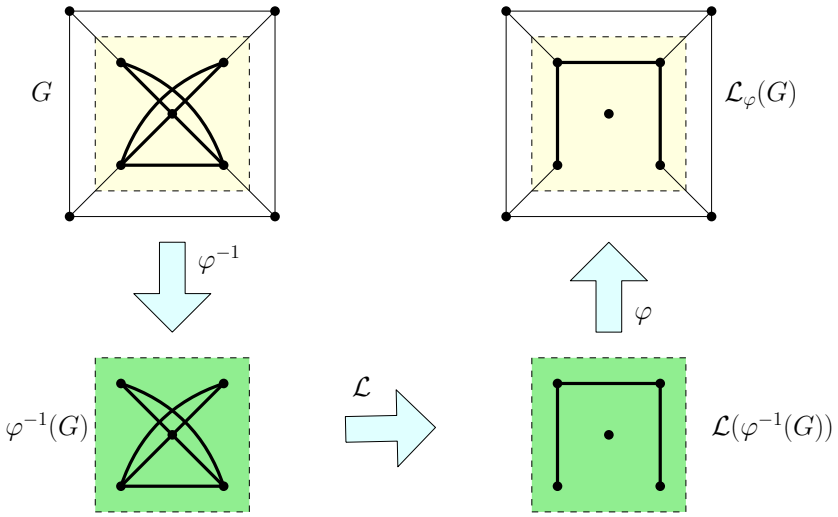


Figure 1 An illustration of a replacement action where the R-action $\mathcal{L}$ replaces graphs by their complements.

Let G be a graph and let $\varphi \in \mathbf{inj}([k], V(G))$. We define $\varphi^{-1}(G) = ([k], \{\varphi^{-1}(e) \mid e \in E(G[\varphi([k])])\})$, i.e., we see $\varphi^{-1}(G)$ is the graph in $\mathcal{H}_k$ that is isomorphic, via φ, to the subgraph of G where φ applies.

Let G be a graph and let G' be a graph where $V(G') \subseteq V(G)$. We denote $G \sqcup G' = (G \setminus G[V(G')]) \cup G'$, i.e., $G \sqcup G'$ occurs if we remove from G the edges between vertices in G' and then add all edges of G'. Given a graph G, a $\varphi \in \mathbf{inj}([k], V(G))$, and a $H \in \mathcal{H}_k$, we define $\varphi(H) = \{\varphi([k]), \{\varphi(e) \mid e \in E(H)\}\}$. Given an R-action $\mathcal{L} : \mathcal{H} \to \mathcal{H}$, we set $\mathcal{L}_\varphi(G) = G \sqcup \varphi(\mathcal{L}(\varphi^{-1}(G)))$, in other words, we consider the part of G that is delimited by φ and then we replace this part by its image via $\mathcal{L}$ (see Figure 1 for an example).

We now have all ingredients we need for defining our general problem.

Replacement to planarity. We examine the following family of problems, that may vary, depending on the choice of the R-action $\mathcal{L}$:

$\mathcal{L}$-Replacement to a Planar Graph. ($\mathcal{L}$-RP)
Input: A graph G and a non-negative integer k.
Question: Is there a $\varphi \in \mathbf{inj}([k], V(G))$ such that $\mathcal{L}_\varphi(G)$ is planar?

▶ **Theorem 1.** *For every R-action $\mathcal{L}$, there exists an algorithm that given an instance (G, k) of $\mathcal{L}$-Replacement to a Planar Graph, reports whether (G, k) is a yes-instance in $O_k(|G|^2)$ steps*[1].

The main result of the paper is a proof of Theorem 1. In fact, we give an algorithm that runs in the same running time, for the following more general annotated version of this problem:

[1] Given a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and a function $g : \mathbb{N} \to \mathbb{N}$, we use the notation $f(k, n) = O_k(g(n))$ in order to say that there is a function $h : \mathbb{N} \to \mathbb{N}$ where $f(k, n) = O(h(k) \cdot g(n))$.

$\mathcal{L}$-Annotated Replacement to a Planar Graph ($\mathcal{L}$-ARP)
Input: A graph G, a set of *annotated* vertices $R \subseteq V(G)$, and a non-negative integer k.
Question: Is there a $\varphi \in \mathbf{inj}([k], R)$ such that $\mathcal{L}_\varphi(G)$ is planar?

Theorem 1 can easily be generalized for the case when instead of a single R-action $\mathcal{L}$ we are given a set $\mathfrak{L}$ of R-actions and consider the following problem:

$\mathfrak{L}$-List Replacement to a Planar Graph ($\mathfrak{L}$-LRP)
Input: A graph G and a non-negative integer k.
Question: Is there an R-action $\mathcal{L} \in \mathfrak{L}$ and $\varphi \in \mathbf{inj}([k], V(G))$ such that $\mathcal{L}_\varphi(G)$ is planar?

Since $|\mathcal{H}_k| = 2^{\binom{k}{2}}$, by brute force checking all R-actions of $\mathfrak{L}$ restricted to $\mathcal{H}_k$, we reduce this more general problem to $\mathcal{L}$-Replacement to a Planar Graph and obtain the following corollary.

▶ **Corollary 2.** *For every family of R-actions $\mathfrak{L}$, there exists an algorithm that given an instance (G, k) of $\mathfrak{L}$-List Replacement to a Planar Graph, reports whether (G, k) is a yes-instance in $O_k(|G|^2)$ steps.*

Another possibility to generalize the problem is to allow multiple actions, that is, to consider the following variant for a given R-action $\mathcal{L}$:

$\mathcal{L}$-Consecutive Replacement to a Planar Graph ($\mathcal{L}$-CRP)
Input: A graph G, and two non-negative integers k and r.
Question: Is there a tuple $\varphi_1, \ldots, \varphi_h \in \mathbf{inj}([k], V(G))$ for some $h \leq r$ such that $\mathcal{L}_{\varphi_h}(\ldots \mathcal{L}_{\varphi_2}(\mathcal{L}_{\varphi_1}(G))\ldots)$ is planar?

Notice that $\mathcal{L}_{\varphi_h}(\ldots \mathcal{L}_{\varphi_2}(\mathcal{L}_{\varphi_1}(G))\ldots)$ can be seen as $\hat{\mathcal{L}}_\psi$ where $\psi \in \mathbf{inj}([k'], V(G))$ and $\hat{\mathcal{L}}$ is some R-action for $k' = |\varphi_1([k]) \cup \ldots \cup \varphi_h([k])|$. Since $k' \leq rk$, we can check all possible values of k' and for each k', construct the family of all possible R-actions $\mathfrak{L}$ that contain all feasible $\hat{\mathcal{L}}$ that can be results of compositions of $\mathcal{L}$. This way, we reduce $\mathcal{L}$-Consecutive Replacement to a Planar Graph to $\mathfrak{L}$-List Replacement to a Planar Graph and show that the problem is FPT when parameterized by $k + r$.

▶ **Corollary 3.** *For every R-action $\mathcal{L}$ and a positive integer r, there exists an algorithm that given an instance (G, k) of $\mathcal{L}$-Consecutive Replacement to a Planar Graph, reports whether (G, k) is a yes-instance in $O_{k+r}(|G|^2)$ steps.*

In fact, it is possible to combine both generalizations and allow multiple actions chosen from a given list or even distinct lists.

3 High-level description of the algorithm

As our algorithm for $\mathcal{L}$-ARP is quite involved, in this extended abstract we present an outline of its main ideas. Our description is high-level. Formal definitions of the most important concepts in this description can be found in Section 4.

Basic concepts. We start with some conventions. In the course of our description, the word "small-enough" (resp. "big-enough" or "many-enough") means upper (resp. lower) bounded by some function of k. By the term "flat part of G" we refer to some subgraph of G that can be embedded in a disk. Also by the term "bidimensionally big-enough part of a graph G" we refer to a flat part of G that contains a subdivision of a big-enough wall as a subgraph (see Figure 2). We say that a vertex of G is "well-enough insulated" if it is surrounded by a collection of many-enough homocentric cycles of some flat part of G. Intuitively, the existence of a bidimensionally big-enough part implies that a big-enough number of vertices of G are well-enough insulated.

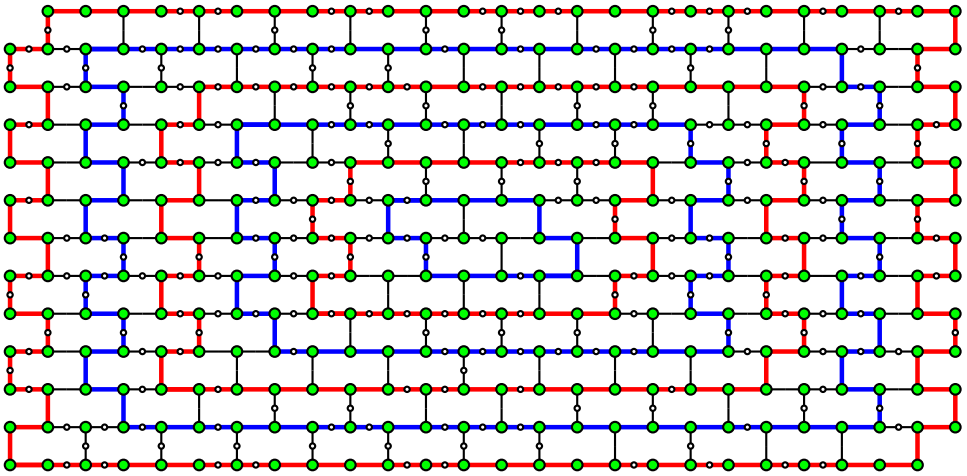


Figure 2 A subdivision of a 12×12-wall and the 6 homocentric cycles insulating its two central vertices.

Some preliminary observations. Before we present the main idea of the algorithm we start with some preliminary observations.

Observation 1: The kick-off observation is that it is possible to express $\mathcal{L}$-ARP in Monadic Second Order logic (MSOL). This implies that when the instance graph G has small-enough treewidth, then the problem can be solved in a linear in $n = |V(G)|$ number of steps. This observation is not straightforward as actions may also add edges in G and it requires some extra effort to translate this into MSOL.

Observation 2: The second observation is that if (G, k) is a yes-instance, then either G has bounded treewidth – and then we are done, because of Observation 1 – or it contains a bidimensionally big-enough flat part K. Moreover, using a result from [12, Subsection 4.1], we can also assume that K, besides the fact that it is bidimensionally big-enough, it has small-enough treewidth. This last property will permit us to apply MSOL-queries to any portion of K. It follows that K, if it exists, can be detected in $O(n)$ steps.

Observation 3: A third observation is that if some non-annotated vertex v in K is "surrounded" by a subdivided 3-wall-annulus A (see Figure 3) and the closed interior I_A of the outer cycle of A contains only non-annotated vertices, then we can remove v from G and reduce the instance to a simpler equivalent one. This reduction is based on the fact that the 3-connectivity of A offers enough rigidity for the graph inside I_A to remain unaffected in any planar graph that may be created by an action on G. Based on the above, we may assume that any bidimensionally big-enough territory of the flat part contains some annotated vertex in its interior.

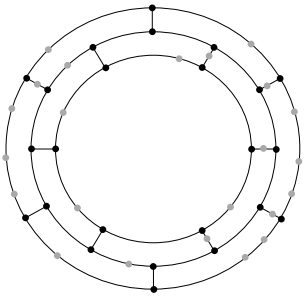


Figure 3 A subdivided 3-wall-annulus. The grey vertices are the subdivision vertices.

The territory-equivalence idea. The main idea of the algorithm is to detect, inside a bidimensionally big-enough part of G, a collection of many-enough pairwise-disjoint *territories* and to define a suitable notion of equivalence between them that expresses all the ways an action may affect them.

A critical aspect of our approach is that the number of equivalence classes of such an equivalence relation should depend *only* on k. This permits us, given that we have many-enough territories, to algorithmically detect some sub-collection of $k+1$ of them that are indistinguishable with respect to any action that can be applied on G. As an action affects vertices of at most k territories, we can arbitrary pick some particular annotated vertex inside one of them and create the equivalent instance where this vertex is no longer annotated. In this way, we can recurse to an equivalent instance of the problem that is more simple than the original one.

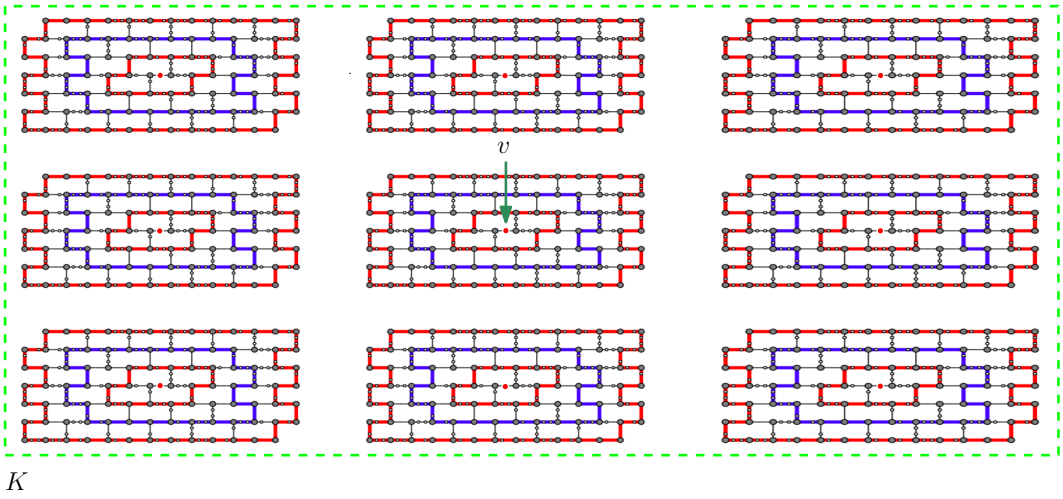


Figure 4 A visualization of the interior of the flat part K. Each subdivided sub-wall contains an annotated vertex v surrounded by many-enough homocentric cycles. In this particular figure, each subdivided sub-wall gives rise to a separation sequence of only 3 layers.

Separation sequences. We now explain how the above idea is implemented. First of all, we work on the flat part K (that can be detected due to the Observation 2) and use the fact that it is bidimensionally big-enough in order to detect in it a big-enough set X of well-enough insulated annotated "central" vertices. Thus each $v \in X$ is accompanied with a big-enough collection $\mathcal{C}_v = \{C_1, \ldots, C_r\}$ of surrounding cycles, called *a separation sequence* (assuming

that C_1 is the outermost cycle). Moreover, based on the big-enough bidimensionality of K we can also assume that for every two distinct $v, v' \in X$, the closed interiors of the outermost cycles of $\mathcal{C}_v$ and $\mathcal{C}_{v'}$ are disjoint (see Figure 4). Based on Observations 2 and 3, it is possible to detect such a set X of annotated vertices along with their accompanying cycle collections, in $O(n)$ steps. Moreover, for each cycle C_j in $\mathcal{C}_v$ we denote by G_j the graph cropped by its interior and we make sure that the 3 outermost cycles are parts of a subdivided 3-wall-annulus A_j. We call G_j *j-th prefix graph of v*.

We see each prefix graph as a doubly annotated graph $\mathbf{G}_j = (G_j, Z_j, R_j)$ where Z_j are the vertices inside the annulus A_j and R_j are the annotated vertices inside G_j. As r is big-enough, we know that for any application of an action on a set S of k vertices, that is able to transform G to a planar graph, there will be some $Z_j, j \in \{1, \ldots, r\}$ that will be disjoint from S. This permits us to see this Z_j as a *separator* of G whose inner part (including Z_j) is G_j. Moreover after an action that does not affect Z_j, the set Z_j will maintain its status as a separator in the resulting graph. This notion of separator is formalized by the concept of an *annulus-embedded separator* that is formally described in Subsection 4.2.

An equivalence relation on prefix graphs. Recall that a prefix graph G_j is a planar graph of bounded treewidth. We stress that typical equivalence relations that are based on MSOL-expressible properties require that the boundary of the bounded-treewidth graphs is also bounded. However, this is not our case. Instead we will use the fact that the boundary Z_j "well insulates" G_j from the rest of the graph.

We need to encode, for each $\mathbf{G}_j = (G_j, Z_j, R_j)$, all possible ways an action may rearrange G in both sides of the separator Z_j, assuming that it does not affect the vertices of Z_j. Clearly, if S is the set of vertices of G on which an action is applied, then some part of S is outside G_j and another part is inside G_j (but not in Z_j). As the action does not affect the separator Z_j, it will create a planar graph G' where the outer and inner part will still be well-defined with respect to Z_j.

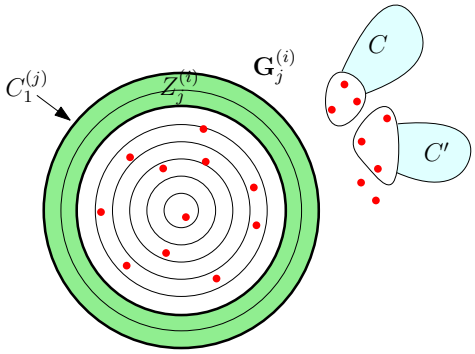


Figure 5 A prefix graph G_i. The set Z_j are the vertices in the annulus defined by its 3 first layers. The red vertices are the vertices affected by some action. The graphs C' and C are connected components of $(G \setminus V(G_j)) \setminus S$ that will be relocated after the application of an action to G.

The vertices of S along with the connected components of $(G \setminus V(G_j)) \setminus S$ will be relocated in G' in the inner and the outer parts of Z_j. However, these possible relocations depend only one the size of S (that is k) and the ways these components can be seen as "partially-planar graphs" with respect to their boundary in S (which also depends on k). The precise encoding of this is expressed by the notion of a *replacement folio of a doubly annotated graph*, formally defined in Subsection 4.3. This encoding is justified by a Lemma asserting (the proof is

omitted in this extended abstract) that any action on G that avoids Z_j can be "represented" by this folio in the sense that if two doubly annotated graphs, that correspond to different separation sequences $\mathcal{C}_v$ and $\mathcal{C}_{v'}$, have the same replacement folio, then the application of an action to the first can be simulated by the application of an action to the second.

An important technical step is to prove a series of lemmata that "translate" all possible elements of a replacement folio to MSOL-formulas. Again we stress that this is not straightforward: besides the need of suitably encoding partially-planar graphs, we also have to express all the ways "missing edges" of G may appear as the result of an action. As a consequence of its MSOL-expressibility, it follows that our equivalence relation on territories has a small-enough number of equivalence relations. This, together with the bounded treewidth of the flat part K, permits us to detect in it $k+1$ equivalent territories in $O_k(n)$ steps, "de-annotate" one of their annotated "central vertices" and create a simpler equivalence instance. Given this reduction and the reduction of Observation 3, we may use $|V(G)|+|R| \leq 2|V(G)|$ as the complexity-measure of an instance of our algorithm. Therefore, after $O(n)$ recursive calls of the above reductions, the algorithm provides a correct answer or an equivalent instance whose graph has small-enough treewidth. In the latter case, the correct answer can be computed in $O_k(n)$ steps, as mentioned in Observation 1. Based on the above, the overall running time of the algorithm is $O_k(n^2)$.

4 Key concepts

4.1 Graph embeddings and boundaried graphs

Embedded graphs. We denote by $\mathbb{S}_0$ the sphere $\{(x,y,z) \mid x^2+y^2+z^2=1\}$ and we refer to it as *the plane*. We consider embeddings or partial embeddings of graphs on $\mathbb{S}_0$ and several subsets of it. Such subsets can be closed *disks*, i.e. subsets of $\mathbb{S}_0$ that are homeomorphic to the set $\{(x,y,z) \mid x^2+y^2 \leq 1\}$ and *closed annuli*, i.e., subsets of $\mathbb{S}_0$ that are homeomorphic to the set $\{(x,y,z) \mid \frac{1}{2} \leq x^2+y^2 \leq 1\}$. Given a set X that is either a closed disk or an annulus, we denote its boundary (i.e., the set of points of X for which every neighborhood around them contains some point not in Z) by $\mathbf{bor}(X)$. Notice that if A is an annulus, then the set $\mathbf{bor}(A)$ has two connected components that are both cycles. We call these cycles *boundaries* of A. An *oriented annulus* is a triple $\mathbb{A} = (A, N_{\text{in}}, N_{\text{out}})$ where A is an annulus and N_{in} and N_{out} are its boundaries. We say that N_{in} (resp. N_{out}) is the *inner* (resp. *outer*) boundary of $\mathbb{A}$. Given an oriented annulus $\mathbb{A} = (A, N_{\text{in}}, N_{\text{out}})$ we define $\mathbf{rev}(\mathbb{A}) = (A, N_{\text{out}}, N_{\text{in}})$, i.e., we exchange the roles of the inner and the outer boundary. When we embed a graph G in the plane, in the annulus, or in a disk, we treat G as a set of points. This permits us to make set operations operations between graphs and sets of points. For instance, if G is a graph embedded in the plane and Δ is a closed disk in the plane, we can use the notation $V(G) \cap D$ in order to the set of vertices of G that are points of D. Also, given that $\mathbf{bor}(D) \cap G \subseteq V(G)$, we use $G \cap D$ to denote the graph formed by the vertices and the edges of G that are inside D. We denote by $\mathcal{P}$ the class of all planar graphs.

Annotated graphs. An *annotated graph* is a pair (G,R) where G is a graph and $R \subseteq V(G)$. A triple (G,R,Z) where $R, Z \subseteq V(G)$ is called *doubly annotated graph*.

Boundaried graphs. Let $k \in \mathbb{N}$. A *k-boundaried graph* is a triple $\mathbf{G} = (G, B, \lambda)$ where (G,B) is an annotated graph and $\lambda \in \mathbf{inj}(B,[k])$ (keep in mind that $|B| \leq k$). For every $x \in R$, we refer to the number $\lambda(x)$ as the *index* of x in $\mathbf{G}$ and we define the *index set* of $\mathbf{G}$ as $\Lambda(\mathbf{G}) = \lambda(B)$. We call B the *boundary* of $\mathbf{G}$ and the vertices of B *the boundary vertices*

of $\mathbf{G}$. We also denoted $B(\mathbf{G}) = B$ and $\lambda(\mathbf{B}) = \lambda$. Also we define the *size*, denoted by $|\mathbf{G}|$ of $\mathbf{G} = (G, B, \lambda)$ by $|G|$. We denote by $\mathcal{B}_k$ the set of all k-boundaried graphs, for each $k \in \mathbb{N}$. Given a $\mathbf{G} = (G, B, \lambda) \in \mathcal{B}_k$ we define the graph $\mathsf{gr}(\mathbf{G}) = (\Lambda(\mathbf{G}), \{\lambda(e) \mid e \in E(G[B])\})$, i.e., $\mathsf{gr}(G)$ is the graph in $\mathcal{H}_k$ that is obtained after taking the subgraph of G induced by the boundary $B(\mathbf{G})$ and then mapping it, via λ, to numbers in $[k]$. Also, given two $i, j \in \Lambda(G)$, we say that $i \sim_{\mathbf{G}} j$ if $\lambda^{-1}(i)$ and $\lambda^{-1}(j)$ belong in the same connected component of G. Clearly, $\sim_{\mathbf{G}}$ is an equivalence relation. Let $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{B}_k$. We say that $\mathbf{G}_1$ and $\mathbf{G}_2$ are *compatible* if $\mathsf{gr}(\mathbf{G}_1) = \mathsf{gr}(\mathbf{G}_2)$ and $\sim_{\mathbf{G}_1} = \sim_{\mathbf{G}_2}$.

Let $\mathbf{G}_1 = (G_1, B_1, \lambda_1)$ and $\mathbf{G}_2 = (G_2, B_2, \lambda_2)$ be two compatible k-boundaried graphs. We define the *gluing operation* $\oplus$ such that $\mathbf{G}_1 \oplus \mathbf{G}_2$ is the graph obtained by taking the disjoint union of G_1 and G_2 and then identifying each vertex in B_1 with the same-indexed vertex in B_2. We make the convention that after identifying a vertex $x \in B_1$ with a vertex $y \in B_2$, then the result of this identification in $\mathbf{G}_1 \oplus \mathbf{G}_2$ is again the vertex x (i.e., B_1 prevails over B_2). Finally, we say that $\mathbf{G}_1 \equiv \mathbf{G}_2$, if they are compatible and for every boundary graph $\mathbf{F}$ where $\mathsf{gr}(\mathbf{F}) = \mathsf{gr}(\mathbf{G}_i), i \in [2]$ it holds that

$$\mathbf{F} \oplus \mathbf{G}_1 \in \mathcal{P} \iff \mathbf{F} \oplus \mathbf{G}_2 \in \mathcal{P}. \tag{1}$$

4.2 Annulus-embedded separators

The notion of a subdivided 3-wall-annulus is depicted in Figure 3. Notice that each such graph contains two "boundary" cycles that we call *extremal cycles*.

Given a $S \subseteq V(G)$, we define as $\mathbf{ccin}(G, S)$ as the set of all connected components of $G \setminus S$ that are not connected components of G. We also define $\mathbf{ccout}(G, S)$ as the union of all connected components of $G \setminus S$ that are connected components of G.

Annulus-boundaried graphs. An *annulus-boundaried graph* is a quadruple $(C, K, Y, \mathbb{A})$ where

- C is a graph,
- K is a connected subgraph of C,
- Y is a subdivided 3-wall-annulus that is a subgraph of K,
- $\mathbb{A} = (A, N_{\text{in}}, N_{\text{out}})$ is an oriented annulus,
- Y is embedded in A so that
  - N_{in} and N_{out} are the two extremal cycles of Y, and
  - $G \cap A = K$.

We call the cycle of Y that is identical to N_{in} (resp. N_{out}) *inner* (resp. *outer*) cycle of $(C, K, Y, \mathbb{A})$. We say that an annulus-boundaried graph $(C, K, Y, \mathbb{A})$ is *planar* if Q can be embedded in a disk Δ such that $\mathbb{A} \subseteq \Delta$ and $N_{\text{out}} = \mathbf{bor}(\Delta)$.

Annulus-embedded separators. Let G be a graph. Let also $(K, Y, \mathbb{A})$ be a triple where K is a graph, Y is a subgraph of K and $\mathbb{A}$ is an oriented annulus. We say that $(K, Y, \mathbb{A})$, is a *annulus-embedded separator* of G if there are two subgraphs C_{in} and C_{out} of G such that both $(C_{\text{in}}, K, Y, \mathbb{A})$ and $(C_{\text{out}}, K, Y, \mathbf{rev}(\mathbb{A}))$ are annulus-boundaried graphs. Notice that each connected component of $\mathbf{ccin}(G, V(K))$ contains some vertex x where $N_G(x)$ intersects either the inner or the outer cycle of $(G, K, Y, \mathbb{A})$ (but not both). We also make the convention that all connected components of $\mathbf{ccout}(G, V(K))$ are subgraphs of C_{out} and we denote C_{in} (resp. C_{out}) *inner* (resp. *outer) component* of $(K, Y, \mathbb{A})$ in G. We say that $(K, Y, \mathbb{A})$ is *inner-planar* if $(C_{\text{in}}, K, Y, \mathbb{A})$ is planar.

The following observation easily directly from the definitions and the fact that that every 3-sd-annulus is 3-connected and has a unique embedding in the plane.

▶ **Observation 4.** *Let (G, R) be an annotated graph and let $(K, Y, \mathbb{A})$ be a annulus-embedded separator of G. Let also C_{in} and C_{out} be the inner and the outer component of G respectively. Then G is planar if and only if both C_{in} and C_{out} are planar.*

4.3 Replacement folios

A *k-planarity-folio* is a set $\mathcal{M}_k \subseteq \mathcal{B}_k$ such that for every $\mathbf{G} \in \mathcal{B}_k$ there is a $\mathbf{J} \in \mathcal{M}_k$ such that $\mathbf{J} \equiv \mathbf{G}$. It is known (see e.g., in [2]) that, for every $k \in \mathbb{N}$, it is possible to construct a k-planarity-folio $\mathcal{M}_k$ whose size depends on k. Given a $k \in \mathbb{N}$, we define $g_k = \max\{|\mathbf{C}| \mid \mathbf{C} \in \mathcal{M}_k\}$. For every graph L where $V(L) \subseteq [k]$, we define $\mathcal{C}_L = \{\mathbf{G} \in \mathcal{M}_k \mid \mathsf{gr}(\mathbf{G}) = L\}$. The following is an direct consequence of the definitions (see e.g., [2]).

▶ **Observation 5.** *For every k-numbered graph L and every two compatible boundaried graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ where $\mathsf{gr}(\mathbf{G}_1) = L$, it holds that $\mathbf{G}_1 \equiv \mathbf{G}_2$ if and only if $\forall \mathbf{F} \in \mathcal{C}_L$, $\mathbf{F} \oplus \mathbf{G}_1 \in \mathcal{P} \iff \mathbf{F} \oplus \mathbf{G}_2 \in \mathcal{P}$.*

From now on we fix some k-planarity folio $\mathcal{M}_k$. Given the above observation, the members of the $\mathcal{M}_k$ represent all ways a boundary graph can be "partially planar" with respect to its boundary.

Replacement folios. Our purpose now is to define a structure representing the effect of all replacement actions on a doubly annotated graph (G, R, Z). The actions we want to encode involve vertices in R that are not in Z. Also they involve vertices of some virtual graph D that is not a part of G. Later, the set Z will be the vertex set of a wall embedded separator of a graph and D will represent the part of the graph, affected by an action, that is in the outer component of this embedded separator.

Let $k \in \mathbb{N}$. A *k-pattern* is a quadruple $(\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}}, D, \tau)$ where

- $\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}} \in \mathcal{M}_k$ (both $\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}}$ are members of the k-planarity folio),
- $\Lambda(\hat{\mathbf{J}}_{\text{in}}) \cap \Lambda(\hat{\mathbf{J}}_{\text{out}}) = \emptyset$ ($\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}}$ do not have same-index boundary vertices),
- D is a graph where $|D| = |\mathsf{gr}(\hat{\mathbf{J}}_{\text{in}}) \cup \mathsf{gr}(\hat{\mathbf{J}}_{\text{out}})|$ (the size of D is the sum of the boundaried sizes of $\hat{\mathbf{J}}_{\text{in}}$, and $\hat{\mathbf{J}}_{\text{out}}$), and
- τ is an bijection from $V(D)$ to $\Lambda(\mathbf{J}_{\text{in}}) \cup \Lambda(\mathbf{J}_{\text{out}}))$ (i.e., τ is a labelling of D with numbers from the index sets of $\mathbf{J}_1$ and $\mathbf{J}_2$).

We denote by $\mathcal{P}_k$ the set of all k-patterns. Suppose now that (G, R, Z) is a doubly annotated graph. The *k-planar $\mathcal{L}$-replacement folio* of (G, R, Z) is the set $\mathfrak{L}_k(G, R, Z) \subseteq \mathcal{P}_k$ containing every quadruple $(\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}}, D, \tau) \in \mathcal{P}_k$ for which there exists an $\varphi \in \mathbf{inj}([k], V(D) \cup (R \setminus Z))$ such that

1. $\tau^{-1} \subseteq \varphi$,
2. if $G' = \mathcal{L}_\varphi(G \cup D)$, then $\tau(G') = \mathsf{gr}(\hat{\mathbf{J}}_{\text{in}}) \cup \mathsf{gr}(\hat{\mathbf{J}}_{\text{out}})$

and if

- $S_{\text{in}} = \varphi(\Lambda(\hat{\mathbf{J}}_{\text{in}}))$,
- $S_{\text{out}} = \varphi(\Lambda(\hat{\mathbf{J}}_{\text{out}}))$,
- $\hat{G} = (G', S_{\text{in}}, \varphi^{-1}|_{S_{\text{in}}}) \oplus \hat{\mathbf{J}}_{\text{in}}$,
- $\hat{U}_{\text{in}} = \mathbf{ccin}(\hat{G}, Z)$, and
- $\hat{U}_{\text{out}} = \mathbf{ccout}(\hat{G}, Z)$,

then

3. $\hat{U}_{\text{in}}$ is planar,
4. $S_{\text{out}} \subseteq V(\hat{U}_{\text{out}})$, and
5. $(\hat{U}_{\text{out}}, S_{\text{out}}, \varphi^{-1}|_{S_{\text{out}}}) \equiv \hat{\mathbf{J}}_{\text{out}}$.

Some intuition. Let us give some intuition on the above, quite technical, definition of the set $\mathfrak{L}_k(G, R, Z)$. Suppose that $(\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}}, D, \tau) \in \mathcal{P}_k$. The graph G should be seen as an annulus-boundaried graph $(G, K, Y, \mathbb{A})$ where $Z = V(K)$ where $(K, Y, \mathbb{A})$ is an annulus-embedded separator between G and a virtual "outside part". Assume also that φ is an action that affects vertices in $R \setminus Z$ (i.e., vertices in the interior, call it I_G, of the wall separator, but not in R) and some virtual vertices outside G. The graph D represents these vertices and the way these are connected between them (see Figure 6).

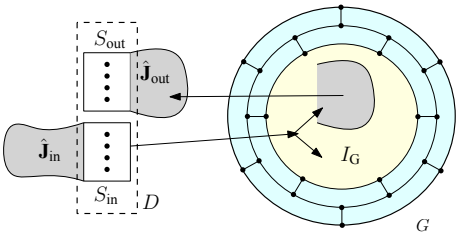


Figure 6 A 3-sd-annulus. The grey vertices are the subdivision vertices.

Notice that the action φ creates some edges between $V(D)$ and I_G. Moreover, the same action is rearranging the edges in D and the edges between the vertices in $\varphi([k]) \cap I_G$. The edges of D are rearranged so as to transform it to a graph isomorphic to $\mathsf{gr}(\hat{\mathbf{J}}_{\text{in}}) \cup \mathsf{gr}(\hat{\mathbf{J}}_{\text{out}})$ and this is enforced by conditions 1 and 2. Let G' be the result of the application of φ on $G \cup D$ and the $\hat{G}$ be result of gluing G' with $\hat{\mathbf{J}}_{\text{in}}$. Here $\mathbf{J}_{\text{in}}$ represents a virtual "outside part" that is not present in G. Notice that the connected components of this outside part are finally scattered either inside or outside the resulting graph. Moreover both parts should be finally "boundaried parts" of a planar graph and this is the reason we incorporated both planarity and connectivity in the definition of the k-planarity folio. Now $\hat{U}_{\text{in}}$ is the part of $\hat{G}$ that goes "inside" and $\hat{U}_{\text{out}}$ is the part of $\hat{G}$ that goes "outside" after the rearrangement. In condition 3 we demand the inside part $\hat{U}_{\text{in}}$ to be planar (we stress that this "inside part" may contain connected components of $\hat{\mathbf{J}}_{\text{in}}$). We also demand that outside part $\hat{U}_{\text{out}}$ contains S_{out} (condition 4) and that $\hat{U}_{\text{out}}$ is boundaried by S_{out} is equivalent to $\hat{\mathbf{J}}_{\text{out}}$ because of Condition 5 (again it is possible that this boundaried graph contains connected components of $\hat{\mathbf{J}}_{\text{in}}$). In this way, $\hat{\mathbf{J}}_{\text{out}}$ represents the virtual "outside part" the separator. Resuming, if $(\hat{\mathbf{J}}_{\text{in}}, \hat{\mathbf{J}}_{\text{out}}, D, \tau) \in \mathcal{P}_k$ then there is an action φ that outside G behaves as indicated by D and τ, that assumes that the virtual outside part is equivalent to $\hat{\mathbf{J}}_{\in}$ and, under this assumption, demands that the occurring outside part is equivalent to $\hat{\mathbf{J}}_{\text{out}}$.

As already mentioned in the high-level description of our algorithm, all conditions of the above definition can be expressed in Monadic Second Order logic. We omit the proof of this fact in this extended abstract.

5 Applications

5.1 Problems generated by different Instantiations of $\mathcal{L}$

$\mathcal{L}$-ARP can express several modification operations based on different instantiations of $\mathcal{L}$. In this section we give a series of examples. We slightly extend the definition of an action by demanding that $\mathcal{L} : \mathcal{H} \to \mathcal{H}$ is a *partial* function, i.e., we allow that $\mathcal{L}(H')$ may be undefined

for some $H' \in \mathcal{H}$ and, in such a case, we write $\mathcal{L}(H') = \mathsf{void}$. Moreover, in the question of $\mathcal{L}$-ARP we additionally demand that $\mathcal{L}(\varphi^{-1}(G)) \neq \mathsf{void}$. To reduce the enhanced version to the old one, we define G' as the disjoint union of G and K_5, we set $k' = k + 5$ and set up a $\mathcal{L}'$ so that $\mathcal{L}'(H)$ is defined as follows: if H' is the disjoint union of a graph H and a K_5 and $\mathcal{L}(H) \neq \mathsf{void}$, then $\mathcal{L}'(H')$ is equal to the disjoint union of $\mathcal{L}(H)$ and K_5^- (that is the graph K_5 without one edge, i.e., a planar graph), otherwise $\mathcal{L}(H) = H$. We now observe that (G, k) is a yes-instance of the enhanced $\mathcal{L}$-ARP iff (G', k') is a yes-instance of $\mathcal{L}'$-ARP.

We now proceed with a series of problems that can be easily expressed by the enhanced version of $\mathcal{L}$-ARP.

Planar Completion to a Subgraph. This problem has as input two planar graphs G and H and asks whether it is possible to add edges in G so that the resulting graph remains planar (a *planar completion* of G) and contains H as a subgraph. A variant of this problem, where G is given along with some plane embedding, has been examined in [4]. We set $k = |H|$. By setting $\mathcal{L} : \mathcal{H} \to \mathcal{H}$ where $\mathcal{L}(H') = H$, i.e., $\mathcal{L}$ is the constant function where the output is always H, we reduce Edge Completion to Subgraph to $\mathcal{L}$-ARP.

Planar Completion to an Induced Subgraph. Here, given G and H, we ask for a planar completion of G that contains H as an induced subgraph. To reduce this problem to $\mathcal{L}$-ARP, we consider $\mathcal{L} : \mathcal{H} \to \mathcal{H}$ where $\mathcal{L}(H') = H$ if H' is an induced subgraph of H, otherwise $\mathcal{L}(H') = \mathsf{void}$.

Edge Deletion to a Planar Graph. Edge Deletion to a Planar Graph asks whether we can remove at most k edges from G such that the resulting graph will be planar. A pair $(H, F) \in \mathcal{H}_{2k} \times \mathcal{H}_{2k}$ is *good* if $E(F) \subseteq E(H)$ and $|E(H) \setminus E(F)| \leq k$. For every good pair (H, F), we define the action $\mathcal{L}_{H,F} : \mathcal{H} \to \mathcal{H}$ where $\mathcal{L}_{H,F}(H') = F$ if $H' = H$, otherwise we set $\mathcal{L}_{H,F}(H') = \mathsf{void}$. Notice that (G, k) is a yes instance of Edge Deletion to a Planar Graph iff there is a good pair (H, F) where (G, k) is a yes-instance of $\mathcal{L}_{H,F}$-ARP. As there are $O_k(1)$ good pairs, this implies an FPT-algorithm for Edge Deletion to a Planar Graph when parameterized by k.

Alternatively, we can see this problem as a special version of $\mathcal{L}$-CRP by exchanging the roles of r and k, setting $k = 2$, $\mathcal{L}(K_2) = \overline{K}_2$, and $\mathcal{L}(\overline{K}_2) = \overline{K}_2$, where $\overline{K}_2$ is the complement of the complete graph on two vertices.

Matching Deletion to a Planar Graph. We ask here whether we can remove a matching of size at most k in order to create a planar graph. The reduction is again the same as in the case of Edge Deletion to a Planar Graph, however now, for a pair $(H, F) \in \mathcal{H}_{2k} \times \mathcal{H}_{2k}$ to be good, we additionally ask that $E(H) \setminus E(G)$ is a matching of H.

Planar Subgraph Isomorphism. Planar Subgraph Isomorphism has as input two planar graphs G and J and asks whether G contains J as a subgraph. We define the action $\mathcal{L} : \mathcal{H} \to \mathcal{H}$ where $\mathcal{L}(H') = H'$ if J is a subgraph of H', otherwise $\mathcal{L}(H') = \mathsf{void}$. Then $\mathcal{L}$-ARP is the Planar Subgraph Isomorphism.

Planar Induced Subgraph Isomorphism. Planar Induced Subgraph Isomorphism has as input two planar graphs G and J and asks whether G contains J as an induced subgraph. The construction of $\mathcal{L}$ is as in the previous case with the difference that we now demand that J is isomorphic to H'.

Edge-disjoint Planar Superposition: given two planar graphs G and H, check whether H is a subgraph of the complement of G and $H \cup G$ is planar. We define the action $\mathcal{L} : \mathcal{H} \to \mathcal{H}$ where $\mathcal{L}(H') = H$ if $E(H') = \emptyset$, otherwise $\mathcal{L}(H') = \mathsf{void}$. Then $\mathcal{L}$-ARP is the Edge-disjoint Planar Superposition.

5.2 Modifications to planar graph with additional properties

Let $\mathcal{G}$ be a graph property, i.e, a subset of the set of all graphs. We consider the following extension of $\mathcal{L}$-RP.

$\mathcal{L}$-Replacement to a Planar Graph with property $\mathcal{G}$ ($\mathcal{L}$-RPP($\mathcal{G}$)).
Input: A graph G and a non-negative integer k.
Question: is there a $\varphi \in \mathbf{inj}([k], V(G))$, such that $\mathcal{L}_\varphi(G)$ a planar graph in $\mathcal{G}$?

We now provide some instantiations of $\mathcal{G}$ for which the $\mathcal{L}$-RPP($\mathcal{G}$) belongs in FPT, when parameterized by k. In each case we explain how our algorithm should be modified.

(1) $\mathcal{G} := \mathcal{G}_H$ is the set of all H-subgraph-free graphs, for some connected graph H. In the definitions of an annulus-boundaried graph and wall embedded separators (see Subsection 4.2) instead of taking a 3-sd-wall, we now consider an r-sd-wall where r is the diameter of H. This permits the modification of the conclusion of Observation 4 to "G is H-subgraph free and planar if and only if both C_{in} and C_{out} are H-subgraph free and planar". Also we may enhance the definition of a replacement folio (defined formally in Subsection 4.3) by considering in Condition 3 $\hat{U}_{\text{in}}$ to be planar and H-subgraph free and in Condition 5, we use a revised version of the equivalence $\equiv$ where we instead demand in (1) that $\mathbf{F} \oplus \mathbf{G}_1 \in \mathcal{P} \cap \mathcal{G} \iff \mathbf{F} \oplus \mathbf{G}_2 \in \mathcal{P} \cap \mathcal{G}$. Notice that for every H, $\mathcal{G}_H$ is a MSOL-expressible property (actually it is even expressible in First Order Logic) therefore the revised $\equiv$ also has finite index.

(2) $\mathcal{G} = \mathcal{G}_H$ is the set of all induced H-subgraph-free graphs, for some connected graph H. The modifications are exactly the same as in the previous case. Just take into account that H-induced minor freeness is MSOL-expressible.

(3) $\mathcal{G} := \mathcal{G}_d$ is the set of all d-regular graphs for $d \in \{3, 4, 5\}$. In this case, one can easily verify that the following relaxed version of the conclusion of Observation 4 holds: "G is d-regular and planar if and only if both C_{in} and C_{out} are almost d-regular and planar". Here by "almost" we demand the vertices of the 3-sd wall $\mathbb{A} = (A, N_{\text{in}}, N_{\text{out}})$ that are on N_{out} to have degree at most d and all the others to have degree exactly d. Notice that this relaxed regularity condition is again expressible in MSOL-logic (for this we need to annotate the vertices in N_{out}). As in the previous cases, we can again demand, in Condition 3, that $\hat{U}_{\text{in}}$ is planar and almost 3-regular and, in Condition 5 enhance the definition of the equivalence relation by additionally asking almost regularity in (1).

(4) $\mathcal{G}$ is the set of all Eulerian graphs. This is the same as in the previous case. The only difference is that we now ask degrees to be even. All modifications are are parallel to the previous case (notice that connectivity demand for Eulerian graphs is already incorporated in the definition of $\equiv$). However, asking for a graph to have all vertices, except possibly from some annotated set, of even degree is not MSOL-expressible. However, we can use an extension of MSOL, called *Counting MSOL* (CMSOL) that is MSOL with an additional predicate $\mathbf{card}_{q,r}$ where $(G, S) \models \mathbf{card}_{q,r} \iff |S| \equiv q \pmod r$. It is known (see e.g. [3, Lemma 3.2]) that equivalence relations on CMSOL-expressible properties have finite index. This permits us to deal with the parity demand of being Eulerian.

(5) $\mathcal{G}$ is the set of all bipartite graphs. As before, one can easily derive a version of Observation 4 where the conclusion is: "G is planar bipartite if and only if both C_{in} and C_{out} are planar bipartite'. The modifications are analogous to those of the previous cases where the property is the exclusion of an odd cycle, which is CMSOL-expressible.

(6) $\mathcal{G}$ is the set of all triangulated graphs. Notice that $\mathcal{G}$ contains every graph G that has exactly $3|G|-6$ edges. Let (G,k) be an instance of $\mathcal{L}$-RPP$(\mathcal{G})$ and let $t_G = |E(G)|-(3|V(G)|-6)$. As an action cannot remove or introduce more than $\binom{k}{2}$ edges, we assume that $-\binom{k}{2} \leq t_G \leq \binom{k}{2}$, otherwise (G,k) is a no-instance. Under this demand, $\mathcal{L}$ should change to $\mathcal{L}'$ where $\mathcal{L}'(H) = \mathcal{L}(H)$ if $|E(H)|-|E(\mathcal{L}(H))| = t_G$ and $\mathcal{L}'(H) = \mathsf{void}$ otherwise. As the action $\mathcal{L}''$ when applied on G always creates a graph G' that has $3|G'|-6$ edges and in the case G' is planar, then it should also be triangulated. Therefore (G,k) is a yes instance of $\mathcal{L}$-ARP$(\mathcal{G})$ iff (G',k) is a yes instance of $\mathcal{L}'$-ARP.

(7) $\mathcal{G}$ is the set of all radial graphs. A radial graph is a graph that can be embedded in the plane so that all its faces are squares. Recall that a graph G is radial if it is planar, bipartite, and has $2|G|-4$ edges. We apply the same reduction as in the previous case for $t_G = |E(G)|-(2|V(G)|-4)$ and we have that $\mathcal{L}$-RPP$(\mathcal{G})$ iff (G',k) is a yes instance of $\mathcal{L}'$-ARP$(\mathcal{G}')$ where $\mathcal{G}'$ is the class of all bipartite graphs (treated in (5)).

6 Conclusions and open problems

In this paper we proved that for every editing operation that is based on adjacency modifications, the planarization problem is FPT, when parameterized by the number of edges that are changed during this modification. We have also seen that the formalization of modification problem by actions is quite versatile and can express most known modifications problems of this flavour. The are three possible extensions of our results that could induce further research on this topic.

First one may consider more general modifications where some (bounded) part of the graph is replaced by another, however not of the same size but still bounded. We believe that this setting is amenable to the techniques that we introduce in this paper, however more complicated to deal with. Also, one may consider modifications that involve the whole (unbounded) neighbourhood of a bounded part of the graph. Contractions or vertex removals would fit in such a more general framework.

Second one may consider, instead of planar graphs, other classes of graphs such as graphs of bounded genus or graphs excluding a graph as a minor, where one may still employ structural results about "flat territories". We believe that our machinery can be extended in this direction. However, such extensions should be quite non-trivial as they involve several technicalities on how separators may split graphs in those families and how actions may rearrange them.

A third direction is to find more examples of, additional to planarity, target properties. Most of the examples in Subsection 5.2 demand changes either to the way Observation 4 applies or to the equivalence relation $\equiv$ that may still keep it of finite index. Is there a way to systematize this into a general meta-algorithmic framework?

References

1 Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Tight Bounds for Linkages in Planar Graphs. In *Proceedings of the 38th International Colloquium of Automata, Languages and Programming (ICALP)*, volume 6755 of *Lecture Notes in Comput. Sci.*, pages 110–121. Springer, 2011.

2 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Optimal Algorithms for Hitting (Topological) Minors on Graphs of Bounded Treewidth. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 4:1–4:12, 2017.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *J. ACM*, 63(5):44:1–44:69, 2016.

4 Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Spyridon Maniatis, Clément Requilé, Dimitrios M. Thilikos, and Dimitris Zoros. FPT algorithms for plane completion problems. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 26:1–26:13, 2016.

5 Bruno Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. *Handbook of Graph Grammars*, pages 313–400, 1997.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

7 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. The planar directed k-Vertex-Disjoint Paths problem is fixed-parameter tractable. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 197–207. IEEE, 2013.

8 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting Forbidden Minors: Approximation and Kernelization. *SIAM Journal on Discrete Mathematics*, 30(1):383–410, 2016. doi:10.1137/140997889.

9 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 470–479. IEEE, 2012.

10 Archontia C Giannopoulou, Bart MP Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Transactions on Algorithms*, 13(3):35, 2017.

11 Archontia C Giannopoulou, Michał Pilipczuk, Dimitrios M Thilikos, Jean-Florent Raymond, and Marcin Wrochna. Linear kernels for edge deletion problems to immersion-closed graph classes. *arXiv preprint arXiv:1609.07780*, 2016.

12 Petr A. Golovach, Marcin Kaminski, Spyridon Maniatis, and Dimitrios M. Thilikos. The Parameterized Complexity of Graph Cyclability. *SIAM J. Discrete Math.*, 31(1):511–541, 2017. doi:10.1137/141000014.

13 Petr A. Golovach, Pim van 't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theor. Comput. Sci.*, 476:38–46, 2013. doi:10.1016/j.tcs.2012.12.041.

14 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 479–488. ACM, 2011.

15 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A Near-optimal Planarization Algorithm. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1802–1811. SIAM, 2014. URL: http://dl.acm.org/citation.cfm?id=2634074.2634204.

16 Ken-ichi Kawarabayashi. Planarity Allowing Few Error Vertices in Linear Time. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 639–648. IEEE, 2009.

17 Ken-ichi Kawarabayashi and Yusuke Kobayashi. The induced disjoint path problem. In *13th Conference on Integer Programming and Combinatorial Optimization, IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 47–61. Springer, Berlin, 2008.

18 Ken-ichi Kawarabayashi and Bojan Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 471–480. ACM, 2008.

19 Ken-ichi Kawarabayashi, Bojan Mohar, and Bruce A. Reed. A Simpler Linear Time Algorithm for Embedding Graphs into an Arbitrary Surface and the Genus of Graphs of Bounded Tree-Width. In *Proceedings of the 49th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 771–780. IEEE Computer Society, 2008.

20 Ken-ichi Kawarabayashi and Bruce A. Reed. Odd cycle packing. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 695–704, 2010.

21 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear Kernels and Single-Exponential Algorithms Via Protrusion Decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016. doi:10.1145/2797140.

22 Dániel Marx. Chordal Deletion is Fixed-Parameter Tractable. *Algorithmica*, 57(4):747–768, 2010. doi:10.1007/s00453-008-9233-8.

23 Dániel Marx and Ildikó Schlotter. Obtaining a Planar Graph by Vertex Deletion. *Algorithmica*, 62(3-4):807–822, 2012.

24 Neil Robertson and Paul D. Seymour. Graph Minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(2):92–114, 1986.

25 Neil Robertson and Paul D. Seymour. Graph Minors. XIII. The disjoint paths problem. *J. Comb. Theory Ser. B*, 63(1):65–110, 1995.

Visibly Pushdown Languages over Sliding Windows

Moses Ganardi
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Abstract

We investigate the class of visibly pushdown languages in the sliding window model. A sliding window algorithm for a language L receives a stream of symbols and has to decide at each time step whether the suffix of length n belongs to L or not. The window size n is either a fixed number (in the fixed-size model) or can be controlled by an adversary in a limited way (in the variable-size model). The main result of this paper states that for every visibly pushdown language the space complexity in the variable-size sliding window model is either constant, logarithmic or linear in the window size. This extends previous results for regular languages.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases visibly pushdown languages, sliding windows, rational transductions

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.29

Related Version A full version of the paper is available at [18], `https://arxiv.org/abs/1812.11549`.

Funding The author is supported by the DFG project LO 748/13-1.

1 Introduction

The sliding window model. A *sliding window algorithm (SWA)* is an algorithm which processes a stream of data elements $a_1a_2a_3\cdots$ and computes at each time instant t a certain value that depends on the suffix $a_{t-n+1}\cdots a_t$ of length n where n is a parameter called the *window size*. This streaming model is motivated by the fact that in many applications data elements are outdated or become irrelevant after a certain time. A general goal in the area of sliding window algorithms is to avoid storing the window content explicitly (which requires $\Omega(n)$ bits) and to design space efficient algorithms, say using polylogarithmic many bits in the window size n.

A prototypical example of a problem considered in the sliding window model is the Basic Counting problem. Here the input is a stream of bits and the task is to approximate the number of 1's in the last n bits (the *active window*). In [15], Datar, Gionis, Indyk and Motwani present an approximation algorithm using $O(\frac{1}{\epsilon}\log^2 n)$ bits of space with an approximation ratio of ϵ. They also prove a matching lower bound of $\Omega(\frac{1}{\epsilon}\log^2 n)$ bits for any deterministic (and even randomized) algorithm for Basic Counting. Other works in the sliding window model include computing statistics [2, 3, 8], optimal sampling [9] and various pattern matching problems [10, 12, 13, 14].

There are two variants of the sliding window model, cf. [2]. One can think of an adversary who can either insert a new element into the window or remove the oldest element from the window. In the *fixed-size* sliding window model the adversary determines the window size n in the beginning and the initial window is set to a^n for some default known element a. At every time step the adversary inserts a new symbol and then immediately removes the oldest element from the window. In the *variable-size* sliding window model the window size is initially set to $n = 0$. Then the adversary is allowed to perform an arbitrary sequence of insert- and remove-operations. A remove-operation on an empty window leaves the window

empty. We also mention the timestamp-based model where every element carries a timestamp (many elements may have the same timestamp) and the active window at time t contains only those elements whose timestamp is at least $t - t_0$ for some parameter t_0 [9]. Both the fixed-size and the timestamp-based model can be simulated in the variable-size model.

Regular languages. In a recent series of works we studied the membership problem to a fixed regular language in the sliding window model. It was shown in [21] that in both the fixed-size and the variable-size sliding window model the space complexity of any regular language is either constant, logarithmic or linear (a *space trichotomy*). In a subsequent paper [19] a characterization of the space classes was given: A regular language has a fixed/variable-size SWA with $O(\log n)$ bits if and only if it is a finite Boolean combination of regular left ideals and regular length languages. A regular language has a fixed-size SWA with $O(1)$ bits if and only if it is a finite Boolean combination of suffix testable languages and regular length languages. A regular language has a variable-size SWA with $O(1)$ bits if and only if it is empty or universal.

Context-free languages. A natural question is whether the results above can be extended to larger language classes, say subclasses of the context-free languages. More precisely, we pose the questions: (i) Which language classes have a "simple" hierarchy of space complexity classes (like the space trichotomy for the regular languages), and (ii) are there natural descriptions of the space classes? A positive answer to question (i) seems to be necessary to answer question (ii) positively. In [22] we presented a family of context-free languages $(L_k)_{k\geq 1}$ which have space complexity $\Theta(n^{1/k})$ in the variable-size model and $O(n^{1/k}) \setminus o(n^{1/k})$ in the fixed-size model, showing that there exists an infinite hierarchy of space complexity classes inside the class of context-free languages. Intuitively, this result can be explained with the fact that a language and its complement have the same sliding window space complexity; however, the class of context-free languages is not closed under complementation (in contrast to the regular languages) and the analysis of co-context-free languages in this setting seems to be very difficult. Even in the class of deterministic context-free languages, which is closed under complementation, there are example languages which have sliding window space complexity $\Theta((\log n)^2)$ [22].

Visibly pushdown languages. Motivated by these observations in this paper we will study the class of *visibly pushdown languages*, introduced by Alur and Madhusudan [1]. They are recognized by *visibly pushdown automata* where the alphabet is partitioned into *call letters*, *return letters* and *internal letters*, which determine the behavior of the stack height. Since visibly pushdown automata can be determinized, the class of visibly pushdown languages turns out to be very robust (it is closed under Boolean operations and other language operations) and to be more tractable in many algorithmic questions than the class of context-free languages [1]. In this paper we prove a space trichotomy for the class of visibly pushdown languages in the variable-size sliding window model, stating that the space complexity of every visibly pushdown language is either $O(1)$, $\Theta(\log n)$ or $O(n) \setminus o(n)$. The main technical result is a growth theorem (Theorem 6) for rational transductions. A natural characterization of the $O(\log n)$-class as well as a study of the fixed-size model are left as open problems.

Let us mention some related work in the context of streaming algorithms for context-free languages. Randomized streaming algorithms were studied for subclasses of context-free languages (DLIN and LL(k)) [4] and for Dyck languages [25]. A streaming property tester for visibly pushdown languages was presented by François et al. [17].

2 Preliminaries

We define $\log n = \lfloor \log_2 n \rfloor$ for all $n \geq 1$, which is the minimum number k of bits required to encode n elements using bit strings of length *at most* k. If $w = a_1 \cdots a_n$ is a word then any word of the form $a_i \cdots a_n$ ($a_1 \cdots a_i$) is called *suffix (prefix)* of w. A prefix (suffix) v of w is *proper* if $v \neq w$. A *factor* of w is any word of the form $a_i \cdots a_j$. A *factorization* of w is formally a sequence of possibly empty factors $(w_0, \ldots, w_m)$ with $w = w_0 \cdots w_m$. We call w_0 the *initial* factor and $w_1, \ldots, w_m$ the *internal* factors. The *reversal* of w is $w^{\mathsf{R}} = a_n a_{n-1} \cdots a_1$. For a language $L \subseteq \Sigma^*$ we denote by $\mathsf{Suf}(L)$ the set of suffixes of words in L. If $L = \mathsf{Suf}(L)$ then L is *suffix-closed*.

Automata. An *automaton* over a monoid M is a tuple $A = (Q, M, I, \Delta, F)$ where Q is a finite set of *states*, $I \subseteq Q$ is a set of *initial states*, $\Delta \subseteq Q \times M \times Q$ is the *transition relation* and $F \subseteq Q$ is the set of *final states*. A *run* on $m \in M$ from q_0 to q_n is a sequence of transitions of the form $\pi = (q_0, m_1, q_1)(q_1, m_2, q_2) \cdots (q_{n-1}, m_n, q_n) \in \Delta^*$ such that $m = m_1 \cdots m_n$. We usually depict π as $q_0 \xrightarrow{m_1} q_1 \xrightarrow{m_2} q_2 \cdots q_{n-1} \xrightarrow{m_n} q_n$, or simply $q_0 \xrightarrow{m} q_n$. It is *initial* if $q_0 \in I$ and *accepting* if $q_n \in F$. The *language* defined by A is the set $\mathsf{L}(A)$ of all elements $m \in M$ such that there exists an initial accepting run on m. A subset $L \subseteq M$ is *rational* if $L = \mathsf{L}(A)$ for some automaton A. We only need the case where M is the free monoid Σ^* over an alphabet Σ or where M is the product $\Sigma^* \times \Omega^*$ of two free monoids. In these cases we change the format and write $(Q, \Sigma, I, \Delta, F)$ and $(Q, \Sigma, \Omega, I, \Delta, F)$, respectively. Subsets of Σ^* are called *languages* and subsets of $\Sigma^* \times \Omega^*$ are called *transductions*. Rational languages are usually called *regular languages*.

In this paper we will also use *right automata*, which read the input from right to left. Formally, a right automaton $A = (Q, M, F, \Delta, I)$ has the same format as a (left) automaton where the sets of initial and final states are swapped. Runs in right automata are defined from right to left, i.e. a run on $m \in M$ from q_n to q_0 is a sequence of transitions of the form $(q_0, m_1, q_1)(q_1, m_2, q_2) \cdots (q_{n-1}, m_n, q_n) \in \Delta^*$ such that $m = m_1 \cdots m_n$. In the graphic notation we write the arrows from right to left. It is initial (accepting) if $q_n \in I$ ($q_0 \in F$).

Right congruences. For any equivalence relation $\sim$ on a set X we write $[x]_\sim$ for the $\sim$-class containing $x \in X$ and $X/\!\sim = \{[x]_\sim \mid x \in X\}$ for the set of all $\sim$-classes. The *index* of $\sim$ is the cardinality of $X/\!\sim$. We denote by $\nu_\sim \colon X \to X/\!\sim$ the function with $\nu_\sim(x) = [x]_\sim$. A subset $L \subseteq X$ is *saturated* by $\sim$ if L is a union of $\sim$-classes. An equivalence relation $\sim$ on the free monoid Σ^* over some alphabet Σ is a *right congruence* if $x \sim y$ implies $xz \sim yz$ for all $x, y, z \in \Sigma^*$. The *Myhill-Nerode right congruence* $\sim_L$ of a language $L \subseteq \Sigma^*$ is the equivalence relation on Σ^* defined by $x \sim_L y$ if and only if $x^{-1}L = y^{-1}L$ where $x^{-1}L = \{z \mid xz \in L\}$. It is indeed the coarsest right congruence on Σ^* which saturates L. We usually write ν_L instead of $\nu_{\sim_L}$. A language $L \subseteq \Sigma^*$ is regular iff $\sim_L$ has finite index.

Rational transductions. Rational transductions are accepted by automata over $\Sigma^* \times \Omega^*$, which are called finite state transducers. In this paper, we will use a slightly extended but equivalent definition. A *transducer* is a tuple $A = (Q, \Sigma, \Omega, I, \Delta, F, o)$ such that $(Q, \Sigma^* \times \Omega^*, I, \Delta, F)$ is an automaton over $\Sigma^* \times \Omega^*$ and a *terminal output function* $o \colon F \to \Omega^*$. To omit parentheses we write runs $p \xrightarrow{(x,y)} q$ in the form $p \xrightarrow{x|y} q$ and depict $o(q) = y$ by a transition $q \xrightarrow{|y}$ without input word and target state. If π is a run $p \xrightarrow{x|y} q$ we define $\mathsf{out}(\pi) = y$ and $\mathsf{out}_F(\pi) = y\, o(q)$. The transduction defined by A is the set $\mathsf{T}(A)$ of all pairs $(x, \mathsf{out}_F(\pi))$ such that π is an initial accepting run $p \xrightarrow{x|y} q$. Since the terminal output

function can be eliminated by ε-transitions, a transduction is rational if and only if it is of the form $\mathsf{T}(A)$ for some transducer A. In this paper we will mainly use *rational functions*, which are partial functions $t\colon \Sigma^* \to \Omega^*$ whose graph $\{(x, t(x)) \mid x \in \mathrm{dom}(t)\}$ is a rational transduction.

A transducer A is *trim* if every state occurs on some accepting run. If every word $x \in \Sigma^*$ has at most one initial accepting run $p \xrightarrow{x|y} q$ for some $y \in \Omega^*$ then A is *unambiguous*. If $\Delta \subseteq Q \times \Sigma \times \Omega^* \times Q$ then A is *real-time*. It is known that every rational function is defined by a trim unambiguous real-time transducer [6, Corollary 4.3]. If A is unambiguous and trim then for every word $x \in \Sigma^*$ and every pair of states $(p, q) \in Q^2$ there exists at most one run from p to q with input word x. Therefore, the state pair (p, q) and the input word x uniquely determine the run (if it exists) and we can simply write $p \xrightarrow{x} q$. Similarly to [28], we define for a real-time transducer A the parameter $\mathsf{iml}(A) = \max(\{|y| \mid (q, a, y, p) \in \Delta\} \cup \{|o(q)| \mid q \in Q\})$. For every run π on a word $x \in \Sigma^*$ we have $|\mathsf{out}(\pi)| \leq \mathsf{iml}(A) \cdot |x|$ and $|\mathsf{out}_F(\pi)| \leq \mathsf{iml}(A) \cdot (|x|+1)$.

The following closure properties for rational transductions are known [6]: The class of rational transductions is closed under inverse, reversal and composition where the *inverse* of T is $T^{-1} = \{(y, x) \mid (x, y) \in T\}$, the *reversal* of T is $T^{\mathsf{R}} = \{(x^{\mathsf{R}}, y^{\mathsf{R}}) \mid (x, y) \in T\}$, and the composition of two transductions T_1, T_2 is $T_1 \circ T_2 = \{(x, z) \mid \exists y : (x, y) \in T_1 \text{ and } (y, z) \in T_2\}$. If $T \subseteq \Sigma^* \times \Omega^*$ is rational and $L \subseteq \Sigma^*$ is regular then the restriction $\{(x, y) \in T \mid x \in L\}$ is also rational. If $K \subseteq \Sigma^*$ is regular (context-free) and $T \subseteq \Sigma^* \times \Omega^*$ is rational then $TK = \{y \in \Omega^* \mid (x, y) \in T \text{ for some } x \in K\}$ is also regular (context-free).

A *right transducer* is a tuple $A = (Q, \Sigma, \Omega, F, \Delta, I, o)$ such that $(Q, \Sigma^* \times \Omega^*, F, \Delta, I)$ is a right automaton over $\Sigma^* \times \Omega^*$ and a *terminal output function* $o\colon F \to \Omega^*$. We depict $o(q) = y$ by a transition $\xleftarrow{|y} q$. If π is a run $q \xleftarrow{x|y} p$ we define $\mathsf{out}(\pi) = y$ and $\mathsf{out}_F(\pi) = o(q)\, y$. All other notions on transducers are defined for right transducers in a dual way.

Growth functions. A function $\gamma\colon \mathbb{N} \to \mathbb{N}$ grows *polynomially* if $\gamma(n) \in O(n^k)$ for some $k \in \mathbb{N}$; we say that γ grows *exponentially* if there exists a number $c > 1$ such that $\gamma(n) \geq c^n$ for infinitely many $n \in \mathbb{N}$. A function $\gamma(n)$ grows exponentially if and only if $\log \gamma(n) \notin o(n)$.

We will define a generalized notion of growth. Let $t\colon \Sigma^* \to Y$ be a partial function and let $X \subseteq \mathrm{dom}(t)$ be a language. The *t-growth* of X is the function $\gamma(n) = |t(X \cap \Sigma^{\leq n})|$, i.e. it counts the number of output elements on input words from X of length at most n. The *growth* of X is simply the id_X-growth of X, i.e. $\gamma(n) = |X \cap \Sigma^{\leq n}|$. It is known that every context-free language has either polynomial or exponential growth [23]. Furthermore, a context-free language L has polynomial growth if and only if it is *bounded*, i.e. $L \subseteq w_1^* \cdots w_k^*$ for some words $w_1, \ldots, w_k$ [23]. We need the fact that if L is a bounded language and K is a set of factors of words in L then K is bounded [24, Lemma 1.1(c)].

3 Visibly pushdown languages

A *pushdown alphabet* is a triple $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_{int})$ consisting of three pairwise disjoint alphabets: a set of *call letters* Σ_c, a set of *return letters* Σ_r and a set of *internal letters* Σ_{int}. We identify $\tilde{\Sigma}$ with the union $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$. The set of *well-matched* words W over Σ is defined as the smallest set which contains $\{\varepsilon\} \cup \Sigma_{int}$, is closed under concatenation, and if w is well-matched, $a \in \Sigma_c$, $b \in \Sigma_r$ then also awb is well-matched. A word is called *descending (ascending)* if it can be factorized into well-matched factors and return (call) letters. The set of descending words is denoted by D. A *visibly pushdown automaton (VPA)* has the form $A = (Q, \tilde{\Sigma}, \Gamma, \bot, q_0, \delta, F)$ where Q is a finite state set, $\tilde{\Sigma}$ is a pushdown alphabet, Γ is the finite stack alphabet containing a special symbol $\bot$ (representing the empty stack), $q_0 \in Q$ is

the initial state, $F \subseteq Q$ is the set of final states and $\delta = \delta_c \cup \delta_r \cup \delta_{int}$ is the transition function where $\delta_c \colon Q \times \Sigma_c \to (\Gamma \setminus \{\bot\}) \times Q$, $\delta_r \colon Q \times \Sigma_r \times \Gamma \to Q$ and $\delta_{int} \colon Q \times \Sigma_{int} \to Q$. The set of *configurations* Conf is the set of all words αq where $q \in Q$ is a state and $\alpha \in \bot(\Gamma \setminus \{\bot\})^*$ is the *stack content*. We define $\delta \colon \mathrm{Conf} \times \Sigma \to \mathrm{Conf}$ for each $p \in Q$ and $a \in \Sigma$ as follows:

- If $a \in \Sigma_c$ and $\delta(p, a) = (\gamma, q)$ then $\delta(\alpha p, a) = \alpha\gamma q$.
- If $a \in \Sigma_{int}$ and $\delta(p, a) = q$ then $\delta(\alpha p, a) = \alpha q$.
- If $a \in \Sigma_r$, $\delta(p, a, \gamma) = q$ and $\gamma \in \Gamma \setminus \{\bot\}$ then $\delta(\alpha\gamma p, a) = \alpha q$.
- If $a \in \Sigma_r$ and $\delta(p, a, \bot) = q$ then $\delta(\bot p) = \bot q$.

As usual we inductively extend δ to a function $\delta \colon \mathrm{Conf} \times \Sigma^* \to \mathrm{Conf}$ where $\delta(c, \varepsilon) = c$ and $\delta(c, wa) = \delta(\delta(c, w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$. The *initial* configuration is $\bot q_0$ and a configuration c is *final* if $c \in \Gamma^* F$. A word $w \in \Sigma^*$ is *accepted* from a configuration c if $\delta(c, w)$ is final. The VPA A *accepts* w if w is accepted from the initial configuration. The set of all words accepted by A is denoted by $\mathsf{L}(A)$; the set of all words accepted from c is denoted by $\mathsf{L}(c)$. A language L is a *visibly pushdown language (VPL)* if $L = \mathsf{L}(A)$ for some VPA A. To exclude some pathological cases we assume that $\Sigma_c \neq \emptyset$ and $\Sigma_r \neq \emptyset$. In fact, if $\Sigma_c = \emptyset$ or $\Sigma_r = \emptyset$ then any VPL over that pushdown alphabet would be regular.

One can also define nondeterministic visibly pushdown automata in the usual way, which can always be converted into deterministic ones [1]. This leads to good closure properties of the class of all VPLs, as closure under Boolean operations, concatenation and Kleene star.

The set W of well-matched words forms a submonoid of Σ^*. Notice that a VPA can only see the top of the stack when reading return symbols. Therefore, the behavior of a VPA on a well-matched word is determined only by the current state and independent of the current stack content. More precisely, there exists a monoid homomorphism $\varphi \colon W \to Q^Q$ into the finite monoid of all state transformations $Q \to Q$ such that $\delta(\alpha p, w) = \alpha\varphi(w)(p)$ for all $w \in W$ and $\alpha p \in \mathrm{Conf}$.

4 Sliding window algorithms and main results

In our context a *streaming algorithm* is a deterministic algorithm A which reads an input word $a_1 \cdots a_m \in \Sigma^*$ symbol by symbol from left to right and outputs after every prefix either 1 or 0. We view A as a deterministic (possibly infinite) automaton whose states are encoded by bit strings and thus abstract away from the actual computation, see [19] for a formal definition. A *variable-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ is a streaming algorithm A which reads an input word $a_1 \cdots a_m$ over the extended alphabet $\overline{\Sigma} = \Sigma \cup \{\downarrow\}$. The symbol $\downarrow$ is the operation which removes the oldest symbol from the window. At time $0 \leq t \leq m$ the algorithm has to decide whether the *active window* $\mathrm{wnd}(a_1 \cdots a_t)$ belongs to L which is defined by

$$\begin{aligned} \mathrm{wnd}(\varepsilon) &= \varepsilon & \mathrm{wnd}(u\downarrow) &= \varepsilon \text{ if } \mathrm{wnd}(u) = \varepsilon \\ \mathrm{wnd}(ua) &= \mathrm{wnd}(u)a & \mathrm{wnd}(u\downarrow) &= v \text{ if } \mathrm{wnd}(u) = av \end{aligned}$$

for $u \in \Sigma^*$, $a \in \Sigma$. For example, a variable-size sliding window algorithm A for the language $L_a = \{w \in \{a, b\}^* \mid w \text{ contains } a\}$ maintains the window length n and the position i (from the right) of the most recent a-symbol in the window (if it exists): We initialize $n := 0$ and $i := \infty$. On input a we increment n and set $i := 1$, on input b we increment both n and i. On input $\downarrow$ we decrement n, unless $n = 0$, and then set $i := \infty$ if $i > n$.

The *space complexity* of A is the function which maps n to the maximum number of bits used when reading an input $a_1 \cdots a_m$ where the window size never exceeds n, i.e. $|\mathrm{wnd}(a_1 \cdots a_t)| \leq n$ for all $0 \leq t \leq n$. Notice that this function is monotonic. For every

language L there exists a space optimal variable-size sliding window algorithm [20, Lemma 3.1] and we write $V_L(n)$ for its space complexity. Clearly we have $V_L(n) \in O(n)$. For example the example language L_a above satisfies $V_{L_a}(n) \in O(\log n)$ because the algorithm above only maintains two numbers using $O(\log n)$ bits. The main result of this paper states:

▶ **Theorem 1** (Trichotomy for VPL). *If L is a visibly pushdown language then $V_L(n)$ is either $O(1)$, $\Theta(\log n)$ or $O(n) \setminus o(n)$.*

In the rest of this section we will give an overview of the proof of Theorem 1.

Suffix expansions. Let $\sim$ be an equivalence relation on Σ^*. The *suffix expansion* of $\sim$ is the equivalence relation $\approx$ on Σ^* defined by $a_1 \cdots a_n \approx b_1 \cdots b_m$ if and only if $n = m$ and $a_i \cdots a_n \sim b_i \cdots b_n$ for all $1 \leq i \leq n$. Notice that $\approx$ saturates each subset $\Sigma^{\leq n}$. Furthermore, if $\sim$ is a right congruence then so is $\approx$ since $|u| = |v|$ implies $|ua| = |va|$ and $a_i \cdots a_n \sim b_i \cdots b_n$ implies $a_i \cdots a_n a \sim b_i \cdots b_n a$. We also define suffix expansions for partial functions $t\colon \Sigma^* \to Y$ with suffix-closed domain $\mathrm{dom}(t)$. The *suffix expansion* of t is the total function $\overleftarrow{t}\colon \mathrm{dom}(t) \to Y^*$ defined by $\overleftarrow{t}(a_1 \cdots a_n) = t(a_1 \cdots a_n)\, t(a_2 \cdots a_n) \cdots t(a_{n-1}a_n)\, t(a_n)$ for all $a_1 \cdots a_n \in \Sigma^*$. Here the range of $\overleftarrow{t}$ is the free monoid (alternatively, the set of all sequences) over Y. If $\sim$ is an equivalence relation on Σ^* then its suffix expansion $\approx$ is the *kernel* of $\overleftarrow{\nu}_\sim$, i.e. $x \approx y$ if and only if $\overleftarrow{\nu}_\sim(x) = \overleftarrow{\nu}_\sim(y)$. The space complexity in the variable-size model is captured by the suffix expansion $\approx_L$ of the Myhill-Nerode right congruence $\sim_L$ or alternatively by the suffix expansion $\overleftarrow{\nu}_L$ of ν_L.

▶ **Theorem 2** ([19, Theorem 4.1]). *For all $\emptyset \subsetneq L \subsetneq \Sigma^*$ we have $V_L(n) = \log|\Sigma^{\leq n}/{\approx_L}| = \log|\overleftarrow{\nu}_L(\Sigma^{\leq n})|$. In particular, $V_L(n) = \Omega(\log n)$ for every non-trivial language.*

If L is empty or universal, then $V_L(n) \in O(1)$ and otherwise $V_L(n) = \Omega(\log n)$. Hence to prove Theorem 1 it suffices to show that either $V_L(n) \in O(\log n)$ or $V_L(n) \notin o(n)$ holds for every VPL L. If L is a regular language and A is the minimal DFA of L with state set Q, one can identify $\nu_L(x)$ with the state $q \in Q$ reached on input x. Hence, $\overleftarrow{\nu}_L(x)$ is represented by a word over Q. Using the transition monoid of A one can show that $\overleftarrow{\nu}_L\colon \Sigma^* \to Q^*$ is rational (in fact *right-subsequential*, see Section 6) and hence the image $\overleftarrow{\nu}_L(\Sigma^*) \subseteq Q^*$ is regular [20, Lemma 4.2]. Since the growth of $\overleftarrow{\nu}_L(\Sigma^*)$ is either polynomial or exponential this implies that $V_L(n) \in O(\log n)$ or $V_L(n) \notin o(n)$.

Restriction to descending words. The approach above for regular languages can be extended to visibly pushdown languages L if we restrict ourselves to the set D of descending words. If a VPA with state set Q reads a descending word $x \in D$ from the initial configuration it reaches some configuration $\bot q$ with empty stack. Notice that there may be distinct configurations $\bot p \neq \bot q$ with $\mathsf{L}(\bot p) = \mathsf{L}(\bot q)$, in which case we need to pick a single representative. Since every suffix of x is again descending we can represent $\overleftarrow{\nu}_L(x)$ by a word $\sigma_0(x) \in Q^*$ and in fact we will prove that $S_0 = \sigma_0(D)$ is a context-free language (Lemma 10). By the growth theorem for context-free languages the growth of S_0 is either polynomial or exponential. If S_0 grows exponentially we obtain an exponential lower bound on $|\overleftarrow{\nu}_L(\Sigma^{\leq n})|$ (Lemma 11). Hence, the interesting case is that S_0 has polynomial growth, i.e. S_0 is bounded.

Representation by rational functions. In order to simulate a VPA by a finite automaton on arbitrary words we will "flatten" the input word in the following way. The input word w is factorized $w = w_0 w_1 \cdots w_m$ into a descending prefix w_0, and call letters and well-matched factors $w_1, \ldots, w_m$. The descending prefix w_0 is replaced by $\sigma_0(w_0)$ and each well-matched

factor w_i is replaced by a similar information $\sigma_1(w_i)$ which describes the behavior of the VPA on the factor w_i and on each of its suffixes. The set Flat of all flattenings is a context-free language. Furthermore, there exists a rational function ν_f such that, if a flattening s represents a word $w \in \Sigma^*$ then $\nu_f(s)$ is a configuration representing the Myhill-Nerode class $\nu_L(w)$ (Proposition 9). Hence, we can reduce proving the main theorem to the question whether the $\overleftarrow{\nu}_f$-growth of Flat is always either polynomial or exponential.

This question is resolved positively as follows. We prove that for every rational function t with suffix-closed domain $X = \mathrm{dom}(t)$ the $\overleftarrow{t}$-growth of X is either polynomial or exponential (Theorem 6). In the case that S_0 has polynomial growth we can overapproximate Flat by a regular superset RegFlat. If the $\overleftarrow{\nu}_f$-growth of RegFlat is polynomial then the same holds trivially for the subset Flat. If the $\overleftarrow{\nu}_f$-growth of RegFlat is exponential then the proper choice of RegFlat ensures that Flat also has exponential $\overleftarrow{\nu}_f$-growth (Proposition 14).

Dichotomy for rational functions. The main technical result of this paper states that for every rational function $t\colon \Sigma^* \to \Omega^*$ with suffix-closed domain $X = \mathrm{dom}(t)$ the $\overleftarrow{t}$-growth of X is either polynomial or exponential. We emphasize that the range of $\overleftarrow{t}$ is not Ω^* but the free monoid over Ω^* (consisting of all finite sequences of words over Ω). There are in fact two reasons for exponential $\overleftarrow{t}$-growth: (i) The image $t(X)$ has exponential growth, and (ii) X contains a so called linear fooling set. We need these lower bounds in the more general setting where $X \subseteq \mathrm{dom}(t)$ is a context-free subset, namely $X = \mathrm{Flat}$.

▶ **Proposition 3.** *Let $t\colon \Sigma^* \to \Omega^*$ be rational with suffix-closed domain. If $X \subseteq \mathrm{dom}(t)$ is context-free and $t(X)$ has exponential growth then X has exponential t-growth and exponential $\overleftarrow{t}$-growth.*

▶ **Example 4.** Consider the transduction $f\colon \{a,b\}^* \to a^*$ defined by

$$f = \{(a^n, a^n) \mid n \in \mathbb{N}\} \cup \{(a^n b w, a^n) \mid n \in \mathbb{N}, w \in \{a,b\}^*\},$$

which projects a word over $\{a,b\}$ to its left-most (maximal) a-block and is rational. Its image $\overleftarrow{f}(\{a,b\}^*)$ can be identified with the set of all sequences of natural numbers which are concatenations of monotonically decreasing sequences of the form $(k, k-1, \ldots, 0)$. There are exactly 2^n of such sequences of length n and hence $\{a,b\}^*$ has exponential $\overleftarrow{f}$-growth.

A *linear fooling scheme* for a partial function $t\colon \Sigma^* \to Y$ is a tuple (u_2, v_2, u, v, Z) where $u_2, v_2, u, v \in \Sigma^*$ and $Z \subseteq \Sigma^*$ such that u_2 is a suffix of u and v_2 is a suffix of v, $|u_2| = |v_2|$, $\{u_2, v_2\}\{u,v\}^* Z \subseteq \mathrm{dom}(t)$ and for all $n \in \mathbb{N}$ there exists a word $z_n \in Z$ of length $|z_n| \leq O(n)$ such that $t(u_2 w z_n) \neq t(v_2 w z_n)$ for all $w \in \{u,v\}^{\leq n}$. The set $\{u_2, v_2\}\{u,v\}^* Z$ is called a *linear fooling set* for t. Notice that the definition implies that $u_2 \neq v_2$ and hence u is not a suffix of v, and vice versa, i.e. $\{u,v\}$ is a *suffix code*. Therefore $\{u,v\}^n$ contains 2^n words of length $O(n)$ and thus $\{u_2, v_2\}\{u,v\}^*$ has exponential growth.

▶ **Proposition 5.** *Let $t\colon \Sigma^* \to \Omega^*$ be a partial function with suffix-closed domain. If $X \subseteq \mathrm{dom}(t)$ contains a linear fooling set for t then the $\overleftarrow{t}$-growth of X is exponential.*

Proof. Let (u_2, v_2, u, v, Z) be a linear fooling scheme with $\{u_2, v_2\}\{u,v\}^* Z \subseteq X$. Let $n \in \mathbb{N}$ and let $z_n \in Z$ with the properties from the definition. Consider two distinct words $w, w' \in \{u,v\}^n$. Without loss of generality the words have the form $w = w_1 u w_2$ and $w' = w_3 v w_2$ for some $w_1, w_2, w_3 \in \{u,v\}^*$. Hence w has the suffix $u_2 w_2$ and w' has the suffix $v_2 w_2$, which are suffixes of the same length. By assumption we have $t(u_2 w_2 z_n) \neq t(v_2 w_2 z_n)$ and hence also $\overleftarrow{t}(w z_n) \neq \overleftarrow{t}(w' z_n)$. This implies that $|\overleftarrow{t}(u_2\{u,v\}^n z_n)| \geq 2^n$ for all $n \in \mathbb{N}$. Since all words in $u_2\{u,v\}^n z_n \subseteq X$ have length $O(n)$ there exists a number $c > 1$ such that $|\overleftarrow{t}(X \cap \Sigma^{\leq cn})| \geq 2^n$ for sufficiently large n. ◀

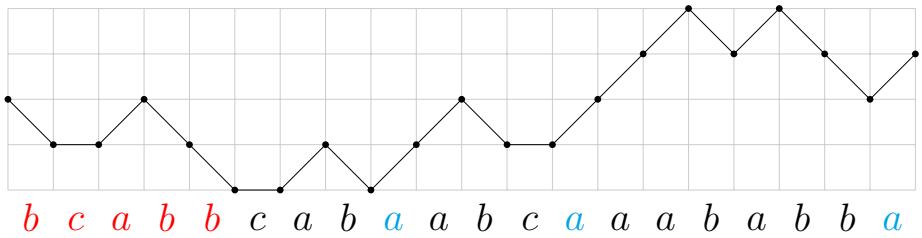


Figure 1 The stack height function for a word ($\Sigma_c = \{a\}$, $\Sigma_r = \{b\}$, $\Sigma_{int} = \{c\}$) and a monotonic factorization *bcabb cab a abc a aababb a*.

The following dichotomy theorem will be proved in Section 6.

▶ **Theorem 6.** *Let $t\colon \Sigma^* \to \Omega^*$ be rational and with suffix-closed domain $X = \mathrm{dom}(t)$. If X contains no linear fooling set for t and $t(X)$ is bounded then the $\overleftarrow{t}$-growth of X is polynomial. Otherwise the $\overleftarrow{t}$-growth of X is exponential.*

5 Reduction to transducer problem

Fix a VPA $A = (Q, \tilde{\Sigma}, \Gamma, \bot, q_0, \delta, F)$ and let $\emptyset \subsetneq L = \mathsf{L}(A) \subsetneq \Sigma^*$ for the rest of this section.

Monotonic factorization. A factorization of $w = w_0 w_1 \cdots w_m \in \Sigma^*$ into factors $w_i \in \Sigma^*$ is *monotonic* if w_0 is descending (possibly empty) and for each $1 \leq i \leq m$ the factor w_i is either a call letter $w_i \in \Sigma_c$ or a non-empty well-matched factor. If $w_0 w_1 \cdots w_m$ is a monotonic factorization then $w_i' w_{i+1} \cdots w_j$ is a monotonic factorization for any $0 \leq i \leq j \leq m$ and suffix w_i' of w_i. To see that every word $w \in \Sigma^*$ has at least one monotonic factorization consider the set of non-empty maximal well-matched factors in w (maximal with respect to inclusion). Observe that two distinct maximal well-matched factors in a word cannot overlap because the union of two overlapping well-matched factors is again well-matched. Since every internal letter is well-matched the remaining positions contain only return and call letters. Furthermore, every remaining call letter must be to the right of every remaining return letter, which yields a monotonic factorization of w. Figure 1 shows a monotonic factorization $w = w_0 w_1 \cdots w_m$ where the descending prefix w_0 is colored red and call letters w_i are colored green. The *stack height function* for the word w increases (decreases) by one on call (return) letters and stays constant on internal letters.

Representation of Myhill-Nerode classes. To apply Theorem 2 we need a suitable description of the $\sim_L$-classes. We follow the approach in [5] of choosing length-lexicographic minimal representative configurations. Since their definition slightly differs from ours (according to their definition, a VPA may not read a return letter if the stack contains $\bot$ only) we briefly recall their argument (in the appendix). Let $\mathrm{rConf} = \{\delta(\bot q_0, w) \mid w \in \Sigma^*\}$ be the set of all *reachable* configurations in A, which is known to be regular [7, 11]. Two configurations $c_1, c_2 \in \mathrm{rConf}$ are *equivalent*, denoted by $c_1 \sim c_2$, if $\mathsf{L}(c_1) = \mathsf{L}(c_2)$. By fixing arbitrary linear orders on Γ and Q we can consider the length-lexicographical order on rConf and define the function $\mathsf{rep}\colon \mathrm{rConf} \to \mathrm{rConf}$ which chooses the minimal representative from each $\sim$-class, i.e. for all $c \in \mathrm{rConf}$ we have $\mathsf{rep}(c) \sim c$ and for any $c' \in \mathrm{rConf}$ with $c \sim c'$ we have $\mathsf{rep}(c) \leq_{llex} c'$. The set of representative configurations is denoted by $\mathrm{Rep} = \mathsf{rep}(\mathrm{rConf})$.

▶ **Lemma 7** ([5])**.** *The function* rep *is rational.*

Finally we define $\nu_A \colon \Sigma^* \to \mathrm{Rep}$ by $\nu_A(w) = \mathsf{rep}(\delta(\bot q_0, w))$ for all $w \in \Sigma^*$. It represents $\sim_L$ in the sense that $\mathsf{L}(\nu_A(w)) = w^{-1}\mathsf{L}(A)$ for all $w \in \Sigma^*$ and hence $\nu_A(u) = \nu_A(v)$ if and only if $u \sim_L v$. Therefore we have $V_L(n) = \log|\tilde{\nu}_A(\Sigma^{\leq n})|$ by Theorem 2.

Flattenings. Since we cannot compute ν_A using a finite state transducer we choose a different representation of the input. Define the alphabet $\Sigma_f = \Sigma_c \cup Q \cup Q^Q$. A *flattening* is a word $s_0 s_1 \cdots s_m \in \Sigma_f^*$ where $s_0 \in Q^*$ and $s_i \in \Sigma_c \cup Q^Q Q^*$ for all $1 \leq i \leq m$. Notice that the factorization $s = s_0 s_1 \cdots s_m$ is unique. The set of all flattenings is $\mathrm{AllFlat} = Q^*(\Sigma_c \cup Q^Q Q^*)^*$. We define a function $t_f \colon \mathrm{AllFlat} \to \mathrm{rConf}$ as follows. Let $s = s_0 s_1 \cdots s_m \in \Sigma_f^*$ be a flattening and we define $t_f(s)$ by induction on m:

- If $s_0 = \varepsilon$ then $t_f(s_0) = \bot q_0$. If $s_0 = q_1 \cdots q_n \in Q^+$ then $t_f(s_0) = \bot q_1$.
- If $s_m \in \Sigma_c$ then $t_f(s_0 \cdots s_m) = \delta(t_f(s_0 \cdots s_{m-1}), s_m)$.
- If $s_m = \tau q_2 \cdots q_m \in Q^Q Q^*$ and $t_f(s_0 \cdots s_{m-1}) = \alpha q$ then $t_f(s) = \alpha\tau(q)$.

Define the function $\nu_f \colon \mathrm{AllFlat} \to \mathrm{Rep}$ by $\nu_f = \mathsf{rep} \circ t_f$.

▶ **Lemma 8.** *The functions t_f and ν_f are rational.*

Proof. We first define a transducer A_1 which handles flattenings where the initial factor is empty. Let $A_1 = (Q, \Sigma_f, Q \cup \Gamma, \{q_0\}, \Delta', Q, o)$ with the following transitions:

- $p \xrightarrow{q|\varepsilon} p$ for all $p, q \in Q$
- $p \xrightarrow{a|\gamma} q$ for all $\delta(p, a) = (\gamma, q)$ where $a \in \Sigma_c$
- $p \xrightarrow{\tau|\varepsilon} \tau(p)$ for all $p \in Q, \tau \in Q^Q$

and $o(q) = q$. For each $p \in Q$ let t_p be the rational function defined by A_1 with the only initial state p. One can easily show that for all $s \in \mathrm{AllFlat}$ we have $t_f(s) = \bot t_{q_0}(s)$ and $t_f(q_1 \cdots q_k s) = \bot t_{q_1}(s)$ for all $q_1 \cdots q_k \in Q^+$. Hence we can prove that t_f is rational by providing a transducer for t_f: First it verifies whether the input word belongs to the regular language $\mathrm{AllFlat} \subseteq \Sigma_f^*$. Simultaneously, it verifies whether the input word starts with a state $q \in Q$. If so, it memorizes q and simulates A_1 on s' from q, and otherwise A_1 is directly simulated on s from q_0. Since rep is rational by Lemma 7, ν_f is also rational. ◀

If $w = a_1 \cdots a_n \in D$ is a descending word then $\delta(\bot q_0, w) = \bot p$ for some $p \in Q$. By definition of ν_A there exists a state $q \in Q$ with $\nu_A(w) = \bot q$. Since each suffix of w is also descending we have $\tilde{\nu}_A(w) = \bot q_1 \bot q_2 \cdots \bot q_n$ for some $q_1, \ldots, q_n \in Q$. We define $\sigma_0(w) = q_1 \cdots q_n \in Q^*$, i.e. we remove the redundant $\bot$-symbols from $\tilde{\nu}_A(w)$. If w is non-empty and well-matched we additionally define $\sigma_1(w) = \tau q_2 \cdots q_n \in Q^Q Q^*$ where $\tau = \varphi(w)$. We define the sets $S_0 = \sigma_0(D)$ and $S_1 = \sigma_1(W \setminus \{\varepsilon\})$. Notice that S_0 is exactly the set of proper suffixes of words from S_1 since descending words are exactly the (proper) suffixes of well-matched words. We say that $s = s_0 s_1 \cdots s_m \in \mathrm{AllFlat}$ *represents* a word $w \in \Sigma^*$ if there exists a monotonic factorization $w = w_0 w_1 \cdots w_m \in \Sigma^*$ such that $s_0 = \sigma_0(w_0)$, and for all $1 \leq i \leq m$ if w_i is well-matched, then $s_i = \sigma_1(w_i)$, and if $w_i \in \Sigma_c$ then $s_i = w_i$. Since a word may have different monotonic factorizations, it may also be represented by many flattenings. We define the suffix-closed set $\mathrm{Flat} = S_0(\Sigma_c \cup S_1)^*$, containing all flattenings which represent some word.

▶ **Proposition 9.** *If $s \in \mathrm{AllFlat}$ represents $w \in \Sigma^*$ then $\nu_f(s) = \nu_A(w)$. Therefore, $\nu_f(\mathrm{Flat}) = \mathrm{Rep}$ and $V_L(n) = \log|\tilde{\nu}_f(\mathrm{Flat} \cap \Sigma_f^{\leq n})|$.*

▶ **Lemma 10.** *The languages S_0 and S_1 are context-free.*

Proof. Since S_0 is the set of all proper suffixes of words from S_1 it suffices to consider S_1. We will prove that $\{w \otimes \sigma_1(w) \mid w \in W\}$ is a VPL over the pushdown alphabet $(\Sigma_c \times \Sigma_f, \Sigma_r \times \Sigma_f, \Sigma_{int} \times \Sigma_f)$. Since the class of context-free languages is closed under

projections it then follows that S_1 is context-free. A VPA can test whether the first component $w = a_1 \cdots a_n$ is well-matched and whether the second component has the form $\tau q_2 \cdots q_n \in Q^Q Q^*$. Since VPLs are closed under Boolean operations, it suffices to test whether $\tau \neq \varphi(w)$ or there exists a state q_i with $\nu_A(a_i \cdots a_n) \neq \bot q_i$. To guess an incorrect state we use a VPA whose stack alphabet contains all stack symbols of A and a special symbol # representing the stack bottom. We guess and read a prefix of the input word and push/pop only the special symbol # on/from the stack. Then at some point we store the second component q_i in the next symbol and simulate A on the remaining suffix. Finally, we accept if and only if the reached state is q and $\mathsf{rep}(\bot q) \neq \bot q_i$. Similarly, we can verify τ by testing whether there exists a state $p \in Q$ with $\varphi(w)(p) \neq \tau(p)$. ◀

▶ **Lemma 11.** *The language S_0 is bounded if and only if S_1 is bounded. If S_0 is not bounded then the $\check{\nu}_A$-growth of Σ^* is exponential and therefore $V_L(n) \notin o(n)$.*

Proof. Assume that $S_0 \subseteq s_1^* \cdots s_k^*$ is bounded. Since $S_1 \subseteq \bigcup\{\tau S_0 \mid \tau \in Q^Q\}$ we have $S_1 \subseteq \tau_1^* \cdots \tau_m^* s_1^* \cdots s_k^*$ for any enumeration $\tau_1, \ldots, \tau_m$ of Q^Q. Conversely, if S_1 is bounded then each word in S_0 is a factor, namely a proper suffix, of a word from S_1. Therefore S_0 must be also bounded.

If the context-free language $S_0 = \sigma_0(D) \subseteq Q^*$ is not bounded then its growth must be exponential. Recall that $\check{\nu}_A(w)$ and $\sigma_0(w)$ are equal for all $w \in D$ up to the $\bot$-symbol. Hence $|\check{\nu}_A(\Sigma^{\leq n})| \geq |\check{\nu}_A(D \cap \Sigma^{\leq n})| = |\sigma_0(D \cap \Sigma^{\leq n})| = |S_0 \cap Q^{\leq n}|$, which proves the growth bound. ◀

Bounded overapproximation. By Lemma 11 we can restrict ourselves to the case that S_0 and S_1 are bounded languages, which will be assumed in the following. We define $\Psi(a_1 \cdots a_n) = \{(a_1, n), (a_2, n-1) \ldots, (a_n, 1)\}$ and $\Psi(L) = \bigcup_{w \in L} \Psi(w)$.

▶ **Lemma 12.** *Let K be a bounded context-free language. Then there exists a bounded regular superset $R \supseteq K$ such that $\{|w| \mid w \in K\} = \{|w| \mid w \in R\}$ and $\Psi(K) = \Psi(R)$, called a* bounded overapproximation *of K.*

Proof. We use Parikh's theorem [26], which implies that for every context-free language $K \subseteq \Sigma^*$ the set $\{|w| \mid w \in K\}$ is *semilinear*, i.e. a finite union of arithmetic progressions, and hence $\{v \in \Sigma^* \mid \exists w \in K : |v| = |w|\}$ is a regular language. Assume that $K \subseteq w_1^* \cdots w_k^*$ for some $w_1, \ldots, w_k \in \Sigma^*$. We define

$$R = (w_1^* \cdots w_k^*) \cap \{v \in \Sigma^* \mid \exists w \in K : |v| = |w|\} \cap \{w \in \Sigma^* \mid \Psi(w) \subseteq \Psi(K)\}.$$

Clearly, K is contained in R and it remains to verify that the third part is regular. It suffices to show that for each $a \in \Sigma$ the set $P_a = \{i \mid (a, i) \in \Psi(K)\}$ is semilinear because then an automaton can verify the property $\Psi(w) \subseteq \Psi(K)$. Consider the transducer

$$T_a = \{(a_1 \cdots a_n, \square^{n-i+1}) \mid a_1 \cdots a_n \in \Sigma^*, a_i = a\}.$$

It is easy to see that T_a is rational and $T_a K = \{\square^i \mid i \in P_a\}$. The claim follows again from Parikh's theorem. ◀

For each $\tau \in Q^Q$ let R_τ be a bounded overapproximation of $\tau^{-1} S_1$ and let $R_1 = \bigcup_{\tau \in Q^Q} (\tau R_\tau)$. Let $R_0 = \bigcup_{\tau \in Q^Q} \mathsf{Suf}(R_\tau)$, which is the set of all proper suffixes of words in R_1. Both R_0 and R_1 are also bounded languages. Finally, set $\mathrm{RegFlat} = R_0(\Sigma_c \cup R_1)^*$, which is the same as $\mathsf{Suf}((\Sigma_c \cup R_1)^*)$ and is suffix-closed. According to the definition of bounded overapproximations we can approximate a word $v = \tau q_2 \cdots q_k \in R_1$ in two

possible ways: Firstly, define $\mathsf{apx}_\ell(v)$ to be any word of the form $\mathsf{apx}_\ell(v) = \tau p_2 \cdots p_k \in S_1$ with $|v| = |\mathsf{apx}_\ell(v)|$. Secondly, for any position $2 \leq i \leq k$ define $\mathsf{apx}_i(v)$ to be any word $\mathsf{apx}_i(v) = \tau s' q_i p_{i+1} \cdots p_k \in S_1$ where $s', p_{i+1} \cdots p_k \in Q^*$. If $r = r_0 r_1 \cdots r_m \in \mathrm{RegFlat}$ then we can replace any internal factor $r_i \in R_1$ by $\mathsf{apx}_\ell(r_i)$ or any $\mathsf{apx}_j(r_i)$ without changing the value of $\nu_f(r)$.

▶ **Proposition 13.** $\nu_f(\mathrm{Flat}) = \nu_f(\mathrm{RegFlat}) = \mathrm{Rep}$.

▶ **Proposition 14.** *If* RegFlat *contains a linear fooling set for* ν_f *then also* Flat *contains a linear fooling set for* ν_f.

Proof of Theorem 1. If $L = \emptyset$ or $L = \Sigma^*$ then $V_L(n) \in O(1)$. Now assume $\emptyset \subsetneq L \subsetneq \Sigma^*$, in which case we have $V_L(n) = \Omega(\log n)$. Furthermore we know that $V_L(n) = \log |\check{\nu}_f(\mathrm{Flat} \cap \Sigma_f^{\leq n})|$ by Proposition 9. If the constructed language S_0 is not bounded then $V_L(n) \notin o(n)$ by Lemma 11. Now assume that S_0 is bounded, in which case we can construct the regular language RegFlat. By Theorem 6 the $\check{\nu}_f$-growth of RegFlat is either polynomial or exponential (formally, we have to restrict the domain of ν_f to the regular language RegFlat). If the $\check{\nu}_f$-growth of RegFlat is polynomial then the same holds for its subset Flat, and hence $V_L(n) \in O(\log n)$. If the $\check{\nu}_f$-growth of RegFlat is exponential then by Theorem 6 either the image $\nu_f(\mathrm{RegFlat})$ is not bounded or RegFlat contains a linear fooling set for ν_f. By Proposition 13 we have $\nu_f(\mathrm{RegFlat}) = \nu_f(\mathrm{Flat}) = \mathrm{Rep}$. Hence, if Rep has exponential growth then Proposition 3 implies that Flat has exponential $\check{\nu}_f$-growth and hence $V_L(n) \notin o(n)$. If RegFlat contains a linear fooling set for ν_f then also Flat contains one by Proposition 14. By Proposition 5 the $\check{\nu}_f$-growth of Flat is exponential and hence $V_L(n) \notin o(n)$. ◀

6 Dichotomy for rational functions

In this section we will prove Theorem 6. Let $t \colon \Sigma^* \to \Omega^*$ be a rational function with suffix-closed domain $X = \mathrm{dom}(t)$. By Proposition 3 the interesting case is where the image $t(X)$ is polynomial growing, i.e. a bounded language. There are two further necessary properties in order to achieve polynomial $\check{t}$-growth. Since we apply the rational function to all suffixes, it is natural to consider right transducers, reading the input from right to left. The first property states that t has to resemble so called right-subsequential functions, which are defined by deterministic finite right transducers. Here we will make use of a representation of rational functions due to Reutenauer and Schützenberger, which decomposes the rational function t into a right congruence $\mathcal{R}_t$ and a right-subsequential transducer B [27]. Secondly, we demand that B is well-behaved, which means that, roughly speaking, the output produced during a run inside a strongly connected component only depends on its entry state and the length of the run. We will prove that in fact these properties are sufficient for the polynomial $\check{t}$-growth and in all other cases X contains a linear fooling set.

The case of finite-index right congruences. Let $\sim$ be a finite index right congruence on Σ^* and $\approx$ its suffix expansion. We will characterize those finite index right congruences $\sim$ where $\Sigma^{\leq n}/{\approx}$ is polynomially bounded, which can be viewed as a special case of Theorem 6 since $\nu_\sim \colon \Sigma^* \to \Sigma^*/{\sim}$ is rational. First assume that $\sim$ is the Myhill-Nerode right congruence $\sim_L$ of a regular language L. Since $\log |\Sigma^{\leq n}/{\approx}|$ is exactly the space complexity $V_L(n)$ by Theorem 2, this case was characterized in [19] using so called critical tuples in the minimal DFA for L. We slightly adapt this definition for right congruences. A *critical tuple* in a right congruence $\sim$ is a tuple of words $(u_2, v_2, u, v) \in (\Sigma^*)^4$ such that $|u_2| = |v_2| \geq 1$, there exist $u_1, v_1 \in \Sigma^*$ with $u = u_1 u_2$, $v = v_1 v_2$, and $u_2 w \not\sim v_2 w$ for all $w \in \{u, v\}^*$.

▶ **Proposition 15.** *If $\sim$ has a critical tuple then $|\Sigma^{\leq n}/\approx|$ grows exponentially and there exists a critical tuple (u_2, v_2, u, v) in $\sim$ such that $u_2u \sim u_2wu$ and $v_2u \sim v_2wu$ for all $w \in \{u, v\}^*$.*

Proof. If (u_2, v_2, u, v) is critical tuple in a right congruence $\sim$ then we claim that $|\Sigma^{\leq n}/\approx|$ grows exponentially. Let $n \in \mathbb{N}$ and let $w \neq w' \in \{u, v\}^n$. There exists a word $z \in \{u, v\}^*$ such that w and w' have the suffixes u_2z and v_2z of equal length. By the definition of critical tuples we have $u_2z \not\sim v_2z$, which implies $w \not\approx w'$. Therefore $|\Sigma^{\leq cn}/\approx| \geq 2^n$ where $c = \max\{|u|, |v|\}$.

The second part is based on the proof of [20, Lemma 7.4]. Let $\equiv$ be the syntactic congruence on Σ^* defined by $x \equiv y$ if and only if $\ell x \sim \ell y$ for all $\ell \in \Sigma^*$. Since $\sim$ is a right congruence $\equiv$ is a congruence on Σ^* of finite index satisfying $\equiv \subseteq \sim$. Define the monoid $M = \Sigma^*/\equiv$. It is known that there exists a number $\omega \in \mathbb{N}$ such that m^ω is idempotent for all $m \in M$, i.e. $m^\omega \cdot m^\omega = m^\omega$. Now let (u_2, v_2, u, v) be a critical tuple and define $u' = (v^\omega u^\omega)^\omega$ and $v' = (v^\omega u^\omega)^\omega v^\omega$. Since u_2 is a suffix of u', v_2 is a suffix of u' and $u', v' \in \{u, v\}^*$ the tuple (u_2, v_2, u', v') is again a critical tuple in $\sim$. Furthermore we have $u'u' = (v^\omega u^\omega)^\omega (v^\omega u^\omega)^\omega \equiv (v^\omega u^\omega)^\omega = u'$ and $v'u' = (v^\omega u^\omega)^\omega v^\omega (v^\omega u^\omega)^\omega \equiv (v^\omega u^\omega)^\omega = u'$, and therefore $u' \equiv wu'$ for all $w \in \{u', v'\}^*$. Since $\equiv$ is a congruence this implies $u_2u' \equiv u_2wu'$ and $v_2u' \equiv v_2wu'$ for all $w \in \{u', v'\}^*$, and thus also $u_2u' \sim u_2wu'$ and $v_2u' \sim v_2wu'$, which concludes the proof. ◀

▶ **Theorem 16.** *Let $L \subseteq \Sigma^*$ be regular. Then $V_L(n) \in O(\log n)$ if and only if $|\Sigma^{\leq n}/\approx_L|$ is polynomially bounded if and only if $\sim_L$ has no critical tuple.*

Proof. The first equivalence follows from Theorem 2. By Proposition 15 the existence of a critical tuple in $\sim$ implies exponential growth of $|\Sigma^{\leq n}/\approx|$.

Now assume that $V_L(n) \notin O(\log n)$. By [19, Lemma 7.2] there exist words $u_2, v_2, u, v \in \Sigma^*$ such that u_2 is a suffix of u, v_2 is a suffix of v, $|u_2| = |v_2|$ and $u_2w \not\sim_L v_2w'$ for all $w, w' \in \{u, v\}^*$ (one needs the fact that $x \sim_L y$ if and only if x and y reach the same state in the minimal DFA for L). Since in particular $u_2w \not\sim_L v_2w$ for all $w \in \{u, v\}^*$ the tuple (u_2, v_2, u, v) constitutes a critical tuple. ◀

We generalize this theorem to arbitrary finite index right congruences (Theorem 18). Given equivalence relations $\sim$ and $\sim'$ on a set X, we say that $\sim'$ is *coarser* than $\sim$ if $\sim \subseteq \sim'$, i.e. each $\sim'$-class is a union of $\sim$-classes. The *intersection* $\sim \cap \sim'$ is again an equivalence relation on X.

▶ **Lemma 17.** *Let $\sim$ and $\sim'$ be right congruences.*

(a) *If $\sim'$ is coarser than $\sim$ and $\sim$ has no critical tuple, then $\sim'$ also has no critical tuple.*

(b) *If $\sim$ and $\sim'$ have no critical tuple then $\sim \cap \sim'$ is also a right congruence which has no critical tuple*

Proof. Closure under coarsening is clear because the property "$\sim$ has no critical tuple" is *positive* in $\sim$: $\forall u = u_1u_2 \, \forall v = v_1v_2 (|u_2| = |v_2| \to \exists w \in \{u, v\}^* : u_2w \sim v_2w)$.

Consider two right congruences $\sim$, $\sim'$ which have no critical tuples. One can verify that their intersection $\sim \cap \sim'$ is again a right congruence. Let $u = u_1u_2$ and $v = v_1v_2$ with $|u_2| = |v_2|$. Because $\sim$ has no critical tuple there exist a word $w \in \{u, v\}^*$ with $u_2w \sim v_2w$. Now consider the condition for the words $u_1(u_2w)$ and $v_1(v_2w)$. Because $\sim'$ has no critical tuple there exists a word $x \in \{uw, vw\}^*$ such that $u_2wx \sim' v_2wx$. Since $\sim$ is a right congruence we also have $u_2wx \sim v_2wx$ and thus $u_2wx \, (\sim \cap \sim') \, v_2wx$. This proves that $\sim \cap \sim'$ has no critical tuple. ◀

▶ **Theorem 18.** *$|\Sigma^{\leq n}/\approx|$ is polynomially bounded if and only if $\sim$ has no critical tuple.*

Proof. Let $u_1, \ldots, u_m$ be representatives from each $\sim$-class. Observe that $\sim = \bigcap_{i=1}^m \sim_{[u_i]_\sim}$ because $\sim$ saturates each class $[u_i]_\sim$ and $\bigcap_{i=1}^m \sim_{[u_i]_\sim}$ also saturates each class $[v]_\sim$. Let us write $\sim_i$ instead of $\sim_{[u_i]_\sim}$ and let $\approx_i$ be its suffix expansion $\approx_{[u_i]_\sim}$. Then we have $\sim = \bigcap_{i=1}^m \sim_i$ and $\approx = \bigcap_{i=1}^m \approx_i$. This implies that

$$\max_{1 \le i \le m} |\Sigma^{\le n}/{\approx_i}| \le |\Sigma^{\le n}/{\approx}| \le \prod_{i=1}^m |\Sigma^{\le n}/{\approx_i}|. \tag{1}$$

($\Rightarrow$): If $|\Sigma^{\le n}/{\approx}|$ is polynomially bounded then the same holds for $|\Sigma^{\le n}/{\approx_i}|$ for all $1 \le i \le k$ by (1). By Theorem 16 $\sim_{[u_i]_\sim}$ has no critical tuple for all $1 \le i \le k$ and therefore Lemma 17(b) implies that $\sim = \bigcap_{i=1}^m \sim_{[u_i]_\sim}$ has no critical tuple.

($\Leftarrow$): If $\sim$ has no critical tuple then each congruence $\sim_i$ has no critical tuple by Lemma 17(a) because $\sim_i$ is coarser than $\sim$. Theorem 16 implies that $|\Sigma^{\le n}/{\approx_i}|$ is polynomially bounded for all $1 \le i \le k$. By (1) also $|\Sigma^{\le n}/{\approx}|$ is polynomially bounded. ◀

Regular look-ahead. A result due to Reutenauer and Schützenberger states that every rational function f can be factorized as $f = r \circ \ell$ where ℓ and r are *left- and right-subsequential*, respectively [27]. A rational function is left- or right-subsequential if the input is read in a deterministic fashion from left to right and right to left, respectively. In the literature the order of the directions is usually reversed, i.e. one decomposes t as $f = r \circ \ell$. Often this is described by the statement that every rational function is (left-)subsequential with regular look-ahead. Furthermore, this decomposition is canonical in a certain sense.

We follow the notation from the survey paper [16]. A *right-subsequential transducer* $B = (Q, \Sigma, \Omega, F, \Delta, \{q_{in}\}, o)$ is a real-time right transducer which is *deterministic*, i.e. q_{in} is the only initial state and for every $p \in Q$ and $a \in \Sigma$ there exists at most one transition $(p, a, y, q) \in \Delta$. Clearly, right-subsequential transducers define rational functions, the so called *right-subsequential functions*, but not every rational function is right-subsequential. Let $\mathcal{R}$ be a right congruence on Σ^* with finite index. The *look-ahead extension* is the injective function $e_\mathcal{R} \colon \Sigma^* \to (\Sigma \times \Sigma^*/\mathcal{R})^*$ defined by

$$e_\mathcal{R}(a_1 \cdots a_n) = (a_1, [\varepsilon]_\mathcal{R})(a_2, [a_1]_\mathcal{R})(a_3, [a_1 a_2]_\mathcal{R}) \cdots (a_n, [a_1 \cdots a_{n-1}]_\mathcal{R}).$$

Let $f \colon \Sigma^* \to \Omega^*$ be a partial function. The partial function $f[\mathcal{R}] \colon (\Sigma \times \Sigma^*/\mathcal{R})^* \to \Omega^*$ with $\mathrm{dom}(f[\mathcal{R}]) = e_\mathcal{R}(\mathrm{dom}(f))$ is defined by $f[\mathcal{R}](e_\mathcal{R}(x)) = f(x)$. Furthermore we define a right congruence $\mathcal{R}_f$ on Σ^*. For this we need the distance function $\|x, y\| = |x| + |y| - 2|x \wedge y|$ where $x \wedge y$ is the longest common suffix of x and y. Equivalently, $\|x, y\|$ is the length of the reduced word of xy^{-1} in the free group generated by Σ. Notice that $\|\cdot, \cdot\|$ satisfies the triangle inequality. We define $u \,\mathcal{R}_f\, v$ if and only if (i) $u \sim_{\mathrm{dom}(f)} v$ and (ii) $\{\|f(uw), f(vw)\| \mid uw, vw \in \mathrm{dom}(f)\}$ is finite. One can verify that $\mathcal{R}_f$ is a right congruence on Σ^*. As an example, recall the rational transduction f from Example 4. The induced right congruence $\mathcal{R}_f$ has two classes, which are a^* and $a^* b \{a, b\}^*$.

▶ **Theorem 19** ([27]). *A partial function $f \colon \Sigma^* \to \Omega^*$ is rational if and only if $\mathcal{R}_f$ has finite index and $f[\mathcal{R}_f]$ is right-subsequential.*

For the rest of the section let $B = (Q, \Sigma \times \Sigma^*/\mathcal{R}_t, \Omega, F, \Delta, \{q_{in}\}, o)$ be a trim right-subsequential transducer for $t[\mathcal{R}_t]$. One obtains an unambiguous real-time right transducer A for t by projection to the first component, i.e. $A = (Q, \Sigma, \Omega, F, \Lambda, \{q_{in}\}, o)$ where $\Lambda = \{(q, a, y, p) \mid (q, (a, b), y, p) \in \Delta\}$. Notice that every run $q \xleftarrow{x|y} p$ in A induces a corresponding

run $q \xleftarrow{(x,z)|y} p$ in B for some $z \in (\Sigma^*/\mathcal{R}_t)^*$ and that this correspondence is a bijection between the sets of all runs in A and B. We need two auxiliary lemmas which concern the right congruence $\mathcal{R}_t$.

▶ **Lemma 20** (Short distances). *Let $u, v, w \in \Sigma^*$ with $uw, vw \in X$. If $u \mathcal{R}_t v$ then $\|t(uw), t(vw)\| \leq O(|u| + |v|)$.*

Two partial functions $t_1, t_2 \colon \Sigma^* \to \Omega^*$ are *adjacent* if $\sup\{\|t_1(w), t_2(w)\| \mid w \in \mathrm{dom}(t_1) \cap \mathrm{dom}(t_2)\} < \infty$ where $\sup \emptyset = -\infty$. We remark that two functions are adjacent in our definition if and only if their reversals are adjacent according to the original definition [27]. Notice that $u \mathcal{R}_t v$ if and only if $u \sim_X v$ and the functions $w \mapsto t(uw)$ and $w \mapsto t(vw)$ are adjacent.

▶ **Lemma 21** (Short witnesses). *Let $t_1, t_2 \colon \Sigma^* \to \Omega^*$ be rational functions which are not adjacent. Then there are words $x, y, z \in \Sigma^*$ such that $xy^*z \subseteq \mathrm{dom}(t_1) \cap \mathrm{dom}(t_2)$ and $\|t_1(xy^kz), t_2(xy^kz)\| = \Omega(k)$. In particular, for each $k \in \mathbb{N}$ there exists a word $x \in \mathrm{dom}(t_1) \cap \mathrm{dom}(t_2)$ of length $|x| \leq O(k)$ such that $\|t_1(x), t_2(x)\| \geq k$.*

▶ **Proposition 22.** *If $\mathcal{R}_t$ has a critical tuple then X contains a linear fooling set.*

Proof. Let (u_2, v_2, u, v) be a critical tuple in $\mathcal{R}_t$ with $u = u_1u_2$ and $v = v_1v_2$. By Proposition 15 we can assume that $u_2u \mathcal{R}_t u_2wu$ and $v_2u \mathcal{R}_t v_2wu$ for all $w \in \{u, v\}^*$. By assumption we know that $(u_2u, v_2u) \notin \mathcal{R}_t$. Furthermore, we claim that $u_2u \sim_X v_2u$: Let $z \in \Sigma^*$ and assume that $u_2uz \in X$. Then $u_2v_1v_2uz \in X$ because $u_2u \sim_X u_2v_1v_2u$, and thus $v_2uz \in X$ because X is suffix-closed. The other direction follows by a symmetric argument.

Let $n \in \mathbb{N}$ and define

$$N = \max_{x \in \{u_2, v_2\}} \max_{w \in \{u,v\}^{\leq n}} \sup\{\|t(xuz), t(xwuz)\| \mid xuz, xwuz \in X\} < \infty.$$

By Lemma 20 we have $N \leq O(n)$. Since $(u_2u, v_2u) \notin \mathcal{R}_t$ and $u_2u \sim_X v_2u$, the functions $z \mapsto t(u_2uz)$ and $z \mapsto t(v_2uz)$ are not adjacent. By Lemma 21 there exists a word $z_n \in (u_2u)^{-1}X$ with $\|t(u_2uz_n), t(v_2uz_n)\| \geq 2N + 1$ and $|z_n| \leq O(N) \leq O(n)$. We claim that $t(u_2wuz_n) \neq t(v_2wuz_n)$ for all $w \in \{u, v\}^{\leq n}$: By the triangle inequality we have

$$\begin{aligned} 2N + 1 &\leq \|t(u_2uz_n), t(v_2uz_n)\| \\ &\leq \|t(u_2uz_n), t(u_2wuz_n)\| + \|t(u_2wuz_n), t(v_2wuz_n)\| + \|t(v_2wuz_n), t(v_2uz_n)\| \\ &\leq 2N + \|t(u_2wuz_n), t(v_2wuz_n)\| \end{aligned}$$

which implies $\|t(u_2wuz_n), t(v_2wuz_n)\| \geq 1$ and in particular $t(u_2wuz_n) \neq t(v_2wuz_n)$. We have proved that for each $n \in \mathbb{N}$ there exists a word z_n of length $O(n)$ such that $t(u_2wuz_n) \neq t(v_2wuz_n)$ for all $w \in \{u, v\}^{\leq n}$. If Z is the set of all constructed z_n for $n \in \mathbb{N}$ then $\{u_2, v_2\}\{u, v\}^*uZ \subseteq X$ and (u_2, v_2, u, v, uZ) is a linear fooling scheme. ◀

Well-behaved transducers. Let $(Q, \preceq)$ be the quasi-order defined by $q \preceq p$ iff there exists a run from p to q in A or equivalently in B. Its equivalence classes are the strongly connected components (SCCs) of A and B. A word $w \in \Sigma^*$ is *guarded* by a state $p \in Q$ if there exists a run $q' \xleftarrow{w} p$ in A such that $p \preceq q'$, i.e. p and q' belong to the same SCC. Notice that the set of all words which are guarded by a fixed state p is suffix-closed. A run $q \xleftarrow{w} p$ in A is *guarded* if w is guarded by p. We say that A is *well-behaved* if for all $p \in Q$ and all guarded accepting runs π, π' from p with $|\pi| = |\pi'|$ we have $\mathsf{out}_F(\pi) = \mathsf{out}_F(\pi')$.

▶ **Proposition 23.** *If A is not well-behaved then X contains a linear fooling set.*

Proof. Assume there exist states $p, q, r, q', r' \in Q$, and accepting runs $q \xleftarrow{u_2} p$ and $r \xleftarrow{v_2} p$ with $|u_2| = |v_2|$ and $\mathsf{out}_F(q \xleftarrow{u_2} p) \neq \mathsf{out}_F(r \xleftarrow{v_2} p)$. Furthermore let $p \xleftarrow{u_1} q' \xleftarrow{u_2} p$, $p \xleftarrow{v_1} r' \xleftarrow{v_2} p$ and $p \xleftarrow{s} q_{in}$ be runs. Let $u = u_1u_2$ and $v = v_1v_2$ and consider any word $w \in \{u, v\}^*$. Since $t(u_2ws) = \mathsf{out}_F(q \xleftarrow{u_2} p)\,\mathsf{out}(p \xleftarrow{ws} q_{in})$ and $t(v_2ws) = \mathsf{out}_F(r \xleftarrow{v_2} p)\,\mathsf{out}(p \xleftarrow{ws} q_{in})$, we have $t(u_2ws) \neq t(v_2ws)$. This shows that $(u_2, v_2, u, v, \{s\})$ is a linear fooling scheme. ◀

If π is a non-empty run $p \xleftarrow{a_1\cdots a_n} q$ in A and $p \xleftarrow{(a_1,\rho_1)\cdots(a_n,\rho_n)} q$ is the corresponding run in B then we call ρ_1 the *key* of π. The following lemma justifies the name, stating that π is determined by the state q, the word $a_1 \cdots a_n$ and the key ρ_1.

▶ **Lemma 24.** *If $p \xleftarrow{w} q$ and $p' \xleftarrow{w} q$ are non-empty runs in A with the same key then the runs must be identical.*

Proof. Assume that $w = a_1 \cdots a_n$ and let $p \xleftarrow{(a_1,\rho_1)\cdots(a_n,\rho_n)} q$ and $p' \xleftarrow{(a_1,\rho'_1)\cdots(a_n,\rho'_n)} q$ be the corresponding runs in B with $\rho_1 = \rho'_1$. We proceed by induction on n. If $n = 1$ then this statement is trivial because B is deterministic. Now assume $n \geq 2$ and let $p \xleftarrow{a_1} r \xleftarrow{a_2\cdots a_n} q$ and $p' \xleftarrow{a_1} r' \xleftarrow{a_2\cdots a_n} q$. Since B is trim there exist an accepting run on $e_{\mathcal{R}_t}(u)$ from p and an accepting run on $e_{\mathcal{R}_t}(u')$ from p' for some words $u, u' \in \Sigma^*$. By definition of $t[\mathcal{R}_t]$ we have $[u]_{\mathcal{R}_t} = \rho_1 = \rho'_1 = [u']_{\mathcal{R}_t}$ and therefore $\rho_2 = [ua_1]_{\mathcal{R}_t} = [u'a_1]_{\mathcal{R}_t} = \rho'_2$. By induction hypothesis we know that the runs $r \xleftarrow{a_2\cdots a_n} q$ and $r' \xleftarrow{a_2\cdots a_n} q$ are identical. Since $p \xleftarrow{(a_1,\rho_1)} r$ and $p' \xleftarrow{(a_1,\rho'_1)} r'$ and B is deterministic we must have $p = p'$. ◀

Let π be any run on a word $y \in \Sigma^*$. If π is not guarded, we can factorize $\pi = \pi'\pi''$ such that π'' is the shortest suffix of π which is unguarded, and then iterate this process on π'. This yields unique factorizations $\pi = \pi_0\pi_1 \cdots \pi_m$ and $y = y_0y_1 \cdots y_m$ where π_i is a run on y_i from a state q_i to a state q_{i-1} such that y_i is the shortest suffix of $y_0 \cdots y_i$ which is not guarded by q_i for all $1 \leq i \leq m$ and π_0 is guarded. The factorization $\pi = \pi_0\pi_1 \cdots \pi_m$ is the *guarded factorization* of π.

▶ **Proposition 25.** *Assume that $t(X)$ is bounded, A is well-behaved and $\mathcal{R}_t$ has no critical tuple. Then the $\overleftarrow{t}$-growth of X is polynomially bounded.*

Proof. We will describe an encoding of $\overleftarrow{t}(w)$ for $w \in X$ using $O(\log |w|)$ bits. For each word $w \in \Sigma^*$ and each state $q \in Q$ we define a tree $T_{q,w}$ recursively, which carries information at the nodes and edges. If w is guarded by q then $T_{q,w}$ consists of a single node labelled by the pair $(q, |w|)$. Otherwise let $w = uv$ such that v is the shortest suffix of w which is not guarded by q. Then $T_{q,w}$ has a root which is labelled by the tuple $(q, |w|, |v|, \overleftarrow{\nu}_{\mathcal{R}_t}(u))$. For each run $p \xleftarrow{v} q$ we attach $T_{p,u}$ to the root as a direct subtree. The edge is labelled by the pair $(\rho, \mathsf{out}(p \xleftarrow{v} q))$ where ρ is the key of $p \xleftarrow{v} q$. By Lemma 24 distinct outgoing edges from the root are labelled by distinct keys.

The tree $T_{q,w}$ can be encoded using $O(\log |w|)$ bits: Since we have $p \prec q$ for every unguarded run $p \xleftarrow{v} q$ the tree $T_{q,w}$ has height at most $|Q|$ and size at most $|Q|^{|Q|}$. All occurring numbers have at most magnitude $|w|$, and the states and keys can be encoded by $O(1)$ bits. The output words $\mathsf{out}(p \xleftarrow{v} q)$ are factors of words from the bounded language $t(X)$ and have length at most $\mathsf{iml}(A) \cdot |v|$. Thus they can be encoded using $O(\log |w|)$ bits. The node label $\overleftarrow{\nu}_{\mathcal{R}_t}(u)$ can be encoded using $O(\log |w|)$ bits by Theorem 18 since $\mathcal{R}_t$ has no critical tuple.

Let $w = xy \in \Sigma^*$, $q \in Q$ and let π be an accepting run on y from q. We show that $T_{q,w}$ and $|y|$ determine $\mathsf{out}_F(\pi)$ by induction on the length of the guarded factorization $\pi = \pi_0\pi_1 \cdots \pi_m$. Since $T_{q_{in},w}$ determines the length $|w|$, the tuple $\overleftarrow{t}(w)$ is determined by $T_{q_{in},w}$ for all $w \in X$.

If $m = 0$ then y is guarded by q. Since A is well-behaved $\mathsf{out}_F(\pi)$ is determined by q (which is part of the label of the root of $T_{q,w}$) and $|y|$ only. Now assume $m \geq 1$ and suppose that π_i is a run $q_{i-1} \xleftarrow{y_i} q_i$ for all $1 \leq i \leq m$ with $q_m = q$. Then y_m is the shortest suffix of w which is not guarded by q. The root of $T_{q,w}$ is labelled by $(q, |y_m|, \overleftarrow{\nu}_{\mathcal{R}_t}(xy_0 \cdots y_{m-1}))$. Since $|y_m|$ and $|y|$ are known, we can also determine $|y_0 \cdots y_{m-1}|$. From $\overleftarrow{\nu}_{\mathcal{R}_t}(xy_0 \cdots y_{m-1})$ and $|y_0 \cdots y_{m-1}|$ we can then determine $[y_0 \cdots y_{m-1}]_{\mathcal{R}_t}$, which is the key of π_m. By Lemma 24 we can find the unique edge which is labelled by $([y_0 \cdots y_{m-1}]_{\mathcal{R}_t}, \mathsf{out}(\pi_m))$. It leads to the direct subtree $T_{q_{m-1}, xy_0 \cdots y_{m-1}}$ of $T_{q,w}$. By induction hypothesis $T_{q_{m-1}, xy_0 \cdots y_{m-1}}$ and $|y_0 \cdots y_{m-1}|$ determine $\mathsf{out}_F(\pi_0 \cdots \pi_{m-1})$. Finally, we can determine $\mathsf{out}_F(\pi_0 \cdots \pi_m) = \mathsf{out}_F(\pi_0 \cdots \pi_{m-1})\,\mathsf{out}(\pi_m)$, concluding the proof. ◀

Now we can prove Theorem 6: If X contains no linear fooling set for t then A must be well-behaved by Proposition 23 and $\mathcal{R}_t$ has no critical tuple by Proposition 22. If additionally $t(X)$ is bounded then the $\overleftarrow{t}$-growth of X is polynomially bounded by Proposition 25. Otherwise, if either X contains a linear fooling set or $t(X)$ is not bounded then the $\overleftarrow{t}$-growth of X is exponential by Proposition 5 and by Proposition 3.

References

1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211, 2004. doi:10.1145/1007352.1007390.

2 Arvind Arasu and Gurmeet Singh Manku. Approximate Counts and Quantiles over Sliding Windows. In *Proceedings of PODS 2004*, pages 286–296. ACM, 2004.

3 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of PODS 2003*, pages 234–243. ACM, 2003.

4 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theor. Comput. Sci.*, 494:13–23, 2013. doi:10.1016/j.tcs.2012.12.028.

5 Vince Bárány, Christof Löding, and Olivier Serre. Regularity Problems for Visibly Pushdown Languages. In *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, pages 420–431, 2006. doi:10.1007/11672142_34.

6 Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher: Informatik*. Teubner, 1979.

7 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, pages 135–150, 1997. doi:10.1007/3-540-63141-0_10.

8 Vladimir Braverman and Rafail Ostrovsky. Smooth Histograms for Sliding Windows. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science FOCS 2007*, pages 283–293. IEEE Computer Society, 2007.

9 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.

10 Dany Breslauer and Zvi Galil. Real-Time Streaming String-Matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014.

11 J Richard Büchi. Regular canonical systems. *Archiv für mathematische Logik und Grundlagenforschung*, 6(3-4):91–111, 1964.

12 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary Matching in a Stream. In *Proceedings of ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015.

13 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of SODA 2016*, pages 2039–2052. SIAM, 2016.

14 Raphaël Clifford and Tatiana A. Starikovskaya. Approximate Hamming Distance in a Stream. In *Proceedings of ICALP 2016*, volume 55 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

15 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

16 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *SIGLOG News*, 3(3):4–19, 2016. doi:10.1145/2984450.2984453.

17 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming Property Testing of Visibly Pushdown Languages. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

18 Moses Ganardi. Visibly Pushdown Languages over Sliding Windows. Technical report, arXiv.org, 2018. arXiv:1812.11549.

19 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata Theory on Sliding Windows. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

20 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. Technical report, arXiv.org, 2018. arXiv:1702.04376.

21 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying Regular Languages over Sliding Windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

22 Moses Ganardi, Artur Jez, and Markus Lohrey. Sliding Windows over Context-Free Languages. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 15:1–15:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.15.

23 Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., New York, NY, USA, 1966.

24 Seymour Ginsburg and Edwin H Spanier. Bounded ALGOL-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

25 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing Well-Parenthesized Expressions in the Streaming Model. *SIAM J. Comput.*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

26 Rohit Parikh. On Context-Free Languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

27 Christophe Reutenauer and Marcel Paul Schützenberger. Minimization of Rational Word Functions. *SIAM J. Comput.*, 20(4):669–685, 1991. doi:10.1137/0220042.

28 Andreas Weber and Reinhard Klemm. Economy of Description for Single-Valued Transducers. *Inf. Comput.*, 118(2):327–340, 1995. doi:10.1006/inco.1995.1071.

Fast and Longest Rollercoasters

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Florin Manea
Kiel University, Germany
flm@informatik.uni-kiel.de

Radosław Serafin
Institute of Computer Science, University of Wrocław, Poland
radserafin@gmail.com

Abstract

For $k \geq 3$, a *k-rollercoaster* is a sequence of numbers whose every maximal contiguous subsequence, that is increasing or decreasing, has length at least k; 3-rollercoasters are called simply rollercoasters. Given a sequence of distinct real numbers, we are interested in computing its maximum-length (not necessarily contiguous) subsequence that is a k-rollercoaster. Biedl et al. (2018) have shown that each sequence of n distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$ for $n > 7$, and that a longest rollercoaster contained in such a sequence can be computed in $O(n \log n)$-time (or faster, in $O(n \log \log n)$ time, when the input sequence is a permutation of $\{1, \ldots, n\}$). They have also shown that every sequence of $n \geqslant (k-1)^2 + 1$ distinct real numbers contains a k-rollercoaster of length at least $\frac{n}{2(k-1)} - \frac{3k}{2}$, and gave an $O(nk \log n)$-time (respectively, $O(nk \log \log n)$-time) algorithm computing a longest k-rollercoaster in a sequence of length n (respectively, a permutation of $\{1, \ldots, n\}$).

In this paper, we give an $O(nk^2)$-time algorithm computing the length of a longest k-rollercoaster contained in a sequence of n distinct real numbers; hence, for constant k, our algorithm computes the length of a longest k-rollercoaster in optimal linear time. The algorithm can be easily adapted to output the respective k-rollercoaster. In particular, this improves the results of Biedl et al. (2018), by showing that a longest rollercoaster can be computed in optimal linear time. We also present an algorithm computing the length of a longest k-rollercoaster in $O(n \log^2 n)$-time, that is, subquadratic even for large values of $k \leq n$. Again, the rollercoaster can be easily retrieved. Finally, we show an $\Omega(n \log k)$ lower bound for the number of comparisons in any comparison-based algorithm computing the length of a longest k-rollercoaster.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases sequences, alternating runs, patterns in permutations

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.30

1 Introduction

The mathematical study of patterns occurring in sequences of numbers is a rather old and well developed topic in combinatorics and algorithms on sequences. Within this topic, of a particularly high interest is the study of long increasing and decreasing (not necessarily contiguous) subsequences occurring in a sequence. For example, already in 1749, Euler defined the Eulerian polynomials, which are the generating function for the number of descents in permutations. Almost 200 years later, Erdős and Szekeres [9] proved the existence of an increasing or a decreasing subsequence of length at least $a + 1$ in a sequence of at least $n = a^2 + 1$ distinct reals. More precisely, they have shown the following theorem.

▶ **Theorem 1** (Erdős and Szekeres, 1935). *Every sequence of $ab + 1$ distinct real numbers contains an increasing subsequence of length at least $a + 1$ or a decreasing subsequence of length at least $b + 1$.*

The theorem of Erdős–Szekeres is strongly related to, and in fact also follows from, the well-known decomposition of Dilworth (see [18]) regarding chains and antichains in a finite partially ordered set. Dilworth's result can be restated in the context of the combinatorics of patterns in sequences of numbers as follows.

▶ **Theorem 2** (Dilworth, 1950). *Any finite sequence S of distinct real numbers can be partitioned into k ascending sequences, where k is the maximum length of a descending sequence in S.*

Recent surveys on the combinatorics of patterns occurring in sequences are [14, 15].

The study of patterns in sequences of numbers also has a well developed algorithmic side (see, e.g., [4, 8, 10, 13]). For instance, finding a longest increasing subsequence (not necessarily contiguous) contained in the input sequence is a basic problem in theoretical computer science, studied already from the 1960s [3, 16, 17], with applications in areas such as bioinfomatics and physics (see [19] and the references therein). In particular, in 1975 Fredman [10] presented an algorithm (which he attributed to Knuth, now considered folklore) computing the length of a longest increasing subsequence (LIS) in an array of n numbers in $O(n \log n)$ time, and proved that this is optimal for comparison-based algorithms. If required, the algorithm can be extended to retrieve such a subsequence. If the input sequence can be sorted in linear time (in particular, when the input sequence is a permutation of $\{1, \ldots, n\}$) and we do not require the algorithm to be comparison-based, the solution given by Fredman can be implemented in $O(n \log \log n)$ time, see [8] and the references therein. Fredman's algorithm is often called Patience Sorting, and has some connections to constructing the so-called Young Tableaux [3, 16].

We consider a notion that is strongly related to longest increasing subsequences (and longest decreasing subsequences). A *run* in a sequence of numbers is a maximal contiguous subsequence that is either increasing or decreasing. A *k-rollercoaster*, where $k \geq 3$, is a sequence of numbers whose every run has length at least k; 3-rollercoasters are called, for short, rollercoasters. For example, the sequence $(3, 6, 8, 10, 9, 5, 1, 2, 4, 7, 11)$ is a 4-rollercoaster with runs $(3, 6, 8, 10)$, $(10, 9, 5, 1)$, $(1, 2, 4, 7, 11)$. Given a sequence $S[1 : n] = (S[1], S[2], \ldots, S[n])$ of n distinct numbers, the k-rollercoaster problem is to find a maximum-size set of indices $i_1 < i_2 < \cdots < i_m$ such that $(S[i_1], S[i_2], \ldots, S[i_m])$ is a k-rollercoaster. In other words, this problem asks for a longest k-rollercoaster contained in the input sequence S.

There is a simple, but useful, geometrical interpretation of k-rollercoasters. The input sequence $S[1 : n]$ can be depicted as a set P of points in the plane by translating, for i from 1 to n, the number $S[i]$ to a point $p_i = (i, S[i])$. In this setting, a k-rollercoaster in S translates to a polygonal path in the plane, whose vertices are points of P, and such that every maximal sub-path, with positive- or negative-sloped edges, has at least k points. The rollercoaster $(3, 6, 8, 10, 9, 5, 1, 2, 4, 7, 11)$ is depicted in the left half of Figure 1. Two 4-rollercoasters occurring in the sequence $(3, 6, 1, 8, 7, 17, 13, 10, 11, 12, 9, 5, 14, 4, 2, 15, 16)$ are depicted in the right half of the same figure.

While rollercoasters seem interesting on their own as a combinatorial structure, the original motivation for their study was a connection to computational geometry and graph drawing, namely to point-set embeddings of *caterpillars* (see [5, 6] and the references therein). More precisely, constructing a long rollercoaster in a sequence of numbers was used as an

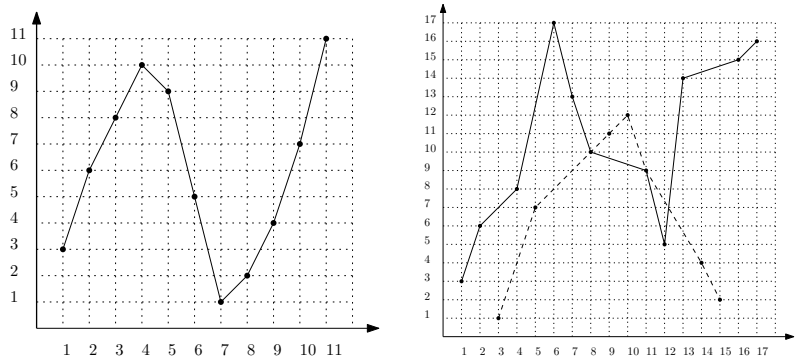


Figure 1 Left: a 4-rollercoaster $(3, 6, 8, 10, 9, 5, 1, 2, 4, 7, 11)$ with runs $(3, 6, 8, 10)$, $(10, 9, 5, 1)$, $(1, 2, 4, 7, 11)$. Right: two 4-rollercoasters, represented with a solid and, respectively, a dashed line, in $(3, 6, 1, 8, 7, 17, 13, 10, 11, 12, 9, 5, 14, 4, 2, 15, 16)$.

intermediate step towards obtaining a method of drawing a n-vertex top-view caterpillar, with L-shaped edges, on a set of $\frac{25}{3}n$ general orthogonal position points in the plane. This is currently the best known bound on the number of points required to draw such a graph.

In [5], the following results regarding k-rollercoasters were shown. First, from a combinatorial point of view, for $k = 3$, it was shown that the length of a longest rollercoaster contained in a sequence of $n \geq 7$ distinct numbers is at least $\lceil \frac{n}{2} \rceil$. As far as k-rollercoasters are concerned, it was shown that for $k \geqslant 4$ every sequence of $n \geqslant (k-1)^2 + 1$ distinct numbers contains a k-rollercoaster of length at least $\frac{n}{2(k-1)} - \frac{3k}{2}$. From an algorithmic point of view, both previously mentioned results were constructive, leading to an $O(n)$-time (respectively $O(n \log k)$) algorithm computing a long (but not necessarily a longest) rollercoaster (respectively, k-rollercoaster) contained in a sequence of n distinct numbers. A longest rollercoaster contained in such a sequence was computed by an extension of Fredman's algorithm in $O(n \log n)$-time, and if the input sequence is a permutation of $\{1, \ldots, n\}$ (or, more generally, sortable in linear time) in $O(n \log \log n)$ time. By further generalising this approach, an $O(nk \log n)$-time (respectively, $O(nk \log \log n)$-time) algorithm computing a longest k-rollercoaster in a sequence of n distinct numbers (respectively, a permutation of $\{1, \ldots, n\}$) can be obtained. Note that, by the theorem of Erdös and Szekeres, a sequence of n distinct numbers always contains a $\lfloor \sqrt{n} \rfloor$-rollercoaster, and the aforementioned algorithm computes a longest such rollercoaster in $O(n^{1.5} \log n)$ time.

Our contributions. We consider the problem of computing a longest k-rollercoaster in an input sequence $S[1 : n]$ and provide three results.

Firstly, we design a comparison-based algorithm computing the length of a longest k-rollercoaster in a sequence of n distinct numbers in $O(nk^2)$ time. Thus, we obtain an optimal linear-time algorithm for constant values of k, in particular for $k = 3$. This significantly improves the results of [5] and shows that, even though longest rollercoasters are related to longest increasing subsequences, the rich combinatorial structure of the former makes them provably easier to find. The starting point of our algorithm is the following natural dynamic programming formulation. For each $2 \leq i \leq k$, and for each element $S[j]$, we compute a longest (not necessarily contiguous) subsequence of S ending with $S[j]$ and with every run of length at least k, except for the last run, which has only i elements if $i < k$ and at least k elements if $i = k$. Now the difficulty is to find the predecessor $S[j']$ of $S[j]$ in such a subsequence in time proportional to k, in particular avoiding any kind of binary search.

We greedily decompose the input sequence into blocks with a certain property related to Dilworth's theorem and prove, by a careful case analysis, that j' must belong to the previous few such blocks. This, together with the special structure of the blocks and appropriate data structures, allows us to find j' in $O(k)$ amortised time.

Secondly, we focus on the case of large k. Given that both the previous and the new algorithm have at least linear dependency on k, it might seem plausible that this is inherent to the problem, for example that for $k \geq \lfloor \sqrt{n} \rfloor$ the running time of any algorithm needs to be $\Omega(n^{1.5})$. We show that this is not the case by designing a subquadratic algorithm that computes a longest k-rollercoaster in a sequence of n distinct numbers in $O(n \log^2 n)$ time. To obtain this result, we exploit the fact that if an increasing (respectively, decreasing) run in a longest k-rollercoaster extends from $S[i]$ to $S[j]$, then that run should be LIS (respectively, longest decreasing sequence, LDS for short) in $S[i : j]$. If one arranges the length of LIS (respectively, LDS) in $S[i : j]$ in an $n \times n$ matrix then the matrix has the anti-Monge property. It is known that all row maxima of an anti-Monge matrix can be found in $O(n)$ time [2], that is, in sublinear time w.r.t. the size of the matrix (given an oracle access to the elements of the matrix). Such properties have been successfully exploited to speed up certain dynamic programming algorithms. We also follow this route, and construct a longest k-rollercoaster using dynamic programming, essentially by gluing together LISs and LDSs of consecutive contiguous subsequences of S.

Thirdly, we show that any comparison-based algorithm computing a longest k-rollercoaster needs $\Omega(n \log k)$ comparisons. Our reasoning is similar to the one used by Fredman to show that any comparison-based algorithm computing a LIS needs $\Omega(n \log n)$ comparisons. We leave as an open problem to close the gap between the lower and upper bounds shown here.

The paper is organised as follows. After a series of preliminaries, we describe the $O(nk^2)$-time algorithm for computing the length of a longest k-rollercoaster, followed by the $O(n \log^2 n)$-time algorithm. We conclude with the lower bound for the number of comparisons needed to compute the length of a longest k-rollercoaster in a sequence of length n. The proofs omitted here for space reasons can be found in [12].

2 Preliminaries

We consider sequences of distinct real numbers and work in the comparison-based model. If S is a sequence of n numbers, then $|S| = n$ is the length of the sequence, and $S[i]$ denotes its i^{th} element. A *subsequence* of S is a sequence $(S[i_1], S[i_2], \ldots, S[i_m])$, defined by specifying the indices $1 \leq i_1 < i_2 < \ldots < i_m \leq n$. For $1 \leq i \leq j \leq n$, $S[i : j]$ denotes the *contiguous subsequence* $(S[i], S[i+1], \ldots, S[j])$; in particular, $S[1 : n]$ denotes the entire S. Note that unless explicitly stated, a subsequence is not necessarily contiguous. An increasing subsequence (respectively, decreasing subsequence) of S is a subsequence $(S[i_1], S[i_2], \ldots, S[i_m])$ such that $S[i_j] < S[i_{j+1}]$, for all $1 \leq j \leq m-1$ (respectively, $S[i_j] > S[i_{j+1}]$, for all $1 \leq j \leq m-1$). A longest increasing (respectively, decreasing) sequence, for short LIS (respectively, LDS), is an increasing (respectively, decreasing) sequence with the largest possible length. Fredman gave an $O(n \log n)$-time algorithm for computing the length of LIS, denoted res in Algorithm 1. A byproduct of this algorithm is a partition of $S[1 : n]$ into res non-increasing subsequences that can be obtained by creating, for every $1 \leq j \leq \mathsf{res}$, a list of elements that has been stored in $R[j]$.

A *run* in a sequence of numbers is a maximal contiguous subsequence that is increasing or decreasing. A *k-rollercoaster* is a sequence of numbers such that every run has length at least k; 3-rollercoasters are called, for short, rollercoasters. Given a sequence $S[1 : n]$

Algorithm 1 Finding the length of LIS of S.

```
1: R[0] ← 0
2: res ← 0
3: for i ← 1 to n do
4:     k ← max{j : R[j] < S[i]}        ▷ binary search over R[0] < R[1] < R[2] < ...
5:     R[k + 1] ← S[i]
6:     res ← max{res, k + 1}
7: return res
```

we are interested in finding its longest subsequence that is a k-rollercoaster. To make the exposition easier to follow, we focus on finding the length of such a subsequence. Recovering the subsequence itself is, in all our algorithms, rather straightforward.

3 Computing a Longest k-Rollercoaster in $O(nk^2)$-Time

In this section we show how to find a longest k-rollercoaster of $S[1:n]$ in $O(nk^2)$ time.

We begin our algorithm with a preprocessing phase. An alternating k-decomposition of $S[1:n]$ is a partition of $S[1:n]$ into contiguous subsequences (called parts) $S_1, S_2, \ldots, S_m$ such that the length of LIS in the odd parts (S_1, S_3, S_5, and so on) is k while the length of LDS in the even parts is k, possibly smaller for the very last part, and additionally by removing the last element of any odd (even) part we obtain a sequence with LIS (LDS) of length less than k. In other words, for $\ell \geq 1$, S_ℓ is either the shortest contiguous subsequence of S that follows directly after $S_1 \cdots S_{\ell-1}$ and has for ℓ odd (even) a LIS (respectively, LDS) of length k, if such a subsequence exists, or the whole remaining part of S otherwise. For example, an alternating 3-decomposition of $S = (1, 4, 2, 5, 8, 7, 6, 3)$ is $(1, 4, 2, 5), (8, 7, 6), (3)$.

▶ **Lemma 3.** *An alternating k-decomposition of $S[1:n]$ can be found in $O(n \log k)$ time.*

Proof. By terminating Algorithm 1 as soon as $\mathsf{res} = k$ we can find the shortest prefix of S with LIS equal to k in $O(d \log k)$ time, where d is the length of the prefix. Then we find the shortest prefix of the remaining suffix of S with LDS equal to k, and repeat. Overall, this takes $O(n \log k)$ time because all parts are disjoint. ◀

▶ **Proposition 4.** *Let A be a k-rollercoaster in S. Any part S_ℓ contains elements of at most four consecutive runs of A.*

Proof. By contradiction. Let S'_ℓ be S_ℓ without the last element. If S_ℓ contains elements of five consecutive runs of A then S'_ℓ contains elements of four consecutive runs of A, and hence all elements of two such consecutive runs. Thus, if S_ℓ is an odd (even) part then S'_ℓ contains LIS (LDS) of length k, which contradicts the definition of an alternating k-decomposition. ◀

By Dilworth's theorem, a part with LIS of length k can be decomposed into k decreasing subsequences, and such a decomposition can be obtained as a byproduct of Algorithm 1. Thus, we can decompose each part into up to k monotone (increasing or decreasing, depending on whether the part is odd or even) subsequences. These subsequences can be then merged to obtain a sorted list P_ℓ of all elements in the corresponding part S_ℓ in $O(n \log k)$ overall time, for example by first merging pairs of subsequences, then quadruples, and so on.

Before moving on to the description of our algorithm, we need a combinatorial lemma that relates an alternating k-decomposition to a longest rollercoaster.

▶ **Lemma 5.** *Suppose that $x = S[j]$ is a non-first element occurring in an increasing run of a longest k-rollercoaster, and $y = S[j']$ is its predecessor in the same run, and consider an alternating k-decomposition of $S[1:n]$. Then either x and y are in the same part S_i, or y is in one of the parts $S_{i-4}, S_{i-3}, S_{i-2}, S_{i-1}$.*

Proof. By contradiction. Suppose that there are at least four parts between x and y, i.e., x is in S_i and y is in some S_k with $k < i-4$. Let r denote the run in the k-rollercoaster that contains x and y, let d be the length of r, and let ℓ be such that $r[\ell] = y$ and $r[\ell+1] = x$. We assume that r is an increasing run (see Figure 2); the case when r is decreasing can be treated in the same way.

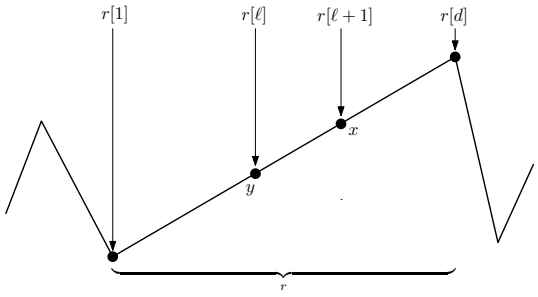


Figure 2 The increasing run r from Lemma 5, with the points x and y highlighted.

Consider the following four cases:

1. $\ell \leq k-1$ (i.e., there are at most $k-2$ elements in r before y) and $k-2 \geq d-\ell-1$ (there are at most $k-2$ elements in r after x).
2. $\ell \leq k-1$ and $k-1 \leq d-\ell-1$ (there at least $k-1$ elements in r after x).
3. $\ell \geq k$ (there are at least $k-1$ elements in r before y) and $k-2 \geq d-\ell-1$.
4. $\ell \geq k$ and $k-1 \leq d-\ell-1$.

Recall that there are at least four whole parts between x and y. Therefore, in particular there are three consecutive parts $S_{i'}, S_{i'+1}$, and $S_{i'+2}$ such that the first has LIS of length k, the second has LDS of length k, and the third has LIS of length k.

In the first case, we replace $r[2:d-1]$ with LIS of $S_{i'}$, the LDS of $S_{i'+1}$, and LIS of $S_{i'+2}$. It is straightforward to verify that we obtain a valid k-rollercoaster, and because we remove at most $2k-4$ elements and add at least $3k$, this creates a longer k-rollercoaster, which is a contradiction. In the second case, we replace $r[2:\ell]$ with LIS of $S_{i'}$ and LDS of $S_{i'+1}$. Again, it is straightforward to verify that we obtain a valid longer k-rollercoaster, because we remove at most $k-2$ elements and add at least $2k$. Similarly, in the third case, we replace $r[\ell+1:d-1]$ with LDS of $S_{i'+1}$ and LIS of $S_{i'+2}$ to obtain a longer k-rollercoaster. Finally, in the fourth case we simply insert LDS of $S_{i'+1}$ between x and y to obtain a longer k-rollercoaster. ◀

After the initial preprocessing phase we apply dynamic programming. For $1 \leq i \leq k$, we say that a subsequence of S (not necessarily contiguous) is a $(k,i)_+$-rollercoaster if it ends with an increasing run of length exactly i when $i < k$ and at least k when $i = k$, while every other run is of length at least k. Additionally, we consider k-rollercoaster ending with a decreasing run as $(k,1)_+$-rollercoaster. We want to construct, for every $1 \leq i \leq k$ and $1 \leq j \leq n$, a longest $(k,i)_+$-rollercoaster ending with $S[j]$. To this end we calculate $M_+[j,i]$, the position in S of the predecessor of $S[j]$ in such a $(k,i)_+$-rollercoaster, and $L_+[j,i]$, the length of the respective $(k,i)_+$-rollercoaster. A $(k,i)_-$-rollercoaster is defined similarly, except

that the last run should be decreasing, and we also calculate the values $M_-[j,i]$ and $L_-[j,i]$, defined similarly to the above and corresponding to such a $(k,i)_-$-rollercoaster. We only describe in detail how to compute $M_+[j,i]$ and $L_+[j,i]$, as $M_-[j,i]$ and $L_-[j,i]$ are computed analogously. The computation proceeds from left to right, that is, we iterate over the parts $S_1, S_2, \ldots$ and compute, for every element $S[j]$ of the current part S_ℓ, the values of $M_+[j,i]$ and $L_+[j,i]$ for every $1 \leq i \leq k$. See Algorithm 2 for a high-level overview of the algorithm.

Algorithm 2 Computing the length of a longest k-rollercoaster.

Find an alternating k-decomposition $S_1, \ldots, S_m$ of S.
for $1 \leq \ell \leq m$ **do**
    Merge the k monotone subsequences constituting S_ℓ to obtain a single sorted list P_ℓ.
for $1 \leq \ell \leq m$ **do**
    ▷ For each $S[j]$ in S_ℓ and $1 \leq i \leq k$, we compute the following:
        $M_+[j,i]$: position in S of the predecessor of $S[j]$ in its $(k,i)_+$-rollercoaster
        $L_+[j,i]$: length of the respective $(k,i)_+$-rollercoaster
    for $2 \leq i \leq k$ **do**
        for $1 \leq d \leq 4$ and each $S[j] \in S_\ell$ in the order of their occurrences in P_ℓ **do**
            Find $M_+^d[j,i]$ and $L_+^d[j,i]$.
        for each $S[j] \in S_\ell$ **do**
            $L_+[j,i] \leftarrow \max\{L_+^d[j,i] : 1 \leq d \leq 4\}$
            Set $M_+[j,i]$ so that it corresponds to $L_+[j,i]$.
    Compute, for each $S[j] \in S_\ell$, $L_-[j,i]$ and $M_-[j,i]$ with a similar approach.
    repeat 4 times
        for each $S[j] \in S_\ell$ **do**
            $L_+[j,1] \leftarrow \max\{L_-[j,k], 1\}$
            $M_+[j,1] \leftarrow M_-[j,k]$ if $L_-[j,k] > 0$ and 0 otherwise
        for $2 \leq i \leq k$ **do**
            for each $S[j] \in S_\ell$ in the order of their occurrences in P_ℓ **do**
                Find $M'_+[j,i]$, $L'_+[j,i]$ using decomposition of S_ℓ into k monotone sequences.
            for each $S[j] \in S_\ell$ **do**
                $L_+[j,i] \leftarrow \max\{L_+[j,i], L'_+[j,i]\}$
                Update $M_+[j,i]$ so that it corresponds to $L_+[j,i]$.
        Update, for each $S[j] \in S_\ell$, $L_-[j,i]$ and $M_-[j,i]$ with a similar approach.
return $\max\{\max\{L_-[j,k], L_+[j,k]\} : 1 \leq j \leq n\}$

When we begin computing the arrays $M_+[\cdot,i]$, $L_+[\cdot,i]$, $M_-[\cdot,i]$ and $L_-[\cdot,i]$, for $2 \leq i \leq k$, corresponding to all $S[j] \in S_\ell$, we have already computed $M_+[j',1]$, $M_+[j',2], \ldots, M_+[j',k]$ and $L_+[j',1]$, $L_+[j',2], \ldots, L_+[j',k]$, as well as $M_-[j',1]$, $M_-[j',2], \ldots, M_-[j',k]$ and $L_-[j',1]$, $L_-[j',2], \ldots, L_-[j',k]$, for every $S[j'] \in S_{\ell'}$ such that $\ell' < \ell$.

We start with computing the values $M_+[\cdot,i]$, $L_+[\cdot,i]$, $M_-[\cdot,i]$ and $L_-[\cdot,i]$, for $2 \leq i \leq k$, assuming that the predecessor $S[j']$ of $S[j]$ in its corresponding rollercoaster belongs to $S_{\ell-d}$, for some $1 \leq d \leq 4$. In such case the longest rollercoaster ending at $S[j']$ has been already correctly determined and the computation is quite straightforward. If $S[j']$ also belongs to S_ℓ, we must be more careful to guarantee that the longest rollercoaster ending at $S[j']$ is already known. We proceed in iterations. In the t^{th} iteration, we guarantee to compute the values such that at most t runs of the corresponding rollercoaster contain elements from S_ℓ. By Proposition 4, four iterations are enough. In a single iteration, we start with computing

the initial values $M_+[\cdot,1]$, $L_+[\cdot,1]$, $M_-[\cdot,1]$ and $L_-[\cdot,1]$ corresponding to $S[j]$ being the first element of its run. These values can be simply copied from the already known $M_-[\cdot,k]$, $L_-[\cdot,k]$, $M_+[\cdot,k]$ and $L_+[\cdot,k]$ corresponding to $S[j]$ being the last element of a rollercoaster with less than t runs containing elements from S_ℓ (or set to 1 corresponding to $S[j]$ being the only element in the rollercoaster). This is correct because a $(k,1)_+$-rollercoaster is actually either a $(k,k)_-$-rollercoaster or a sequence consisting of a single element. Then, we calculate the values $M_+[\cdot,i]$, $L_+[\cdot,i]$, $M_-[\cdot,i]$ and $L_-[\cdot,i]$, for $2 \leq i < k$, such that the predecessor $S[j'] \in S_\ell$ belongs to the same run as $S[j]$. By performing the calculation for $i = 2,3\ldots,k-1$ in this order we guarantee that the longest rollercoaster ending at the predecessor $S[j'] \in S_\ell$ is already known for all $S[j] \in S_\ell$, but the computation is still not completely trivial and requires a different approach depending on whether S_ℓ was decomposed into at most k increasing, respectively decreasing, subsequences. Finally, we extend this to $i = k$.

$M_+[j,i]$ belongs to $S_{\ell-d}$ for some $1 \leq d \leq 4$. We process $S_{\ell-d}$ to identify some candidates, denoted $M^d_+[j,i]$ and $L^d_+[j,i]$, for $M_+[j,i]$ and $L_+[j,i]$, respectively, for every $S[j] \in S_\ell$. The idea is to compute these candidates in the order in which the elements $S[j]$ occur on the sorted list P_ℓ. So, let us consider P_ℓ and $P_{\ell-d}$. For each element $S[j]$ in the current part we want to identify a longest $(k,i-1)_+$-rollercoaster ending in $S_{\ell-d}$ with an element less than $S[j]$. Thus, as $P_{\ell-d}$ is increasing, for every element of the current part we need to consider all elements in a prefix of $P_{\ell-d}$. Also, if $S[j']$ is to the right of $S[j]$ in P_ℓ, that is, $S[j'] \geq S[j]$, then the prefix of $P_{\ell-d}$ that we need to consider to compute $M^d_+[j',i]$ is at least as long as the prefix that we need to consider to compute $M^d_+[j,i]$. Therefore, we can use two pointers to sweep through P_ℓ and $P_{\ell-d}$ from left to right, and obtain the information needed to compute $M^d_+[j,i]$ and $L^d_+[j,i]$, for every $S[j] \in S_\ell$. At the beginning the pointers point to the first element of P_ℓ and $P_{\ell-d}$, respectively. Say that the current element in P_ℓ and $P_{\ell-d}$ is $S[j]$ and $S[h]$, respectively (we update indices j and h along with the pointers). We keep moving forward the pointer corresponding to $S[h]$ until we find an element $S[h] > S[j]$. Then we set $M^d_+[j,i] = h'$ and $L^d_+[j,i] = L_+[h',i]+1$, where $S[h']$ is an element occurring earlier than $S[h]$ in $P_{\ell-d}$ with the largest value of $L_+[h',i-1]$. The element $S[h']$ is maintained as we move from left to right in $P_{\ell-d}$. Then we proceed to the next element in P_ℓ. Overall, computing candidates $M^d_+[j,i]$ and $L^d_+[j,i]$, for every $S[j] \in S_\ell$, takes $O(|S_{\ell-d}| + |S_\ell|)$ time.

$M_+[j,i]$ belongs to S_ℓ decomposed into k increasing subsequences. Recall that we have already computed $M_+[j',i']$ and $L_+[j',i']$ for every $i' < i$ and $S[j'] \in S_\ell$, and the goal is to identify candidates, denoted $M'_+[j,i]$ and $L'_+[j,i]$, for $M_+[j,i]$ and $L_+[j,i]$, respectively, for every $S[j] \in S_\ell$. Consider the decomposition of S_ℓ into k increasing subsequences $I_1, I_2, \ldots, I_k$. The elements of every sequence are increasing w.r.t. their value and w.r.t their position in S. Consider an element $S[j] \in I_a$ and $1 \leq b \leq k$ (possibly $a = b$). The elements of I_b that can be the predecessor of $S[j]$ in a $(k,i)_+$-rollercoaster (that is, possible candidates for $M_+[j,i]$) are both less w.r.t. value and w.r.t. position in S. Thus, these elements form a prefix of I_b, and for every $S[j] \in I_a$ and $1 \leq b \leq k$ we want to maximise $L_+[h',i]$ over all $S[h']$ in such a prefix. As in the previous case, we can use two pointers to sweep through I_a and I_b and compute, for every $S[j] \in I_a$, the element $S[h'] \in I_b$ that could precede $S[j]$ in a $(k,i)_+$-rollercoaster with the largest value of $L_+[h',i-1]$. Finally, we set $M'_+[j,i]$ and $L'_+[j,i]$ to correspond to the largest such value among all $1 \leq b \leq k$. Overall, computing the candidates $M'_+[j,i]$ and $L'_+[j,i]$, for every $S[j] \in S_\ell$, takes $O(k|S_\ell|)$ time.

$M_+[j,i]$ belongs to S_ℓ decomposed into k decreasing subsequences. This is the most complicated case. Recall that the decomposition into k decreasing subsequences $D_1, D_2, \ldots, D_k$ was obtained with Algorithm 1. In more detail, D_a consists of elements assigned to $R[a]$ throughout the execution of the algorithm. Thus, if $S[j] \in D_a$ then the predecessor of $S[j]$ in a sought longest $(k,i)_+$-rollercoaster, denoted $S[j']$, must belong to D_b for some $1 \leq b < a$. Indeed, Algorithm 1 first processes $S[j']$ and then $S[j]$, so if $S[j'] \in D_b$ then $R[b] \leq S[j']$ when processing $S[j]$ and consequently $S[j'] < S[j]$ implies that $S[j]$ is assigned to $R[a]$ with $a > b$. So, we first compute the candidates $M'_+[j,i]$ and $L'_+[j,i]$ for every $S[j] \in D_1$, then for every $S[j] \in D_2$, and so on. That is, consider a decreasing subsequence D_a and suppose that we have already computed the desired result for all elements in $D_1, D_2, \ldots, D_{a-1}$. Note that at this point we have already computed, for every $S[j] \in D_1 \cup \ldots \cup D_{a-1}$, the values of $M^d_+[j,i]$ and $L^d_+[j,i]$, for $1 \leq d \leq 4$, as well as the values $M'_+[j,i]$ and $L'_+[j,i]$ corresponding to the current iteration. Thus, we are already able to set $M_+[j,i]$ and $L_+[j,i]$ by choosing the option that maximises the length of the corresponding $(k,i)_+$-rollercoaster, which is important when extending this case to $i = k$.

Consider an element $S[j] \in D_a$ and $1 \leq b < a$. The elements of D_b that can be the predecessor of $S[j]$ in a $(k,i)_+$-rollercoaster (that is, possible candidates for $M_+[j,i]$) are both less w.r.t. value and w.r.t. position in S, similarly as in the previous case. The difference is that now these elements form contiguous subsequence X of D_b that is not necessarily a prefix. The first element of X can be found by searching for the first element with sufficiently small value, while its last element can be found by searching the last element with sufficiently small position (note that X might be empty). Let $S[j']$ be the next element after $S[j]$ in D_a, and Y be its corresponding contiguous subsequence of D_b consisting of possible predecessors in a $(k,i)_+$-rollercoaster. Clearly, $S[j] > S[j']$ while $j < j'$. Thus, the first element of Y is either the same as the first element of X or occurs after the first element of X in D_a, while the last element of Y is either the same as the last element of X or occurs after the last element of X in D_a (assuming that both X and Y are non-empty). Thus, we sweep through D_a while maintaining the current contiguous subsequence X of D_b corresponding to the possible predecessors of the current $S[j] \in D_a$. This requires the following tool.

▶ **Lemma 6** ([11]). *There is a data structure that maintains a list of elements under the following operations: pop an element from the front, push an element in the back, and return the maximum element in the current list, each in $O(1)$ time.*

When processing the current element $S[j] \in D_a$ we maintain the first element $S[f] \in D_b$ such that $S[f] < S[j]$ and the last element $S[\ell] \in D_b$ such that $\ell < j$. Then X consists of all elements between $S[f]$ and $S[\ell]$ in D_b (inclusive), and is maintained in a structure from Lemma 6 storing the lengths of their corresponding $(k,i)_+$-rollercoaster, that is, the already known value of $L_+[\cdot, i-1]$. This allows us to extract the element $S[j'] \in X$ with the largest value of $L_+[j', i-1]$, and set $M'_+[j,i] = j'$ and $L'_+[j,i] = L_+[j', i-1] + 1$ in constant time, while updating f and ℓ takes amortised constant time. Overall, computing the candidates $M'_+[j,i]$ and $L'_+[j,i]$, for every $S[j] \in S_\ell$, takes $O(k|S_\ell|)$ time.

Case $i = k$. To compute $M_+[j,k]$ and $L_+[j,k]$, we first use exactly the same approach as before for $i = k$, so consider the values of $M_+[\cdot, k-1]$ and $L_+[\cdot, k-1]$. But this only allows us to compute the length of a longest $(k,k)_+$-rollercoaster with the last run of length exactly k. To extend this to arbitrary $(k,k)_+$-rollercoasters with the last run of length greater than k we additionally run the same algorithm but instead of looking at $M_+[\cdot, k-1]$ and $L_+[\cdot, k-1]$ we use $M_+[\cdot, k]$ and $L_+[\cdot, k]$, including the values already computed in this extra step in the

third case. The reason why this works is that, due to the order in which we consider the elements of S_ℓ, at the moment when we compute the length of a longest $(k,k)_+$-rollercoaster ending with $S[j]$, and which may have more than k elements in the final run, we have already computed the length of a longest $(k,k)_+$-rollercoaster ending with any element $S[j']$ which may be a predecessor of $S[j]$ on the respective $(k,k)_+$-rollercoaster.

Conclusion. With these final remarks, our algorithm is completely described. It only remains to find the element $S[j]$ for which $\max\{L_-[j,k], L_+[j,k]\}$ is maximum. The correctness follows from the comments made throughout its description. To compute the complexity, it is enough to note that each part S_ℓ of the partition of S is processed in $O(k|S_\ell|)$ time, for each $2 \leq i \leq k$. Adding this up, we get that the total complexity of our algorithm is $O(nk^2)$.

▶ **Theorem 7.** *For every sequence $S[1:n]$ and $k \geq 3$, the length of a longest k-rollercoaster in S can be found in $O(nk^2)$-time.*

4 Computing a Longest k-Rollercoaster in $O(n \log^2 n)$-Time

Before we describe our algorithm, we introduce two preliminary procedures. Firstly, we introduce the definition of an anti-Monge matrix and the algorithm for finding the maximum in every column of such a matrix. Secondly, we describe the algorithm for finding LIS in contiguous subsequences of the input sequence. Finally, we describe the algorithm computing a longest k-rollercoaster in this sequence, using the previously developed tools as black boxes.

Monge matrices. Let A be an $n \times n$ matrix, and $A[i,j]$ denote its element in the i^{th} row from the top and the j^{th} column from the left. A is *Monge* (respectively, *anti-Monge*) if, for every $1 \leq i < j \leq n$ and $1 \leq k < \ell \leq n$, the *Monge equality* holds, namely $A[i,k] + A[j,\ell] \leq A[i,\ell] + A[j,k]$ (respectively, $A[i,k] + A[j,\ell] \geq A[i,\ell] + A[j,k]$). An $n \times n$ *falling staircase anti-Monge matrix* is a matrix with blanks such that for every blank all elements below and to the left are blanks, and the anti-Monge inequality holds whenever the four concerned elements are non-blank. Similarly, an $n \times n$ *reverse falling staircase anti-Monge matrix* is a matrix with blanks such that for every blank all elements above and to the right are blanks, and the anti-Monge inequality holds whenever the four concerned elements are non-blank. Finally, an $n \times n$ matrix A is *totally monotone* if, for every $1 \leq i < j \leq n$ and $1 \leq k < \ell \leq n$, $A[i,k] \leq A[i,\ell]$ implies $A[j,k] \leq A[j,\ell]$.

0	1	2	2	2
-1	0	1	1	2
-2	-1	0	1	2
-3	-2	-1	0	1
-4	-3	-2	-1	0

4				
3	2			
2	1			
4	4	2	6	

	1	2	2	2
			1	2
			1	2
				1

Figure 3 Anti-Monge matrix, reverse falling staircase anti-Monge matrix, and falling staircase anti-Monge matrix.

Let us now recall some basic facts regarding Monge matrices.

▶ **Observation 8.** *Adding the same value to every element in a row (or a column) of an anti-Monge matrix results in an anti-Monge matrix.*

▶ **Observation 9.** *To check if an array is anti-Monge it is sufficient to check if every contiguous 2×2 submatrix is anti-Monge.*

The following lemma follows from the well-known SMAWK algorithm [2].

▶ **Lemma 10** (Lemma 3.3 in Aggarwal et al. [1]). *All row maxima in a reverse falling staircase totally monotone matrix can be found in $O(n)$ time.*

By transposing the matrix and observing that being anti-Monge implies being totally monotone we obtain the following.

▶ **Corollary 11.** *All column maxima in a falling staircase anti-Monge matrix can be found in $O(n)$ time.*

LIS-in-range queries. Let $S[1:n]$ be the input sequence. Define M as an $(n+1)\times(n+1)$ matrix with 0-indexed rows and columns, such that $M[i,j]$ is the length of LIS in $S[i+1:j]$ for $i<j$ and $M[i,j]=j-i$ otherwise (the anti-Monge matrix in Figure 3 is such a matrix for the sequence $(3,4,1,2)$). As hinted by our example, this matrix turns out to have a rather special structure as observed by Tiskin [20]. We describe this structure in the following.

Let S' be the sequence obtained by sorting S (recall that S consists of distinct elements), and observe that LIS of S is the same as a longest common sequence (LCS, for short) of S and S'. Thus, we can think that $M[i,j]$ is LCS of S' and $S[i+1:j]$. As such, the following result can be shown (see [20] and the references therein).

▶ **Lemma 12.** *M is anti-Monge.*

Our algorithm needs to access the elements of M. Since the matrix contains $(n+1)^2$ elements, it is too large to be explicitly stored in memory. Fortunately, Tiskin also showed how to create in $O(n\log^2 n)$ time an $O(n)$-space implicit representation of M that allows us to obtain any of its elements in $O(\log n)$ time [20]. Before we present the internals of this representation, we need to introduce some additional definitions illustrated in Figure 4.

▶ **Definition 13.** *Let A be any $n\times n$ matrix. Its distribution matrix A^Σ is an $(n+1)\times(n+1)$ matrix defined by $A^\Sigma[x,y]=\sum_{i\geq x,j<y}A[i,j]$, for every $1\leq x\leq n+1, 1\leq y\leq n+1$.*

▶ **Definition 14.** *A permutation matrix is a square matrix that has exactly one 1 in every row and column, and the remaining elements are equal to 0.*

0	1	0
1	0	0
0	0	1

0	1	2	3
0	1	1	2
0	0	0	1
0	0	0	0

Figure 4 A permutation matrix A and its distribution matrix A^Σ.

Now, we can provide the final ingredients of the construction. For two strings w_1 and w_2 of length d, Tiskin defines in [20] a $(2d+1)\times(2d+1)$ matrix L in the following way. Let w_2' be the string equal to $?^d w_2 ?^d$, whose positions are indexed from $-(d-1)$ to $2d$. The rows of L are indexed from $-d$ to d, while the columns of L are indexed from 0 to $2d$. The elements of L are defined by $L[i,j]=\mathrm{LCS}(w_1, w_2'[i+1:j])$ if $j>i$, and $L[i,j]=j-i$ otherwise. In this definition, it is assumed that ? matches any character. If w_2 is the input sequence S and w_1 is S' then, for $0\leq i,j\leq n$ we have $L[i,j]=M[i+1,j+1]$. Tiskin proved (Theorem 4.10 in [20]) that there exists $2d\times 2d$ permutation matrix P such that $L[i,j]=j-i-P^\Sigma[i,j]$. Furthermore, he provided an $O(n\log^2 n)$-time algorithm that finds all the

non-zero entries of P (Algorithm 8.2 in [20]). Having all the non-zero entries of P we can apply a dominance counting structure of Chazelle [7] that can be constructed in $O(n \log n)$ time, uses $O(n)$ space, and calculates $P^{\Sigma}[i, j]$ and hence also $M[i+1, j+1]$ in $O(\log n)$ time. Summarising, in $O(n \log^2 n)$ time we obtain a structure that returns any element of M in $O(\log n)$ time. We similarly obtain a matrix storing the length of LDS of every $S[i+1 : j]$.

Description of the algorithm. Let $S[1 : n]$ be the input sequence. For every $1 \leq x \leq n$, let $\mathsf{res}[x]$ be the length of a longest k-rollercoaster in $S[1 : x]$, and $\mathsf{inc}[x]$ (respectively, $\mathsf{dec}[x]$) be the length of a longest k-rollercoaster in $S[1 : x]$ with the last run increasing (respectively, decreasing). Note that we do not require that these k-rollercoasters contain $S[x]$. Then, $\mathsf{res}[x] = \max\{\mathsf{dec}[x], \mathsf{inc}[x]\}$, for $1 \leq x \leq n$. Firstly, we introduce two structural lemmas.

▶ **Lemma 15.** *Let A be a k-rollercoaster in $S[1 : i]$ with the last run decreasing, and r be an increasing subsequence in $S[i : n]$ such that $|r| \geq k$. Then there exists a k-rollercoaster in $S[1 : n]$ of length at least $|A| + |r| - 1$ with the last run increasing.*

Proof. Let A' be the sequence consisting of all elements from both A and r. Recall that a sequence is a k-rollercoaster if every run has length at least k. In order to show that A' is a k-rollercoaster with last run increasing we need to consider three cases: the first element of r is the last element of A, the first element of r is greater than the last element of A, and the first element of r is less than the last element of A.

In the first case, all runs in A' but the last are the same as in A, and the last run is equal to r. Since A is a k-rollercoaster and $|r| \geq k$ we conclude that A' is a k-rollercoaster. A and r have one common element, so $|A'| = |A| + |r| - 1$.

In the second case, all runs in A' but the last are also the same as in A, and the last run consists of the last element of A and r. Again we conclude that A' is a k-rollercoaster. Since A and r have no common elements, $|A'| = |A| + |r|$.

In the third case, all runs in A' but the last two are the same as in A. The second-to-last run in A' consist of the last run of A and the first element of r, and the last run in A' is r. Hence, A' is a k-rollercoaster. Since A and r have no common elements, $|A'| = |A| + |r|$. ◀

▶ **Lemma 16.** *Consider a longest k-rollercoaster in $S[1 : n]$ with the last run increasing (respectively, decreasing), and let r be its last run with the first element $S[i]$. Then r is a longest increasing (respectively, decreasing) subsequence in $S[i : n]$.*

Proof. By contradiction. Let A be a longest k-rollercoaster from the statement of the lemma, and suppose that there exists a longer increasing sequence r' in $S[i : n]$. Let A' be the prefix of A ending at $S[i]$. Observe that $|A'| = |A| - |r| + 1$. Then by Lemma 15 there exists a k-rollercoaster in S of length at least $|A'| + |r'| - 1 = |A| - |r| + |r'| > |A|$. ◀

The above lemmas allow us to obtain the formula for calculating the arrays inc and dec. Recall that $M[i, j]$ is the length of LIS in $S[i+1 : j]$. Let M' be the matrix obtained from M by replacing all elements less than k by $-\infty$, and let $Z(j, j')$ be the set of indices $j \leq i \leq j'$ such that length of LIS in $S[i : j']$ is at least k (or, in other words, $M'[i-1, j'] \neq -\infty$).

▶ **Proposition 17.** *For every $1 \leq x \leq n$, the following holds:*

$$\mathsf{inc}'[x] = \max\{\mathsf{dec}[i] + M'[i-1, x] - 1 : i \in Z(1, x)\}, \mathsf{inc}[x] = \max\{\mathsf{inc}'[x], M'[0, x]\}.$$

If $Z(1, x)$ is empty then we set $\mathsf{inc}'[x] = 0$.

Proof. By Lemma 15 we obtain that for every $i \in Z(1,x)$ there exists a k-rollercoaster in $S[1:x]$ with the last run increasing of length at least $\mathsf{dec}[i] + M'[i-1,x] - 1$. We conclude that $\mathsf{inc}'[x]$ is less or equal to the length of a longest k-rollercoaster with the last run increasing in $S[1:x]$. Observe that $M'[0,x]$ corresponds to an increasing run of length at least k or is equal to $-\infty$. We obtain that $\mathsf{inc}[x]$ is less or equal than the length of a longest k-rollercoaster with the last run increasing in $S[1:x]$.

For the converse, consider a k-rollercoaster A with the last run increasing in $S[1:x]$. If A consists of just a single run then its length is $M'[0,x]$. Otherwise, let $S[i]$ be the first element in the last run of A. Then by Lemma 16 the length of the last run is equal to $M'[i-1,x]$ and the length of A is $\mathsf{dec}[i] + M'[i-1,x] - 1$. Overall, the length of A is at most $\mathsf{inc}[x]$. ◀

Proposition 17 cannot be applied directly if we aim to achieve the announced $O(n \log^2 n)$ time complexity, and we need to introduce some auxiliary definitions. For every $1 \leq d \leq x$ we define $\mathsf{inc}_d[x]$ as follows:

$$\mathsf{inc}'_d[x] = \max\{\mathsf{dec}[i] + M'[i-1,x] - 1 : i \in Z(1,d-1)\}, \mathsf{inc}_d[x] = \max\{\mathsf{inc}'_d[x], M'[0,x]\}.$$

If $Z(1,d-1)$ is empty then we set $\mathsf{inc}'_d[x] = 0$. In other words, $\mathsf{inc}_d[x]$ is equal to the length of a longest k-rollercoaster in $S[1:x]$ with the last run increasing and starting at an element $S[i]$ with $i < d$ or LIS of $S[1:n]$ of length at least k. Thus, $\mathsf{inc}_1[x]$ is equal to either 0 or the length of a LIS in $S[1:x]$. We similarly define $\mathsf{dec}_d[x]$.

▶ **Observation 18.** *For every $j > i-k+1$, $\mathsf{inc}_j[i] = \mathsf{inc}[i]$.*

We describe a function Compute that receives a contiguous subsequence $S[i:j]$ together with the previously calculated arrays $\mathsf{inc}_i[i:j]$ and $\mathsf{dec}_i[i:j]$, and returns the arrays $\mathsf{inc}[i:j]$ and $\mathsf{dec}[i:j]$. To calculate the length of a longest k-rollercoaster in $S[1:n]$ we invoke the function with the whole $S[1:n]$ and the arrays $\mathsf{inc}_1[1:n]$, $\mathsf{dec}_1[1:n]$ as arguments, and return the maximum over the two resulting arrays. Note that $\mathsf{inc}_1[1:n]$ and $\mathsf{dec}_1[1:n]$ can be calculated in $O(n \log n)$ time using Algorithm 1.

Let $m = \lceil \frac{i+j}{2} \rceil$. The main idea of Compute is to call the function recursively for the left half to calculate $\mathsf{inc}[i:m-1]$ and $\mathsf{dec}[i:m-1]$. The next step is to calculate $\mathsf{inc}_m[m:j]$ and $\mathsf{dec}_m[m:j]$ using tools from the previous paragraphs (as described below). Finally, we recursively calculate $\mathsf{inc}[m:j]$ and $\mathsf{dec}[m:j]$. Concatenating the results from both recursive calls gives us the desired result. This is summarised in Algorithm 3.

Algorithm 3 Computing the length of a longest k-rollercoaster.

```
1: procedure Compute(k, S[i:j], inc_i[i:j], dec_i[i:j])
2:     if j − i + 2 ≤ k then
3:         {inc[i:j], dec[i:j]} ← {inc_i[i:j], dec_i[i:j]}
4:         return {inc[i:j], dec[i:j]}
5:     m ← ⌈(i+j)/2⌉
6:     {inc[i:m−1], dec[i:m−1]} ← Compute(k, S[i:m−1], inc_i[i:m−1], dec_i[i:m−1])
7:     Compute inc_m[m:j] and dec_m[m:j]
8:     {inc[m:j], dec[m:j]} ← Compute(k, S[m:j], inc_m[m:j], dec_m[m:j])
9:     return {inc[i:j], dec[i:j]}
```

Computing $\mathsf{inc}_m[m:j]$ and $\mathsf{dec}_m[m:j]$. We only describe how to calculate $\mathsf{inc}_m[m:j]$, as $\mathsf{dec}_m[m:j]$ can be computed by a similar approach. Recall the previously introduced matrix M', obtained by replacing values less than k by $-\infty$ in M. Let A_{inc} be the $(m-i)\times(j+1-m)$ matrix with rows indexed from i to $m-1$ and columns indexed from m to j satisfying:

$$A_{\mathsf{inc}}[x,y]=\begin{cases}\mathsf{dec}[x]+M'[x-1,y]-1 & \text{when } M'[x-1,y]\neq-\infty,\\ \text{blank} & \text{otherwise.}\end{cases}$$

Since we are able to retrieve any element of M' in $O(\log n)$ time using LIS-in-range queries, and the value of $\mathsf{dec}[x]$, for every $i\le x\le m-1$, is already available, each element of A_{inc} can be calculated in $O(\log n)$ time. Furthermore, we have the following property.

▶ **Proposition 19.** *A is a falling staircase anti-Monge matrix.*

Proof. By Lemma 12 M is an anti-Monge matrix. By Observation 8 this is still the case if we add the same value to all elements in the same row.

To prove that A is a falling staircase matrix consider a non-blank element $A[i,j]$. Then $M[i,j]\ge k$. But this implies $M[i-1,j]\ge k$ and $M[i,j+1]\ge k$ (as long as $i>1$ and $j<n$), so all elements above and to the right are also non-blank as required. ◀

▶ **Proposition 20.** *For every $m\le\ell\le j$, $\mathsf{inc}_m[\ell]$ is equal to either $\mathsf{inc}_i[\ell]$ or the maximum in the ℓ^{th} column of A.*

Proof. For every $m\le\ell\le j$, $\mathsf{inc}_m[\ell]$ is equal to either $\mathsf{inc}_i[\ell]$ or $\max\{\mathsf{dec}[j]+M'[j-1,\ell]-1 : j\in Z(i,m-1)\}$. However, the latter is exactly the maximum in the ℓ^{th} column of A. ◀

▶ **Lemma 21.** *We can compute $\mathsf{inc}_m[m:j]$ and $\mathsf{dec}_m[m:j]$ in $O((j-i+1)\log n)$ time.*

Proof. By Proposition 20 computing $\mathsf{inc}_m[m:j]$ reduces to finding all the column maxima in A. Since A is a falling staircase anti-Monge matrix, we can use the algorithm from Corollary 11. Access to any element of A requires $O(\log n)$ time, so in total we obtain $O((j-i+1)\log n)$ time complexity. ◀

We can now state with the main result of this section.

▶ **Theorem 22.** *For every sequence $S[1:n]$ and $k\ge 3$, the length of a longest k-rollercoaster in S can be found in $O(n\log^2 n)$ time.*

Proof. The algorithm needs $O(n\log^2 n)$ preprocessing time to construct the LIS-in-range (and LDS-in-range) structure. We compute $\mathsf{inc}_1[1:n]$ and $\mathsf{dec}_1[1:n]$ in $O(n\log n)$ time using Algorithm 1. Then, we call the recursive function Compute. By Lemma 21 a call of the function on $S[i:j]$ takes $O((j-i+1)\log n)$ time, so its running time is described by the recurrence $T(n)=2T(n/2)+O(n\log n)$ that solves to $O(n\log^2 n)$. Thus, the overall time complexity is $O(n\log^2 n)$. ◀

5 Lower Bound

In the final section of our paper, we prove that any comparison-based algorithm computing the length of a longest k-rollercoaster in an permutation S of $\{1,\ldots,n\}$, for $4\le k\le\frac{n}{3}$, performs at least $\Omega(n\log k)$ comparisons. Let T be a binary comparison tree associated with an algorithm that computes the result. The number of comparisons made in the algorithm is equal to the height of T, and this is a lower bound on the execution time of the algorithm.

Let A be a partial ordering associated with a path from the root to some leaf of T. Since the algorithm cannot distinguish between permutations following the same path, every permutation consistent with A has to give the same result. Our approach is to first identify a set U of permutations of $\{1,\ldots,n\}$ such that $\log|U| = \Theta(n \log k)$, and any ordering associated with a leaf of T can be consistent with at most one permutation from U. Hence, the number of leaves in T is at least $|U|$. Since the height of a binary tree is at least logarithm of the number of leaves, this will show that the height of T, and hence also the number of comparison performed by the algorithm, is at least $\Omega(\log|U|) = \Omega(n \log k)$.

We first recall the set Γ of $\ell^{n-2\ell}$ permutations of $\{1,\ldots,n\}$ proposed by Fredman in [10], where ℓ is a parameter. These permutations are essentially different inputs S for an algorithm computing the length of LIS, each leading to a different leaf in the comparison tree.

So, essentially, we want to construct input sequences $(x_1,\ldots,x_n)$, with their elements $x_1,\ldots,x_n$ chosen so that certain linear orderings of the x_is are induced. To create a permutation from Γ we partition $(x_1,\ldots,x_n)$ into ℓ subsequences $P_1, P_2, \ldots, P_\ell$. To simplify the exposure, let ℓ_{prefix} of a sequence be its prefix of length ℓ, while the ℓ_{suffix} is its suffix of length ℓ; the remaining $n-2\ell$ elements are called ℓ_{middle} of the sequence. We partition $(x_1,\ldots,x_n)$ in the following way: the i^{th} element of ℓ_{prefix} (that is, x_i) and the i^{th} element of ℓ_{suffix} ($x_{n-\ell+i}$) belong to P_i. Each element from ℓ_{middle} of the sequence belongs to an arbitrary chosen part P_j. This gives us $\ell^{n-2\ell}$ different partitions. For a partition $P_1,\ldots,P_\ell$, we assign values from $\{1,\ldots,n\}$ to the input sequence in such a way, that the elements of each part P_i form a decreasing sequence and, for $1\ i \le \ell$, each element of P_i is less than any element of P_{i+1} (see Figure 5). So, each such possible assignment gives us a permutation from Γ. LIS of any permutation from Γ is of length ℓ because it contains one element from each P_i. LDS of any permutation of Γ is no longer than $n-2\ell+2$ because it contains at most one element from ℓ_{prefix} and at most one from ℓ_{suffix}.

▶ **Proposition 23.** *Each permutation from Γ can be split into ℓ descending subsequences in only one way. For two different permutations from Γ these ways of splitting are different.*

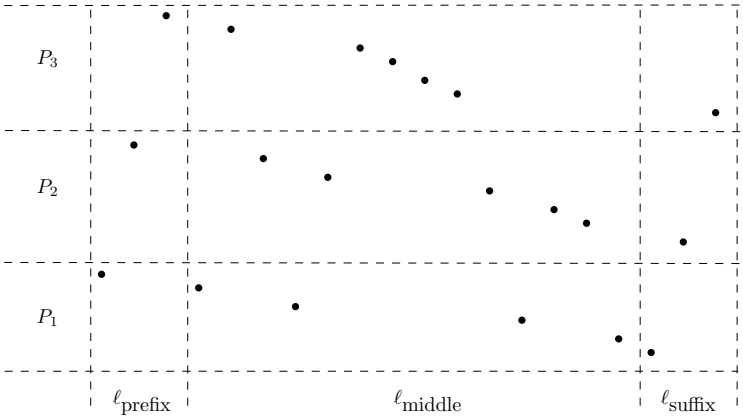


Figure 5 Example permutation $P \in \Gamma$ for $\ell = 3$ in a plane. In this figure, we have $P = (6, 13, 20, 5, 19, 12, 4, 11, 18, 17, 16, 15, 10, 3, 9, 8, 2, 1, 7, 14)$.

We now consider the algorithm computing the length of a longest k-rollercoaster. Using the permutations from Γ we create a set U of $k^{n\frac{k-3}{3k-3}}$ permutations of $\{1,\ldots,n\}$, again with the same principle behind: they should be input sequences which lead to different paths in the comparison tree associated to an algorithm computing the length of a longest

k-rollercoaster. Observe that $\log\left(k^{n\frac{k-3}{3k-3}}\right) = \Theta(n \log k)$, so this would imply the desired lower bound of $\Theta(n \log k)$ on the number of comparisons done by an algorithm to compute the length of a longest k-rollercoaster.

A permutation from U is obtained as follows. Suppose that $(3k-3)$ divides n. Split the sequence $(x_1, \ldots, x_n)$ into $\frac{n}{3k-3}$ *blocks* (contiguous subsequences) of size $3k-3$. We will assign to the elements of the i^{th} contiguous block $(x_{i(3k-3)+1}, \ldots, x_{(i+1)(3k-3)})$ distinct values from the set $\{i(3k-3)+1, \ldots, (i+1)(3k-3)\}$, as follows. In every block, use one of the permutations from Γ (with the parameter ℓ set to k) to values to the elements $x_{i(3k-3)+1}, \ldots, x_{(i+1)(3k-3)}$ of that block, and then assign values to those elements according to that ordering. In this way, we can create $|\Gamma|^{\frac{n}{3k-3}} = (k^{k-3})^{\frac{n}{3k-3}}$ permutations of $\{1, \ldots, n\}$. Observe that in every block the length of a longest decreasing subsequence is less than k. Since every block consists of strictly greater values than the previous ones, a longest decreasing subsequence of every permutation from U is less than k. A longest increasing subsequence of every element of Γ is equal to k, so a longest k-rollercoaster for every element of U is equal to $\frac{kn}{3k-3}$ and consists only of longest increasing subsequences corresponding to all the blocks glued one after the other. We can now show a result similar to Proposition 23.

▶ **Proposition 24.** *Each permutation from U can be split into $\frac{kn}{3k-3}$ descending subsequence in only one way. For two different permutations from U these ways of splitting are different.*

Having constructed the set U, we can proceed with the lower bound. Let A be a partial ordering associated with a path to some leaf of T (the comparison tree associated to the algorithm computing the length of a longest k-rollercoaster). Since the algorithm cannot distinguish between permutations following the same path, every permutation consistent with A has to give the same result. We recall the following lemma.

▶ **Lemma 25** (Lemma 3.6 in [10])**.** *Let $\leq$ be a partial ordering defined on S. The maximum length of LIS in S associated with any linear embedding of this ordering, is equal to the minimum number of decreasing subsequences relative to $\leq$ into which S can be partitioned.*

Now we can prove the following.

▶ **Lemma 26.** *Let A be partial ordering associated with the path from the root to a leaf of T. Only one permutation from U can be consistent with A.*

Proof. Consider $S \in U$ that is consistent with A, and let $D = \frac{kn}{3k-3}$ be the length of its LIS. Now let m be the minimum number of decreasing subsequences relative to the results of the comparisons made on the path A into which S can be partitioned. If $m < d$ then S is consistent with A, so we can partition S into the same decreasing subsequences, but S cannot be divided into less than than d decreasing subsequences, a contradiction. If $m > d$ then by Lemma 25 there exists a permutation S' consistent with A with the length of LIS greater than d. S' follows the same path as S in the comparison tree, but has a longer k-rollercoaster (consisting only of LIS of S') than S, a contradiction. Thus, $m = d$ for any such S.

Consider two $S_1, S_2 \in U$ consistent with A. By Proposition 24, the only partition of S_1 into d decreasing sequences is different from the only such partition of S_2 (into d decreasing sequences), so A can be consistent with only one permutation, a contradiction. ◀

Thus, each permutation from U corresponds to a distinct leaf of T, making the depth of T at least $\log |U| = \Theta(n \log k)$ as required and proving the following theorem.

▶ **Theorem 27.** *For every k satisfying $4 \leq k \leq \frac{n}{3}$, any comparison-based algorithm that computes the length of a longest k-rollercoaster in a permutation of $\{1, \ldots, n\}$ performs at least $\Omega(n \log k)$ comparisons.*

References

1 Alok Aggarwal and Maria M. Klawe. Applications of generalized matrix searching to geometric algorithms. *Discrete Applied Mathematics*, 27(1-2):3–23, 1990.

2 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric Applications of a Matrix-Searching Algorithm. *Algorithmica*, 2:195–208, 1987.

3 David Aldous and Persi Diaconis. Longest Increasing Subsequences: From Patience Sorting to the Baik-Deift-Johansson Theorem. *Bulletin of the American Mathematical Society*, 36:413–432, 1999.

4 Sergei Bespamyatnikh and Michael Segal. Enumerating longest increasing subsequences and patience sorting. *Information Processing Letters*, 76(1):7–11, 2000.

5 Therese C. Biedl, Ahmad Biniaz, Robert Cummings, Anna Lubiw, Florin Manea, Dirk Nowotka, and Jeffrey Shallit. Rollercoasters and Caterpillars. In *ICALP*, volume 107 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

6 Therese C. Biedl, Timothy M. Chan, Martin Derka, Kshitij Jain, and Anna Lubiw. Improved Bounds for Drawing Trees on Fixed Points with L-Shaped Edges. In *Graph Drawing and Network Visualization - 25th International Symposium, GD 2017, Revised Selected Papers*, volume 10692 of *Lecture Notes in Computer Science*, pages 305–317. Springer, 2017.

7 Bernard Chazelle. A Functional Approach to Data Structures and Its Use in Multidimensional Searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

8 Maxime Crochemore and Ely Porat. Fast Computation of a Longest Increasing Subsequence and Application. *Information and Computation*, 208(9):1054–1059, 2010.

9 Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

10 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

11 Hania Gajewska and Robert Endre Tarjan. Deques with Heap Order. *Information Processing Letters*, 22(4):197–200, 1986.

12 P. Gawrychowski, F. Manea, and R. Serafin. Fast and Longest Rollercoasters. *arXiv*, 2018. `arXiv:1810.07422`.

13 James W. Hunt and Thomas G. Szymanski. A Fast Algorithm for Computing Longest Common Subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

14 Sergey Kitaev. *Patterns in Permutations and Words*. Springer, 2011.

15 S. Linton, N. Ruškuc, and V. Vatter, editors. *Permutation Patterns*. London Mathematical Society Lecture Note Series, vol. 376, Cambridge, 2010.

16 Dan Romik. *The Surprising Mathematics of Longest Increasing Subsequences*. Cambridge, 2015.

17 Craige Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961.

18 J. Michael Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In David Aldous, Persi Diaconis, Joel Spencer, and J. Michael Steele, editors, *Discrete Probability and Algorithms*, pages 111–131. Springer New York, 1995.

19 Xiaoming Sun and David P. Woodruff. The Communication and Streaming Complexity of Computing the Longest Common and Increasing Subsequences. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 336–345. Society for Industrial and Applied Mathematics, 2007.

20 Alexander Tiskin. Fast Distance Multiplication of Unit-Monge Matrices. *Algorithmica*, 71(4):859–888, 2015.

Wealth Inequality and the Price of Anarchy

Kurtuluş Gemici
Department of Sociology, National University of Singapore, Singapore
kgemici@nus.edu.sg

Elias Koutsoupias
Department of Computer Science, University of Oxford, United Kingdom
elias@cs.ox.ac.uk

Barnabé Monnot
Engineering Systems & Design, Singapore University of Technology and Design, Singapore
monnot_barnabe@mymail.sutd.edu.sg

Christos H. Papadimitriou
Department of Computer Science, Columbia University, United States of America
christos@cs.columbia.edu

Georgios Piliouras
Engineering Systems & Design, Singapore University of Technology and Design, Singapore
georgios@sutd.edu.sg

Abstract

The price of anarchy quantifies the degradation of social welfare in games due to the lack of a centralized authority that can enforce the optimal outcome. It is known that, in certain games, such effects can be ameliorated via tolls or taxes. This leads to a natural, but largely unexplored, question: what is the effect of such transfers on social inequality? We study this question in nonatomic congestion games, arguably one of the most thoroughly studied settings from the perspective of the price of anarchy. We introduce a new model that incorporates the income distribution of the population and captures the income elasticity of travel time (i.e., how does loss of time translate to lost income). This allows us to argue about the equality of wealth distribution both before and after employing a mechanism. We establish that, under reasonable assumptions, tolls always increase inequality in symmetric congestion games under any reasonable metric of inequality such as the Gini index. We introduce the inequity index, a novel measure for quantifying the magnitude of these forces towards a more unbalanced wealth distribution and show it has good normative properties (robustness to scaling of income, no-regret learning). We analyze inequity both in theoretical settings (Pigou's network under various wealth distributions) as well as experimental ones (based on a large scale field experiment in Singapore). Finally, we provide an algorithm for computing optimal tolls for any point of the trade-off of relative importance of efficiency and equality. We conclude with a discussion of our findings in the context of theories of justice as developed in contemporary social sciences and present several directions for future research.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases congestion games, inequality

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.31

Acknowledgements Kurtuluş Gemici acknowledges NUS Strategic Research Grant (WBS: R-109-000-183-646) awarded to Global Production Networks Centre (GPN@NUS). Elias Koutsoupias acknowledges ERC Advanced Grant 321171 (ALGAME). Barnabé Monnot acknowledges the SUTD Presidential Graduate Fellowship. Christos Papadimitriou acknowledges NSF grant 1408635 "Algorithmic Explorations of Networks, Markets, Evolution, and the Brain". Georgios Piliouras acknowledges SUTD grant SRG ESD 2015 097, MOE AcRF Tier 2 Grant 2016-T2-1-170, NRF grant NRF2016NCR-NCR002-028 and a NRF fellowship. Barnabé Monnot and Georgios Piliouras would like to thank the other members of the National Science Experiment team at SUTD: Garvit Bansal, Francisco Benita, Sarah Nadiawati, Hugh Tay Keng Liang, Nils Ole Tippenhauer, Bige Tunçer,

Darshan Virupashka, Erik Wilhelm and Yuren Zhou. The National Science Experiment is supported by the Singapore National Research Foundation (NRF), Grant RGNRF1402.

1 Introduction

Inequality in wealth and income have been rampant worldwide in the past four decades [32,39], considered by many the scourge of modern societies. Economic analysis, on the other hand, traditionally focuses on efficiency, that is to say, Pareto optimality of the allocation. Whether, and to what extent, efficiency and equality are at loggerheads has been debated in economics, and the verdict appears to depend on context and assumptions.

Modern societies also give rise to a plethora of strategic scenarios, in which the behavior of one agent affects the others, and the outcome of which ultimately affects the agents' overall well-being. In game theory, we study the inefficiency of these strategic situations through the so-called *price of anarchy,* the relative efficiency of the game's Nash equilibria over the social optimum [28]. For congestion games in particular, it is known that the price of anarchy can be combatted through the introduction of *tolls* which enforce the optimal outcome as equilibrium, see [15, 18] among an extensive literature. However, the effect that tolls may have on the level of inequality in the society does not appear to have been addressed in the literature.

The present paper is a first attempt to articulate and study this issue. We consider games (here only congestion games) in which the agents' utility and behavior depend explicitly on their income or wealth, and study the effect the game's equilibria have on inequality.

Example: Transportation in Singapore, seen as a congestion game with tolls, has a price of anarchy that is close to one [29]. The main arteries are almost never clogged, and public transportation is accessible and runs smoothly. This is the result of bold policy decisions: car ownership in Singapore is significantly taxed, and dynamically adaptive tolls are in place. Interestingly, transportation delays seem to be a decreasing function of income (see Section 7 on data). This is no accident: In this paper we show that there is an inherent tension between efficiency and equality in the context of congestion games.

We are interested in the ways in which optimal (or more generally efficiency-enhancing) mechanisms affect inequality. Inequality is measured in many ways, but perhaps most often through what is known as the *Gini coefficient (or Gini index).* Informally (see Section 3 for the definition), the Gini coefficient of a distribution of income or wealth is *twice* the area between the 45^o line and the normalized convex cumulative wealth/income curve (see Figure 1). That is, we compute the cumulative income/wealth $Q(y)$ of the lowest y fraction of the population for all $0 \leq y \leq 1$, we normalize it so that $Q(1) = 1$, and then we integrate $y - Q(y)$ from 0 to 1. At total equality the Gini index is zero, while at total inequality (i.e., when the emperor owns everything) it is one. In 2015, the income Gini in OECD countries ranged from the .20s (Northern Europe) to the .40s and .50s (USA and East Asia).

Our contributions. We study nonatomic congestion games with tolls, where we introduce a new model that incorporates the wealth distribution of the population and captures the income elasticity of travel time (i.e., how loss of time translates to lost income). This allows us to argue about the equality of wealth distribution both before and after participating in a mechanism (with or without tolls). The basics of our modeling are thus: We consider a

continuum of agents, each agent of a type $x > 0$ standing for their income.[1] We assume that the distribution of types is known. Suppose these agents engage in a game Γ and that, at equilibrium, type x receives a cost c_x. This cost is expressed in the same units as income, dollars, say; after incorporating the losses due to time spent in traffic in dollars as well as any possible costs due to tolls/taxes. As a result, the agent's total wealth becomes $x' = x - \alpha c_x$, where α is a small constant standing for the importance of the game under consideration to an individual's well-being. In Section 4 we establish a broad qualitative result, the Inequity Theorem (Theorem 2), showing that tolls always increase inequality in symmetric congestion games under the most classic inequality measure, the Gini coefficient. In fact, participating in a toll-free symmetric congestion game has no impact on the Gini, whereas optimal tolls on the other hand have a negative impact on the Gini. Theorem 6 broadly expands Theorem 2 to *any* inequality measure that satisfies four fundamental axioms: *invariance to population scaling, anonymity, invariance to income scaling and the transfer principle* (see Section 3). These measures include, besides Gini, some of the most widely employed indices, such as [40] or [3].

At a technical level, the proof of the Inequity Theorem combines game-theoretic properties of congestion games with tolls and the axioms of inequality measures. In order to argue that the Gini of the final income distribution is worse than that of the original, it suffices to argue that the Lorenz curve of the original distribution (see Figure 1) dominates the latter. Lorenz curve domination is established via the combination of Lemmas 3, 4, 5, implying Theorem 2. In fact, Lorenz curve domination suffices to argue something stronger. Any inequality measure satisfying the four axioms is also consistent with the Lorenz domination order, yielding Theorem 6.

In Section 5 we introduce the inequity index, a novel measure for quantifying the magnitude of these forces towards a more unbalanced wealth distribution. Let q be the initial income distribution of the population of agents under consideration, let $G(q)$ be its Gini coefficient, and suppose that $\hat{q}$ is the distribution of the income *after* each x becomes $x - \alpha \cdot c_x$ (that is, after the game has been played). We are interested in the way the game affects the Gini coefficient; we express this, informally, as the coefficient of α in $G(\hat{q}) - G(q)$, ignoring terms that are $o(\alpha)$; in other words, we are interested in the *derivative* of $G(q)$ with respect to α. We call this quantity the *inequity* of the game. We show that from a theoretical perspective it has attractive properties. Specifically it is robust to scaling of income (Theorem 9) and it remains unaffected if instead of immediate equilibration we assume that all agents apply regret-minimizing algorithms (Theorem 10).

We analyze inequity both in theoretical settings (Section 6) as well as experimental ones (Section 7). Specifically, these effects become apparent already in the well-trodden Pigou's network [31]. This network has two parallel links, one with constant delay function 1, and another with delay function x (that is, a delay proportional to the percentage of agents that take this option). Its price of anarchy is $\frac{4}{3}$, and the inequity turns out to be zero. It is well-known that the price of anarchy, *in the case of equal incomes,* can be rendered to one by adding tolls, and it is not hard to see that the same can be done for any income distribution [15] – *but then the inequity becomes substantial.* If tolls decrease, we have a full-fledged trade-off between inequity and price of anarchy. In Theorem 11 we calculate the precise price of anarchy to inequity trade-off of any variant of Pigou's network with income distributions of the form y^β.

[1] Or wealth; we write "income" henceforth in this paper, but "wealth" would also be appropriate everywhere.

In Section 7 we perform data analytics on a dataset capturing the routing behavior of tens of thousands of Singaporean students. This dataset captures the movement of each individual at a high frequency (one new datapoint per individual every 13 seconds) and allows us to distinguish between different modes of transportation (walking, bus, train, car). We can pinpoint each individual's home location which allows us to compute estimates about their wealth. Given the level of data granularity, we can control for different parameters and identify a statistically significant increased commute time for the lower-income students, which corroborates our theoretical analysis. Interestingly, the Singapore case also points out some of the successful policies (e.g., polycentric urban development model) that can be implemented to alleviate the trade-off between efficiency and equality. Finally, we provide an algorithm for computing optimum tolls for any point of the trade-off of relative importance of efficiency and equality for symmetric networks on parallel links. We also present inequity results in asymmetric settings, which prompt several open questions. The full online text [22] provides any missing supplementary material and analysis.

2 Related work

Price of anarchy was introduced in congestion games [28], leading to a long sequence of influential papers in the area [13, 16, 20, 37, 38]. A similarly long line of research on tolls is existing in the AGT community, starting with [15, 18]. Recently, [25] have found efficient algorithms to compute tolls that minimize latency where not all edges can be tolled, putting their work close to the current situation in many cities. [6] have shown that taxes depending on the congestion level of a resource for weighted agents increase the efficiency of congestion games with polynomial latency functions. [5] have proved that without knowing the latency functions and using only tolls and an efficient number of queries to an oracle, target equilibrium flows can be reached. The data analytics, experimental part of our paper shows, perhaps unsurprisingly, that the use of public transportation plays a critical, but not well-understood, role in the functioning of a traffic network. [19] introduced a model of congestion games with buses, and hopefully more research will follow along these lines.

Given the proliferation of the usage of algorithms in all aspects of our lives (from suggesting Airbnb hosts to identifying convicts eligible for early parole), the theoretical computer science community has recently focused on understanding issues of fairness, equality and justice. Surprisingly, the intersection of price of anarchy, i.e., efficiency in games, and fairness has not been explored so far. On a related tangent, the issues of altruism and efficiency have been tackled, e.g. by [9] and [10]. [11, 12] have recently used the Gini coefficient over the probabilities of the agent winning probabilities as an inequality measure of different mechanisms and design mechanisms with such good properties. Although syntactically similar, these works do not model wealth distributions nor do they examine the differential effects of mechanisms to equality, which is our focus.

We continue the line of work of [29] where price of anarchy in congestion games is studied using field experiments with thousands of participants. In this paper, our data analytics corroborate our theoretical insights and give rise to novel questions for future research.

3 Model Description

We describe a game-theoretic model where a continuum of agents participates in a traffic congestion game with tolls. The total disutility for each agent depends both on their traffic-induced latency as well as on the tolls, whose effects are experienced differentially based on each agent's income level.

Congestion game. A *symmetric congestion game with type-specific costs* consists of a finite set E of edges, and a finite subset of 2^E called the set of *paths* $\mathcal{P}$, common to all types. We shall only deal with *network* congestion games, where the set of paths consists of all possible paths between two nodes s and t in a graph with edge set E.

Income. We have a continuum of *types* which lie in $[0,1]$. Type x has income $q(x)$, where q is the *quantile function* of the income of a population of agents – that is, $|z : q(z) \leq q(x)| = x$, where $|\cdot|$ is the Lebesgue measure. We shall further assume that $q(0) > 0$ and q is measurable and nondecreasing. Typically, we will assume a continuum of types and a strictly increasing, continuous q. In this case, if we treat income as random variable, then q expresses the inverse of its cumulative distribution function.

Flow. A *flow* $F : [0,1] \to \mathcal{P}$ is a mapping from types to paths; we shall only need to consider *finitary* flows, that is, flows F which divide $[0,1]$ into finitely many intervals, and map the interiors of those intervals to one path in $\mathcal{P}$; that is, F is specified by a finite number of reals $a_0 = 0 < a_1 < a_2 < \cdots < a_k = 1$ such that $F(b) = F(c)$ for all i and $b, c \in (a_i, a_{i+1}]$

Edge cost. Our main result, the Inequity Theorem, holds under general conditions on the edge cost functions. For simplicity of exposition, we look at a specific case that has natural properties and leave to the full online text [22] a discussion on more general results for larger classes of edge cost functions. Each agent x using edge e experiences the edge cost $f_e(q, z, \tau_e)$, where q is the agent's income, z is the level of congestion on edge e and τ_e is the fixed toll paid by the agent.

We are interested in the following edge cost function:

$$f_e(q, z, \tau_e) = \frac{\tau_e}{q} + \ell_e(z)$$

The *path cost* for P is $\sum_{e \in P} f_e(q, z, \tau_e)$.

There is an extensive discussion in the transportation literature of the true cost of transportation to the traveler and the value of time, see [2, 8] for some of the most recent papers, with dozens of references therein. This field has established and studied the *income elasticity* of the value of (travel) time (informally, the precise nature of the formula $\frac{\tau_e}{q}$ above) and validated and measured it through extensive surveys and other studies over three decades. The upshot is that the cross-sectional elasticity (that is, the elasticity with regressive corrections across causal parameters such as having children and living in the capital) is constant across long periods of time, and that the precise relationship seems to be $\frac{\tau_e}{q^\beta}$ where $\beta \leq 1$ is conventionally taken to be one, even though certain countries, such as the UK, use value 0.8.

Agent cost. Let F be a flow. The *congestion* of this flow, c^F, is a function mapping E to the nonnegative reals, where $c^F(e) = |\{x : e \in F(x)\}|$, where $|\cdot|$ denotes the Lebesgue measure. The *agent cost* under flow F to an agent of type x is some function of its income and path cost.

The model allows for some degree of flexibility when designing the overall cost of the agents. We focus our attention to the following agent cost:

$$\text{cost}^F(x) = q(x) \cdot \sum_{e \in F(x)} f_e(q(x), c^F(e), \tau_e) \tag{AC}$$

Edge costs are scaled by the income of the agent and thus the agent cost is given in the units of the toll, i.e. money.

For $f_e(q, z, \tau_e) = \frac{\tau_e}{q} + \ell_e(z)$, we have

$$\text{cost}^F(x) = \sum_{e \in F(x)} \tau_e + q(x) \cdot \ell_e(z) \tag{CAN}$$

We call this agent cost function *canonical* and show further that it is a natural choice with good properties (Section 5).

Nash equilibrium. We say that a flow F is a *Nash equilibrium* in our model if for all types x and for all paths $P \in \mathcal{P}$

$$\text{cost}^F(x) \leq q(x) \cdot \sum_{e \in P} f_e(q(x), c^F(e), \tau_e) \tag{NE}$$

that is, if no type x would be better off by deviating to another path $P \in \mathcal{P}$.

In the following, we define q to be the income distribution of agents before playing the game. With our definition of agent costs, one can study q_0, the income distribution after playing the game *without tolls*, where $f_e(q, z, \tau_e) = \ell_e(z)$. The move from q to q_0 is defined as **the impact of travel**, the variation that is due only to the presence of a game. When tolls are levied, we have a second move, from q_0 to $\hat{q}$, defined as **the impact of tolls**. We will be mostly concerned with the latter impact.

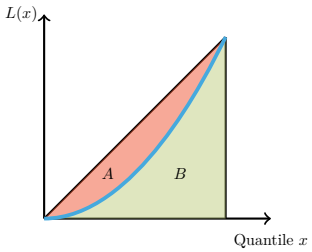


Figure 1 The Lorenz curve is plotted in blue. The green area is $B = \int_0^1 L(t)dt$. The Gini coefficient is then $G = 1 - 2B = 2A$.

Gini coefficient. The Gini coefficient [23] is a central measure of inequality.

▶ **Definition 1.** *The Gini coefficient of income distribution q is given by*

$$G(q) = 1 - 2\int_0^1 L(t)dt$$

where $L(t)$ is the Lorenz curve, or the fraction of total income held by individuals under and at quantile x.

$$L(t) = \frac{1}{\mu}\int_0^t q(x)dy = \frac{1}{\mu}Q(t) \tag{LC}$$

for $Q(t) = \int_0^t q(x)dx$, the cumulative income up to quantile t. We show in Figure 1 the relationship between the Lorenz curve and the Gini coefficient.

A Gini coefficient equal to zero corresponds to perfect equality (everyone has the same income), whereas a Gini coefficient of one corresponds to maximal inequality (e.g., one person has all the income). The Gini coefficient has several desirable properties such as:

- **Scale independence.** The Gini coefficient does not change after rescaling incomes (e.g. change of units/currency).

- **Population independence.** It does not depend on the size of the population.
- **Anonymity.** It does not depend on the identity of the rich/poor individuals.
- **Transfer principle.** If income (less than the difference[2]) is transferred from a rich person to a poor person the resulting distribution is more equal (i.e., the Gini decreases).

Our motivating problem. We consider how Nash equilibrium flow F affects the incomes of the population. In particular, we assume that the income of type x changes from $q(x)$ to $q(x) - \alpha \cdot \text{cost}^F(x)$ for some (intuitively small) $\alpha > 0$. We call the resulting income distribution $\hat{q}(x)$. Notice that, in general, $\hat{q}(x)$ may be different from $q(x) - \alpha \cdot \text{cost}^F(x)$, since the cost of F may rearrange the order of types (recall that distributions such as $q(x)$ are assumed to be nondecreasing). As we shall see in the Inequity Theorem proof of Section 4, this turns out to never be the case and moreover the inequality increases as a result.

4 The Negative Impact of Tolls on Inequality

4.1 The Inequity theorem

Tolls can be used in congestion games so as to induce socially optimal flows (from the perspective of total cost) as Nash equilibrium [15, 18]. We next prove a general theorem showing that tolls always exacerbate societal inequality. So, in a sense to achieve optimality from the perspective of social welfare we have to pay a hidden cost in terms of fairness.

▶ **Theorem 2** (The Inequity Theorem). *In any Nash equilibrium of any symmetric congestion game with type-specific costs, any set of positive edge tolls τ_e increases the inequality of the population. More specifically,*

- *The impact of travel is zero: the Gini coefficient of the ex ante income distribution q is equal to the Gini coefficient of the toll-free income distribution q_0, $G(q_0) = G(q)$.*
- *The impact of tolls is nonpositive: the Gini coefficient of the ex ante income distribution is lower than (or equal to) the Gini coefficient of the ex post income distribution $\hat{q} = q - \alpha \cdot \text{cost}^F$, or $G(\hat{q}) \geq G(q) = G(q_0)$.*

Additionally, if the quantile distribution of income is increasing and edge cost functions are decreasing in income, the Gini coefficient increases strictly.

Proof Sketch. First, we show that the impact of travel is null. In the toll-free version of the game, the Nash equilibrium is the usual Wardrop equilibrium and all agents incur the same cost C, regardless of their route choice. This implies that all agents lose the same share of income exactly equal to $\alpha \cdot C$. Since the Gini coefficient and all inequality measures we will be concerned with are scale invariant, the inequality is not affected by the impact of travel.

The proof of the impact of tolls is done in three steps. First, we show that if two income distributions with equal means cross at one point, one has a higher Gini coefficient than the other (Lemma 3). This is equivalent to the transfer principle, or Pigou-Dalton principle of income inequality measures. Second, we show that when a distribution is obtained by decreasing proportionally less the higher incomes than the lower incomes – in other words, a regressive tax – then the resulting distribution has a higher Gini coefficient than the original one, i.e., is more unequal (Lemma 4). Third, we show that under equilibrium in the game, players with higher incomes have a lower path cost than players with lower incomes (Lemma 5). Finally, Theorem 2 is obtained as a corollary of the three lemmas.

[2] If the income transfer is less than the difference of their incomes, the ordering of the wealth of the users does not change.

▶ **Lemma 3.** *Suppose q and $\hat{q}$ are two income distributions (represented by their quantile functions) of equal means, i.e., $\mu = \int_0^1 q(x)dx = \int_0^1 \hat{q}(x)dx = \hat{\mu}$. If there exists x^* such that $\hat{q}(x) \leq q(x), \forall x \leq x^*$, and $\hat{q}(x) \geq q(x)$ otherwise, then $G(q) \leq G(\hat{q})$.*

▶ **Lemma 4.** *Suppose two income distributions (represented by their quantile functions) q and $\hat{q}$ are such that $\hat{q}(x) = \beta(x) \cdot q(x)$ and $1 \geq \beta(y) \geq \beta(z) > 0$ for $y \geq z$[3] then $G(q) \leq G(\hat{q})$.*

▶ **Lemma 5.** *Let $0 \leq x \leq y \leq 1$ and F be an equilibrium flow. If agent costs are given by the path cost $\sum_{e \in F(x)} \frac{\tau_e}{q} + \ell_e(z)$ then $\text{cost}^F(x) \geq \text{cost}^F(y)$.*

The resulting income distribution in the game is given by $\hat{q}(x) = q(x) \cdot (1 - \alpha \cdot \text{cost}^{\bar{F}}(x))$. At equilibrium costs decrease with income (Lemma 5). Thus, distribution q Lorenz-dominates distribution $\hat{q}$ (Lemma 3 and 4), i.e., the Lorenz curve of q is always above that of $\hat{q}$. This implies that the inequality in $\hat{q}$ is greater than in q. ◀

Theorem 2 can actually be generalized. Lorenz domination, a partial order, is respected by *all* inequality coefficients that satisfy the same four axioms as the Gini coefficient, yielding the following more general version of the Inequity Theorem.

▶ **Theorem 6.** *For any income inequality measure satisfying the axioms of invariance to population scaling, anonymity, invariance to income scaling and the transfer principle, the Inequity Theorem holds and inequality increases as tolls are levied on the players.*

The remaining of the paper focuses on the case of the Gini coefficient.

4.2 Computing the efficiency-equality trade-off

In a *parallel links network* serving a population with a known income distribution, the routing and tolls that optimize any desired trade-off between efficiency and equality can be computed via dynamic programming.

4.2.1 The model

Because of the computational nature of this section, and for the sake of simplicity, we will stick to a simplified, discrete model. Very few of these simplifications are crucial. We assume a population whose income is presented in n *quantiles* $q_1, \ldots, q_n$, where q_1 stands for the average income of the lowest $\frac{1}{n}$ of the population – if n were 100, these would be the income percentiles.

We have K parallel links – we assume that K is fixed. Each link e has a delay function $f_e(x)$ which we assume for simplicity to be piecewise constant with increments at values of x that are multiples of $\frac{1}{n}$ (so that each link accommodates full quantiles), and that the delays have integer values in the set $[D]$, where D is the maximum delay. Evidently, the problem is one of allocating each quantile to a link, and imposing appropriate tolls. It is easy to see that at equilibrium each link will be assigned a *contiguous* set of quantiles.

[3] I.e., $\hat{q}$ is obtained from q by a transformation that reduces lower incomes relatively more than higher incomes. Income *order is preserved* and $\hat{\mu} \leq \mu$.

4.2.2 The objective

We seek to optimize a trade-off between efficiency and equality, that is to say, a weighted sum of total delay and the Gini coefficient resulting from this game, say of the form "minimize total delay + λ times the Gini after the game," where $\lambda > 0$ is the relative importance of equality over efficiency. It is important to note that the Gini coefficient before the game is in this case captured by (ignoring additive terms and a factor of $\frac{-2}{n}$)

$$\frac{\sum_{i=1}^{n}(n+1-i)q_i}{\sum_{i=1}^{n} q_i}.$$

This is on account of the fact that, in the sum that approximates the double integral in Equation (LC), the lowest quantile appears n times, the second lowest $n-1$, etc.

After the game imputes a cost to the ith quantile, the Gini coefficient is captured by

$$\cdot\frac{\sum_{i=1}^{n}(n+1-i)(q_i-\delta_i)}{\sum_{i=1}^{n}(q_i-\delta_i)},$$

where $\delta_i = q_i d_i + \tau_i$ is the cost of the equilibrium to the ith percentile, and d_i is the delay and τ_i is the toll incurred by the ith quantile. Now, since it is reasonable to assume that $\delta_i << q_i$, this quantity can be adequately represented by its numerator divided by the sum of the q_i's[4]. Thus, omitting constant terms (it is important to recall that the q_i's *are* constant), both additive and multiplicative, we conclude that what is minimized is a linear function of the delays d_i and the tolls τ_i. Adding to them the total delay[5], we conclude that the objective is of the form

$$\min_{\text{allocation of quantiles to links}} \sum_{i=1}^{n}(\alpha_i d_i + \beta_i \tau_i),$$

for some known positive parameters α_i, β_i.

4.2.3 The algorithm

The algorithm is dynamic programming; namely, we compute the quantity cost$[S, m, d]$ with $S \subseteq [K]$, $m \leq n$, and $d \leq D$, which is the smallest value of the objective that can be achieved by allocating the *lowest* m percentiles to the set S of links (in the optimum order) with the (largest) delay of the m-th percentile equal to d. The algorithm is presented in Algorithm 1.

By $\tau(d, d', r, m-r)$ we denote the toll required to equalize, for the $m-r$th quantile, the delay d' with the greater delay d. In conclusion (here $D^* \leq n$ is the number of different values of the delay in the network):

▶ **Theorem 7.** *The optimum trade-off between total delay and the Gini coefficient can be computed in time $O(nD^*)$*

But of course, the O-notation hides the constant $K^2 2^K$.

[4] For more accuracy, the computed value of $\sum_i \delta_i$, can be plugged in here and repeat the computation.
[5] Note that even the *total weighted delay* $\sum_i q_i d_i$ can be similarly accommodated as part of the trade-off.

Algorithm 1: A dynamic programming algorithm to compute the trade-off between efficiency and equality.

Data: Calculate the values $\text{cost}[\{e\}, m, d]$ for all links $e, m \in [n], d \in [D]$
begin
  for $s \leftarrow 2$ **to** K **do**
    for *All sets* $S \subseteq L$ *with* $|S| = K$ **do**
      for $m \leftarrow 1$ **to** n **do**
        for $d \leftarrow 1$ **to** D **do**
          $\text{cost}[S, m, d] = \min_{e \in S,\, r < m:\, \ell_e(r) = d;\, d' \leq d} \text{cost}[S - \{e\}, m - r, d']$
            $+ \sum_{j=m-r+1}^{m} (\alpha_j d' + \beta_j t(d, d', r, m - r))$
        end
      end
    end
  end
end

4.3 The asymmetric case

In the case of multiple source-destination pairs the inequality within each set of players in any single commodity is again worsened as a result of tolls. Such a statement is not obtainable for the society as a whole. In the full online text [22], we show how to create, admittedly contrived, counterexamples where despite the fact that within each subpopulation the inequality worsens the population as a whole becomes more equal (e.g. the rich and poor use different subnetworks and only the rich get taxed). We believe that such adversarial counterexamples may be circumvented by imposing more realistic models, and pose this as one of the possible directions for future work.

5 The Inequity Index

The Inequity Theorem shows that under general conditions of the cost functions, the income inequality between agents increases after tolls are levied. In this section, we quantify this deterioration of equality by introducing a new metric. We have captured the importance of the game costs to the agents' income by a parameter $\alpha > 0$, intuitively small. The inequity (index) is defined as the derivative of the Gini coefficient as α goes to zero.

▶ **Definition 8.** *Let Γ be a nonatomic symmetric congestion game. Agents have an initial ex ante distribution $(q(x))_{x \in [0,1]}$ and incur a cost $\text{cost}^F(x)$ under flow F. Let $q_\alpha(x) = q(x) - \alpha \cdot \text{cost}^F(x)$ be the ex post income distribution for some $\alpha > 0$. The inequity of Γ is defined as*

$$I(\Gamma) = \lim_{\alpha \to 0^+} \frac{G(q_\alpha) - G(q)}{\alpha}.$$

Note that this notion is well-defined. The Gini coefficient for distribution q_α is given by

$$G(q_\alpha) = 1 - 2\frac{\int_0^1 \int_0^x (q(t) - \alpha \cdot \text{cost}^F(t))dtdx}{\int_0^1 (q(x) - \alpha \cdot \text{cost}^F(x))dx} = 1 - 2\frac{\int_0^1 Q(x)dx - \alpha \int_0^1 \int_0^x \text{cost}^F(t)dtdx}{\mu - \alpha \cdot SC}$$

where μ is the total income of distribution q and SC is the social cost. This function is indeed differentiable with respect to α, provided the obvious requirement of $\mu > 0$ is satisfied.

5.1 Scale invariance of the inequity index

The Inequity Theorem implies that the inequity is always nonnegative. For the rest of the paper we will focus on the canonical cost functions (CAN). As a reminder, the cost of agent x in edge e is

$$q(x) \cdot f_e(q(x), c^F(e), \tau_e) = q(x) \cdot \ell_e(c^F(e)) + \tau_e \,.$$

The canonical cost functions, besides having strong experimental justification [2,8] provide also significant advantages in the theoretical study of inequity. Specifically, the inequity index is invariant under scaling of the population incomes.

▶ **Theorem 9** (Robustness under scaling of income). *Assume agent cost functions are in canonical form* (CAN) *in a game* Γ. *Then the inequity is scale invariant: if all incomes are scaled by a constant* $\lambda > 0$ *and optimal tolls are used in the resulting game* Γ_λ, *then* $I(\Gamma) = I(\Gamma_\lambda)$.

Proof Sketch. To give the main idea of the proof, we introduce a scaling parameter $\lambda > 0$. This parameter can be understood as a redenomination of the value of money in the game, for both income and the tolls, where one unit of the "new" currency is effectively as valuable as λ units of the previous currency. As such, this does not affect the strategic content of the game (no change of actions) nor the costs, by the scale invariance property of the Gini coefficient. ◀

5.2 No-regret learning

So far we have looked at the inequity index in the context of agents playing the Nash Equilibrium of the routing game. However, it is possible to relax this assumption and let agents implement a no-regret strategy of their own.

Let $F_1, F_2, \ldots$ be a sequence of flows obtained from agents repeatedly playing the game. Agent x is implementing a no-regret algorithm if it has vanishing regret, i.e.

$$R(T) = \frac{1}{T}\sum_{i=1}^{T} cost^{F_i}(x) - \min_{p\in\mathcal{P}} \frac{1}{T}\sum_{i=1}^{T}\sum_{e\in p} f_e(q(x), c^{F_i}(e), \tau_e) \to 0 \text{ as } T \to \infty$$

We also call an ϵ-approximate Nash Equilibrium a flow F_ϵ such that

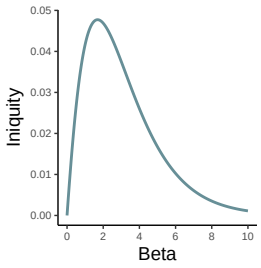
$$\int_0^1 \text{cost}^{F_\epsilon}(x)dx - \min_{p\in\mathcal{P}} \sum_{e\in p} f_e(q(x), c^{F_\epsilon(e)}, \tau_e) \le \epsilon \,.$$

Following the results in [7], we can show that under regret minimizing agents, the flow converges to that of an approximate equilibrium under the assumption of a finite number of wealth/income levels $w_1, \ldots, w_K$. This assumption is rather realistic since in practice there can only be a finite number of income levels. Also, any continuous distributions over incomes can be approximated to arbitrary high accuracy by a distribution of finite but large enough support.

▶ **Theorem 10** (Robustness under no-regret learning). *Given a finite number of income levels, the inequity index is uniquely defined under the assumption of no-regret learning agents. Specifically, if all agents follow a no-regret algorithm, we have*

$$\lim_{\alpha\to 0;\alpha>0} \lim_{T\to\infty} \frac{\frac{1}{T}\sum_{t=1}^{T} G(\hat{q}^t) - G(q)}{\alpha} = I(\Gamma)$$

where $\hat{q}^t$ *is the ex post income distribution of the* t*-th instance of the game.*



(a)

β	0	1	2	3	4
G	0	$\frac{1}{24}$	$\frac{3}{64}$	$\frac{3}{80}$	$\frac{5}{192}$

(b)

Figure 2 a. The Inequity index as a function of the income coefficient β. It is 0 when there is no inequality ($\beta = 0$) because tolls have the same effect on everybody, and rises as the inequality increases. At some point, $\beta \approx 1.688$, the Inequity index starts decreasing, because the toll $(\beta + 1)2^{-(\beta+1)}$ becomes small and has little effect on the inequality index. **b.** Values of the inequity for different β.

Proof Sketch. In the first step of the proof, we show that the symmetric game of type-specific costs Γ reduces to an asymmetric congestion game $\hat{\Gamma}$. In the second step, results on the behavior of no-regret dynamics in asymmetric congestion games [7] imply the robustness of the inequity index. This is due to the Gini coefficient being a bounded and continuous function. ◀

6 Computing the Inequity in Pigou

To illustrate the interplay between wealth or income and congestion games, we consider the well-studied Pigou network, which consists of two parallel links with latency functions $\ell_u(r) = 1$ and $\ell_d(r) = r$. Assume that this transportation network is used by a population of (normalized) size 1 and with wealth or income function $q(x) = x^\beta$, for some nonnegative parameter β.

The perceived cost for quantile x is $\text{cost}(x) = \ell_e(c(e)) \cdot q(x) + \tau_e$, where $e = e(x)$ is the edge used by the quantile, $c(e)$ is the flow through link e, and τ_e the price of link e. It is not hard to argue that at equilibrium the c_u fraction of the population that uses the constant cost link is the poorest c_u part of the population. We will assume further that *the social designer selects tolls to minimize the actual latency* on the network,[6] and that, without loss of generality, the price of the constant cost edge is set to $\tau_u = 0$, while $\tau = \tau_d$ is the optimal price on the variable cost edge.

By continuity, *at equilibrium the perceived cost of quantile c_u must be the same in both links*, from which we get $\tau = q(c_u)c_u$. We want to investigate the effects of price τ on the Gini coefficient.

Let $\hat{q}(x) = q(x) - \alpha \cdot q(x) \cdot \text{cost}(x)$ be the perceived income when we take into account the effects of perceived latency into the actual income, where α indicates the importance of transportation. We are interested in first order effects, so we will always assume that α is very small and that in fact it tends to 0. Let's define as $G(\alpha, \tau)$ the inequality coefficient when we take into account the effects on the income of the transportation cost, assuming that the social designer selects toll τ.

[6] There are reasonable alternatives for the social planner, such as minimizing the social cost, that we do not explore in this work.

We can now compute directly the Gini coefficient. For the Pigou network the optimal switching point is $c_u = 1/2$. For income distribution $q(x) = x^\beta$, this optimal switching point corresponds to toll $\tau = 2^{-(\beta+1)}$. Since at equilibrium, $\tau = q(c_u)c_u = c_u^{\beta+1} = 2^{-(\beta+1)}$, we have

$$G(\alpha, (\beta+1)2^{-(\beta+1)}) = \frac{\beta}{\beta+2} + \frac{\beta(\beta+1)}{(\beta+2)2^{\beta+3}}\alpha + O\left(\alpha^2\right).$$

The last expression comes from the Maclaurin expansion of the function, from which we derive the following Theorem.

▶ **Theorem 11.** *For the Pigou network with two links and latency functions* 1 *and* x*, and for a population with income distribution* $q(x) = x^\beta$*, when tolls are selected to minimize the actual latency, the toll at Nash (Wardrop) equilibrium is* $\tau = q(c_u)c_u = 2^{-(\beta+1)}$*, and the Inequity index is* $I(\Gamma) = \frac{dG(0)}{d\alpha} = \frac{\beta(\beta+1)}{(\beta+2)2^{\beta+3}}$.

The values of the inequity as a function of the income coefficient β are shown in Figure 2. The maximum occurs when the income coefficient β is close to 2 (actually when $\beta \approx 1.688$), which means that real-life income distributions have almost the maximum Inequity index.

7 Tolls and Inequality: Empirical Findings

We use detailed transportation data gathered through Singapore's National Science Experiment (NSE) to test how income inequality affects the distribution of transportation delays in a representative sample of students [29, 30, 43]. Although Singapore is the third most densely populated country in the world, the modern infrastructure, cost of private cars, and significant tolls in Singapore minimize congestion on the roads. We examine whether this gain in efficiency incurs costs in terms of income inequality, as predicted by the theoretical results in this paper. The NSE dataset enables us to accurately split student trips in the morning – the time of the day when tolls are most onerous – by the transportation mode (bus, car, walk, and train) [42]. We then combine the travel data with a dataset on property prices to assess the relationship between income and the average duration and average distance of trips by transportation mode.

By relying on the sociological literature pertaining to income inequality and Singapore's urban development, we divide the students into 9 wealth brackets based on residence. We then conservatively classify these brackets as low-income, middle-income, and high-income groups [1, 14, 17, 24, 26, 36, 41]. The differences in the means of trip distance and duration are statistically significant among these three groups; and, they lend strong support to the predictions of the Inequity Theorem. When one compares low-income and high-income groups, there is a notable increase in car usage and decrease in the use of walking and public transportation. Because cars are much faster than using bus and walking, the use of cars is associated with a sizable difference in the average duration that students spend in traveling to school (Figure 3).

Although students from high-income groups travel a longer distance compared to middle-income groups, this translates into minor differences in travel duration. The opposite is the case when we compare low-income and middle-income groups. These students experience on average 7 to 5 minutes delay compared to middle-income groups, despite the fact that the distance they travel is roughly comparable to high-income groups (Figure 3). Thus, the Singaporean case – which is an ideal setting to examine the relationship between inequality and transportation delays – offers positive evidence on the Inequity Theorem. It also provides some lessons on the policies that can be implemented to mitigate the trade-off between efficiency and inequality, as we discuss in the full text [22].

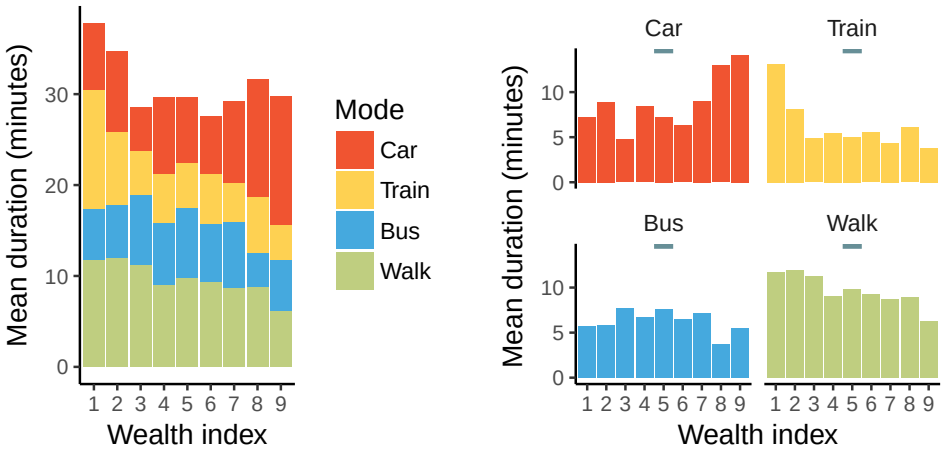


Figure 3 The average trip duration per wealth bracket is presented in the two figures above. *Left:* Average duration in each transport mode. It is notable that brackets 1 and 2 have respectively 7 and 5 minutes more travel time on average than the other brackets. *Right:* The left plot is split along the transport mode, to show the relative durations spent in each mode across wealth brackets. Given that the least-affluent groups spend much more time on the roads compared to middle-income groups, there is a quasi-monotonic increase in the use of car as wealth increases, while we observe that the uses of walking and public transportation decrease as wealth increases.

8 Discussion

The Inequity Theorem raises important questions pertaining to distributive justice and the efficiency of decentralized decision-making mechanisms such as markets [4, 21, 27, 33–35]. Namely, if efficiency can be obtained through purposeful intervention but only at a price of increase in inequality, what are the implications of this trade-off for the organization of markets, industries, and society in general? We offer three potential avenues of research:

- What is the opportunity cost of inequity?
- How does inequity affect cooperation among members of society?
- How does inequity affect the formation of groups and thus cooperation between different groups?

We believe that these questions hold the promise of opening up new lines of research for algorithmic game theory. We hope that future work in this area will shed light on important but largely unexplored issues about the interplay between efficiency and optimality in a wide range of economic scenarios and mechanisms.

References

1 Tilak Abeysinghe and Keen Meng Choy. The Aggregate Consumption Puzzle in Singapore. *Journal of Asian Economics*, 15(3):563–578, June 2004. doi:10.1016/j.asieco.2004.05.007.

2 Pedro AL Abrantes and Mark R Wardman. Meta-analysis of UK values of travel time: An update. *Transportation Research Part A: Policy and Practice*, 45(1):1–17, 2011.

3 Anthony B Atkinson et al. On the measurement of inequality. *Journal of economic theory*, 2(3):244–263, 1970.

4 Jens Beckert. *Beyond the Market: The Social Foundations of Economic Efficiency*. Princeton University Press, Princeton, NJ, 2002.

5 Umang Bhaskar, Katrina Ligett, Leonard J Schulman, and Chaitanya Swamy. Achieving target equilibria in network routing games without knowing the latency functions. *Games and Economic Behavior*, 2018.

6 Vittorio Bilò and Cosimo Vinci. Dynamic taxes for polynomial congestion games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 839–856. ACM, 2016.

7 Avrim Blum, Eyal Even-Dar, and Katrina Ligett. Routing without regret: On convergence to Nash equilibria of regret-minimizing algorithms in routing games. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 45–52. ACM, 2006.

8 Maria Börjesson, Mogens Fosgerau, and Staffan Algers. On the income elasticity of the value of travel time. *Transportation Research Part A: Policy and Practice*, 46(2):368–377, 2012.

9 Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, Maria Kyropoulou, and Evi Papaioannou. The Impact of Altruism on the Efficiency of Atomic Congestion Games. In Martin Wirsing, Martin Hofmann, and Axel Rauschmayer, editors, *Trustworthly Global Computing*, pages 172–188, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

10 Po-An Chen, Bart De Keijzer, David Kempe, and Guido Schäfer. The robust price of anarchy of altruistic games. In *International Workshop on Internet and Network Economics*, pages 383–390. Springer, 2011.

11 Zhou Chen, Qi Qi, and Changjun Wang. Balancing Efficiency and Equality in Vehicle Licenses Allocation. In *International Conference on Web and Internet Economics*. Springer, 2017.

12 Zhou Chen, Qi Qi, Changjun Wang, and Wenwei Wang. Mechanism Design with Efficiency and Equality Considerations. In *International Conference on Web and Internet Economics*, pages 120–132. Springer, 2017.

13 George Christodoulou and Elias Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 67–73. ACM, 2005.

14 Beng-Huat Chua. *Communitarian Ideology and Democracy in Singapore*. Routledge, London, 1995.

15 Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. Pricing network edges for heterogeneous selfish users. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 521–530. ACM, 2003.

16 José R Correa, Andreas S Schulz, and Nicolás E Stier-Moses. Selfish routing in capacitated networks. *Mathematics of Operations Research*, 29(4):961–976, 2004.

17 Robert H. Edelstein and Sau Kim Lum. House Prices, Wealth Effects, and the Singapore Macroeconomy. *Journal of Housing Economics*, 13(4):342–367, December 2004. doi:10.1016/j.jhe.2004.09.006.

18 Lisa Fleischer, Kamal Jain, and Mohammad Mahdian. Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 277–285. IEEE, 2004.

19 Dimitris Fotakis, Laurent Gourvès, and Jérôme Monnot. Selfish transportation games. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 176–187. Springer, 2017.

20 Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2–3):226–239, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004)Automata, Languages and Programming: Algorithms and Complexity 2004. doi:10.1016/j.tcs.2005.09.024.

21 Kurtuluş Gemici. The Neoclassical Origins of Polanyi's Self-Regulating Market. *Sociological Theory*, 33(2):125–147, June 2015. doi:10.1177/0735275115587389.

22 Kurtuluş Gemici, Elias Koutsoupias, Barnabé Monnot, Christos Papadimitriou, and Georgios Piliouras. Wealth inequality and the price of anarchy. arXiv:1802.09269.

23 Corrado Gini. Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126, 1921.

24 Sun Sheng Han. Polycentric Urban Development and Spatial Clustering of Condominium Property Values: Singapore in the 1990s. *Environment and Planning A: Economy and Space*, 37(3):463–481, March 2005. doi:10.1068/a3746.

25 Tobias Harks, Ingo Kleinert, Max Klimm, and Rolf H Möhring. Computing network tolls with support constraints. *Networks*, 65(3):262–285, 2015.

26 W. Gregg Huff. The Developmental State, Government, and Singapore's Economic Development Since 1960. *World Development*, 23(8):1421–1438, 1995.

27 Christopher Jencks. Does Inequality Matter? *Daedalus*, 131(1):49–65, 2002.

28 Elias Koutsoupias and Christos H. Papadimitriou. Worst-case Equilibria. In *STACS*, pages 404–413, 1999.

29 Barnabé Monnot, Francisco Benita, and Georgios Piliouras. Routing Games in the Wild: Efficiency, Equilibration and Regret. In *International Conference on Web and Internet Economics*, pages 340–353. Springer, 2017.

30 Barnabé Monnot, Erik Wilhelm, Georgios Piliouras, Yuren Zhou, Daniel Dahlmeier, Hai Yun Lu, and Wang Jin. Inferring Activities and Optimal Trips: Lessons From Singapore's National Science Experiment. In Michael-Alexandre Cardin, Saik Hay Fong, Pao Chuen Lui, and Yang How Tan, editors, *Complex Systems Design & Management Asia*, pages 247–264. Springer, 2016.

31 Arthur Cecil Pigou. *The economics of welfare*. Palgrave Macmillan, 1920.

32 Thomas Piketty. *Capital in the 21st Century*. Harvard University Press Cambridge, MA, 2014.

33 Karl Polanyi. *The Great Transformation: The Political and Economic Origins of Our Time*. Beacon Press, Boston, MA, 1957.

34 John Rawls. *A Theory of Justice*. Belknap Press, Cambridge, MA, rev. edition, 1999.

35 John Rawls. *Justice as Fairness: A Restatement*. Harvard University Press, Cambridge, MA, 2001.

36 Sean F. Reardon and Kendra Bischoff. Income Inequality and Income Segregation. *American Journal of Sociology*, 116(4):1092–1153, January 2011. doi:10.1086/657114.

37 Tim Roughgarden. Intrinsic robustness of the price of anarchy. In *Proc. of STOC*, pages 513–522, 2009.

38 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

39 Joseph E Stiglitz. *The price of inequality: How today's divided society endangers our future*. WW Norton & Company, 2012.

40 H Theil. *Economics and information theory*. North-Holland, Amsterdam, 1967. OCLC: 710718565.

41 Carl A. Trocki. *Singapore: Wealth, Power and the Culture of Control*. Psychology Press, 2006.

42 Erik Wilhelm, Don MacKenzie, Yuren Zhou, Lynette Cheah, and Nils Ole Tippenhauer. Evaluation of Transport Mode Using Wearable Sensor Data from Thousands of Students. In *Proceedings of the Transportation Research Board 96th Annual Meeting*, pages 1–18. Transportation Research Board, 2017.

43 Erik Wilhelm, Yuren Zhou, Nan Zhang, Jacksheng Kee, George Loh, and Nils Tippenhauer. Sensg: Large-scale deployment of wearable sensors for trip and transport mode logging. In *Transportation Research Board 95th Annual Meeting*, 2016.

Lean Tree-Cut Decompositions: Obstructions and Algorithms

Archontia C. Giannopoulou
LaS team, Technische Universität Berlin, Germany
archontia.giannopoulou@tu-berlin.de

O-joung Kwon
Department of Mathematics, Incheon National University, South Korea
ojoungkwon@inu.ac.kr

Jean-Florent Raymond
LaS team, Technische Universität Berlin, Germany
raymond@tu-berlin.de

Dimitrios M. Thilikos
AlGCo project-team, LIRMM, Université de Montpellier, CNRS, Montpellier, France
sedthilk@thilikos.info

Abstract

The notion of tree-cut width has been introduced by Wollan in [*The structure of graphs not admitting a fixed immersion*, Journal of Combinatorial Theory, Series B, 110:47–66, 2015]. It is defined via tree-cut decompositions, which are tree-like decompositions that highlight small (edge) cuts in a graph. In that sense, tree-cut decompositions can be seen as an edge-version of tree-decompositions and have algorithmic applications on problems that remain intractable on graphs of bounded treewidth. In this paper, we prove that every graph admits an optimal tree-cut decomposition that satisfies a certain Menger-like condition similar to that of the lean tree decompositions of Thomas [*A Menger-like property of tree-width: The finite case*, Journal of Combinatorial Theory, Series B, 48(1):67–76, 1990]. This allows us to give, for every $k \in \mathbb{N}$, an upper-bound on the number immersion-minimal graphs of tree-cut width k. Our results imply the *constructive* existence of a linear FPT-algorithm for tree-cut width.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases tree-cut width, lean decompositions, immersions, obstructions, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.32

Related Version The full version of the paper can also be found on arXiv at `https://arxiv.org/pdf/1808.00863`.

Funding *Archontia C. Giannopoulou*: Supported by the ERC consolidator grant DISTRUCT-648527.
O-joung Kwon: Supported by the ERC consolidator grant DISTRUCT-648527 and by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. NRF-2018R1D1A1B07050294).
Jean-Florent Raymond: Supported by ERC consolidator grant DISTRUCT-648527.
Dimitrios M. Thilikos: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010). Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.

Acknowledgements We are grateful to Michał Pilipczuk and Marcin Wrochna for extensive discussions about the proof of Theorem 1.

1 Introduction

Menger's theorem is a classic result in graph theory and arguably one of the most used. It can be informally stated as follows: two vertex sets of a graph can be linked by k vertex-disjoint paths if and only if they cannot be separated by the removal of k vertices [25]. A significant feature of this result is that it provides a concise certificate for the optimality (in terms of cardinality) of a collection of k disjoint paths between two sets, in the form of a separator of size k. Many variants of Menger's theorem are known, including edge and directed versions [25, 19].

In 1990, Thomas [31] obtained a similar duality between paths and separators in tree decompositions. Tree decompositions are central objects in the theory of graph minors of Robertson and Seymour. Introduced in the early times of their Graph Minors series [28], they soon became the standard way to approach problems related to minors (such as Disjoint Paths, Feedback Vertex Set, etc.) and to minor-closed graph classes. The result of Thomas is that every graph admits an optimal tree decomposition, said to be *lean*, where the following holds: two bags of the decomposition[1] can be connected by k vertex-disjoint paths if and only if they cannot be separated by the removal of a bag on at most k vertices. That is, while Menger's theorem provides a set of k vertices separating the two bags, Thomas proved that such vertices (again forming a concise certificate of optimality) can moreover be found in a unique bag. Lean tree decompositions had important applications as they have been used to optimize parameter dependencies and to simplify arguments in proofs of the Graph Minors series [2, 9].

The objects of our attention in this paper are tree-like decompositions of graphs called *tree-cut decompositions*. These decompositions, introduced by Wollan in [32], highlight small (edge) cuts in graphs and are particularly suited to deal with problems related to the immersion ordering[2] and immersion-closed graph classes. (Comparatively, tree decompositions display small vertex separators and are tailored for minor-related problems.) A strength of tree-cut decompositions compared to other decompositions defined by edge cuts (such as cutwidth orderings or carving decompositions) is that they enjoy two highly desirable properties that mirror those of tree decompositions: (a) they admit an excluded wall immersion theorem [32] (comparable to the excluded grid minor theorem of Robertson and Seymour [29]) and (b) they can be used for dynamic programming, additionally for problems that cannot be tackled under the bounded-treewidth framework [11, 21, 15]. Thus they are believed to be a credible translation of tree decompositions to the settings of immersions. The importance of tree decomposition-based techniques in graph algorithms is an additional motivation to study their possible counterpart in the immersion world. Our first contribution is to show that paths and cuts in tree-cut decompositions obey a duality as in the aforementioned result of Thomas.

▶ **Theorem 1.** *Every graph has an optimal tree-cut decomposition that is lean.*

For the above statement to be meaningful, we adapted the notion of leanness to the setting of tree-cut decomposition; in particular it relates edge-disjoint paths with edge cuts.[3] A similar notion of leanness has been previously studied in [16] for the simpler setting of cutwidth orderings. We also note that analogs of lean tree decompositions appeared for

[1] We follow here the standard terminology about tree-decompositions, see Section 6 for details.
[2] The immersion ordering is a partial order on graphs that has properties similar to the minor ordering, see Section 2 for details.
[3] The formal definition is given in Section 2.

several different width parameters such as θ-tree-width [6, 13], pathwidth [24], directed path-width [22], DAG-width [23], rank-width [26], linear-rankwidth [20], profile- and block-width [9], matroid treewidth [12, 1, 9] and matroid branchwidth [12].

To prove Theorem 1, we design an improvement procedure that, in a finite number of steps, transforms a tree-cut decomposition that is not lean into a lean one.

Tree-cut decompositions can be used to define the graph parameter of *tree-cut width*, in the same way that treewidth can be defined via tree decompositions. Our second result pertains to *obstructions* for bounded tree-cut width. We say that a graph is an (*immersion*)-*obstruction* for tree-cut width at most k if it has tree-cut width more than k and all its proper immersions have tree-cut width at most k. Intuitively, such graphs delimit the border between the graphs of small tree-cut width (i.e. at most k) and the rest. As a consequence of the results of Robertson and Seymour in [27], the set $\mathbf{obs}(k)$ of obstructions for tree-cut width at most k is finite, for every $k \in \mathbb{N}$. Unfortunately the techniques used to prove this result (namely, properties of well-quasi-orders) do not provide explicit upper-bounds on its size. We consider here the related question of estimating the size of the elements of $\mathbf{obs}(k)$. This question has received significant attention in related settings. In [24], Lagergren gave exponential upper-bounds on the order of minor-obstructions to graphs of pathwidth or treewidth at most k ($2^{O(k^4)}$ and $2^{2^{O(k^5)}}$), respectively. Moreover, the size of immersion-obstructions to graphs of cutwidth at most k is known to lie between $(3^{k-5}-1)/2$ [17] and $2^{O(k^3 \log k)}$ [16]. As noted in [5], these size estimations can be important in practice. Our result on this topic is a constructive upper-bound on the size of obstructions for bounded tree-cut width.

▶ **Theorem 2.** *Let $k \in \mathbb{N}$. If G has tree-cut width more than k and every proper immersion of G has tree-cut width at most k, then G has at most $2^{2^{O(k^4)}}$ vertices.*

Our proof has two main ingredients. The first one is an encoding into a finite number of types of the features of a graph that are relevant when computing tree-cut width, in the fashion of the folios of [30] and the protrusion machinery of [4], but adapted to the different setting studied here (and similar in this sense to the techniques developed in [16, 7]). The second ingredient is our aforementioned result on leanness. Informally, Theorem 1 allows us, when "redundancy" has been identified in a graph (using the first ingredient), to obtain an immersion that has the same tree-cut width and less vertices. To the best of our knowledge, no explicit upper-bound concerning $\mathbf{obs}(k)$ was known before.

The third problem that we consider is of algorithmic nature: how fast can the tree-cut width of a graph be computed? This problem is NP-hard [21] and therefore has been approached through parameterized complexity and approximation algorithms. A consequence of the aforementioned result of Robertson and Seymour [27] and a result in [18] is the existence – and the existence only – of an FPT algorithm that decides, given a graph G and $k \in \mathbb{N}$, whether G has tree-cut width at most k. The first step towards a constructive FPT algorithm for tree-cut width was achieved by Kim et al. [21] who designed a $2^{O(k^2 \log k)} \cdot n^2$-time factor 2 approximation algorithm. Our third result is the first constructive parameterized algorithm for tree-cut width.

▶ **Theorem 3.** *It is possible to construct an algorithm that given an n-vertex graph G and an integer r decides whether G has tree-cut width more than r in $f(r) \cdot n$ steps, where f is some recursive function.*

Our algorithm relies on the fact that graphs of small tree-cut width have small treewidth and on our result on the size of the obstructions. Graphs of large treewidth can be immediately rejected, while graphs of small treewidth can be searched for the existence of an obstruction to tree-cut width at most k using standard techniques on tree decompositions. We leave as an open problem to determine the optimal order of magnitude for f.

Organization of this extended abstract. In Section 2, we give preliminary definitions. The proofs of Theorem 1 is given in Section 3. In Sections 4 and 5 we introduce the machinery used to prove Theorems 2 and 3 whose proofs are given in Section 6. Finally, in Section 7, we provide some discussion on how to bound the parametric dependence of the algorithm of Theorem 3. Due to the space constraints, the proofs of the lemmas marked by ★ have been omitted. They can be found in the full version [14].

2 Preliminaries

In this paper, graphs are undirected, finite, loopless, and may have multiple edges. We respectively denote by $V(G)$ and $E(G)$ the vertex and edge sets of a graph G and by $|G|$ and $\|G\|$ the cardinalities of these sets (counting edges with multiplicities).

A *cut* in a graph G is a set $F \subseteq E(G)$ such that $G - F$ has more connected components than G. If $A, B \subseteq V(G)$, the cut F is an (A, B)-cut if there is no path connecting a vertex of A to a vertex of B in $G - F$. If $A, B \subseteq E(G)$, we say that F is an (A, B)-cut if there is no path from an endpoint of an edge of A to an endpoint of an edge of B in $G - F$. If (X, Y) is a partition of $V(G)$, we sometimes refer to the cut $\{xy \in E(G),\ x \in X \text{ and } y \in Y\}$ by (X, Y). For $k \in \mathbb{N}$, a graph is said to be *k-edge-connected* if it has no cut on (strictly) less than k edges.

If T is a tree and $a, b \in E(T)$, we denote by aTb the (unique) path of T from a to b and containing these edges.

Immersions. For graphs H and G, a pair (ϕ, ψ) is an *H-immersion model* in G if

- ϕ is an injection from $V(H)$ to $V(G)$,
- ψ maps every edge uv of H to a path of G between $\phi(u)$ and $\phi(v)$ such that different edges are mapped to edge-disjoint paths.

We say that H is an *immersion* of G if there is an H-immersion model in G and we denote this by $H \leq G$. This defines a partial order on graphs. We also say that H is a *proper immersion* of G if $H \leq G$ and H is not isomorphic to G.

Tree-cut decompositions. A *near-partition* of a set S is a family of pairwise disjoint subsets $S_1, \ldots, S_k \subseteq S$ such that $\bigcup_{i=1}^{k} S_i = S$. Observe that this definition allows some sets S_i of the family to be empty.

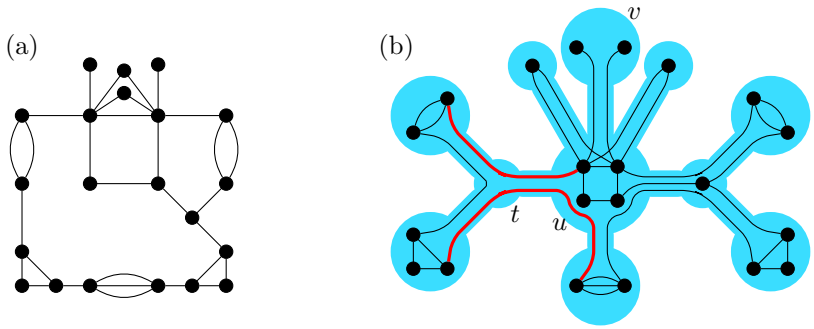


Figure 1 Example of a tree-cut decomposition (right) of a graph (left).

A *tree-cut decomposition* of a graph G is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X} = \{X_t\}_{t \in V(T)}$ is a near-partition of $V(G)$. We call elements of $V(T)$ *nodes* and elements of $V(G)$ *vertices* for clarity. The set X_t is called the *bag* of the decomposition corresponding to the node t, or just the *bag at* t.

An example of a tree-cut decomposition $(\mathcal{X}, T)$ of a graph G is depicted in Figure 1. In this picture, the tree T is depicted in blue and G is drawn on top of it to highlight which vertices (resp. edges) belong to which bags (resp. adhesions). This decomposition has an empty bag, at node t.

For an edge $\{u, v\} \in E(T)$, we denote by T_{uv} and T_{vu} the connected components of $T - \{uv\}$ containing u and v, respectively. The *adhesion* $\mathsf{adh}_{(T,\mathcal{X})}(uv)$ of some edge $uv \in E(T)$ is defined as the set of edges of G that have an endpoint in a bag of T_{uv} and the other one in a bag of T_{vu}. We drop the subscript when it is clear from the context. In Figure 1, the adhesion of the edge tu of T consists in the two thick red edges of G.

If G is 3-edge-connected, we define the width of $(T, \mathcal{X})$ as follows:

$$\mathtt{width}(T, \mathcal{X}) = \max \left\{ \max_{tt' \in E(T)} |\mathsf{adh}(tt')|, \max_{t \in V(T)} (|X_t| + \deg_T(t)) \right\}.$$

The *tree-cut width* of G is the maximum width over all tree-cut decompositions of G. For the simplicity of the presentation we only work on 3-edge-connected graphs in the extended abstract[4]. However, our results hold for all graphs and full proofs can be found in the full version [14]

3 A Menger-like property of tree-cut width

In this section, we give a formal definition of the notion of leanness for tree-cut decompositions and we sketch the proof of Theorem 1.

A tree-cut decomposition $(T, \mathcal{X})$ is said to be *lean* if, for every distinct edges $a, b \in E(T)$ and every $A \subseteq \mathsf{adh}(a), B \subseteq \mathsf{adh}(b)$ with $|A| = |B| =: k$,

- either there are k edge-disjoint paths linking A to B;
- or there is an edge e in aTb such that $|\mathsf{adh}(e)| < k$.

Observe that when the second point holds, $\mathsf{adh}(e)$ is an (A, B)-cut. Therefore, as with Thomas' notion of leanness mentioned in the introduction, the absence of a large collection of (here edge-disjoint) paths can be certified by a small (edge) separator that has restricted position.

Out first result (Theorem 1) shows that every graph admits a tree-cut decomposition that is lean and has optimum width. Its proof can in fact be reduced to the case of 3-edge-connected graphs.

▶ **Lemma 4** (★ (included in the proof of Theorem 1)). *Theorem 1 holds iff it holds for 3-edge-connected graphs.*

Therefore we can now focus on 3-edge-connected graphs. A crucial element of the proof of Theorem 1 is the definition of a potential on tree-cut decompositions, called *fatness*, as in Bellenbaum and Diestel's proof of Thomas leanness result for tree-decompositions [2]. Let G be a graph on m edges and let $(T, \mathcal{X})$ be a tree-cut decomposition of G. For every $i \in [m]$, we denote by $T^{\geq i}$ the subgraph of T induced by the edges that have adhesion at least i. The *fatness*, denoted by $\mathsf{fatness}(T, \mathcal{X})$, of $(T, \mathcal{X})$ is the $(2m)$-tuple $(\alpha_m, -\beta_m, \alpha_{m-1}, -\beta_{m-1}, \ldots, \alpha_1, -\beta_1)$, where for every $i \in [m]$,

[4] There is a definition the width for tree-cut decompositions of graphs that are not 3-edge-connected. It can be found in the full version [14]. The definition that we give here is equivalent to that of Wollan in [32].

- α_i is the number of edges of $T^{\geq i}$; and
- β_i is the number of connected components of $T^{\geq i}$.

We order fatnesses by lexicographic order. Informally, fatness is used to quantify the progress made when improving a non-lean tree-cut decomposition. This is captured by the following lemma which constitutes an important technical ingredient of the paper.

▶ **Lemma 5 (★).** *Let G be a 3-edge-connected graph. If $(T, \mathcal{X})$ is a tree-cut decomposition of G that is not lean, then G has a tree-cut decomposition of smaller fatness and no bigger width than the one of $(T, \mathcal{X})$.*

As the fatness of tree-decompositions of a given graph can take a finite number of values, Lemma 5 implies that we can, after a finite number of "improvement steps", obtain a lean tree-cut decomposition. Let us now describe what we mean by improvement step. Let G be

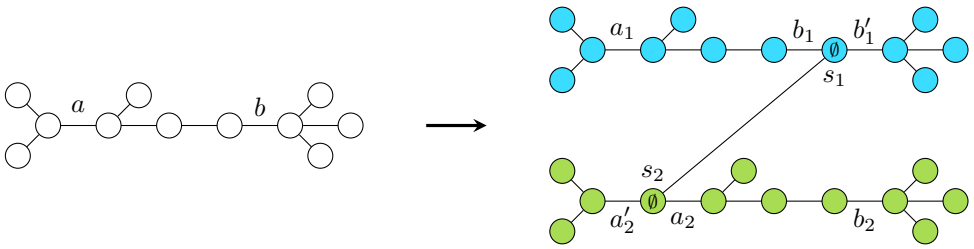


Figure 2 A tree-cut decomposition of a graph G (left) and its (a, b, F)-segregation (right), where F is a cut that separates G into G_A and G_B. The vertices of G_A and G_B respectively lie in blue and green bags. Newly introduced bags, corresponding to nodes s_1, s_2, are empty. The adhesion of $s_1 s_2$ is exactly F.

such a graph and let $(T, \mathcal{X})$ be a tree-cut decomposition of G. Let $a, b \in E(T)$, and let F be an inclusion-wise minimal cut separating a graph G into two graphs that we call G_A and G_B. We define the (a, b, F)-*segregation* of $(T, \mathcal{X})$ as the pair $(U, \mathcal{Y})$ obtained as follows:

1. consider a first copy T_1 of T, subdivide once the edge corresponding to b, call s_1 the subdivision vertex, and call the two created edges b_1 and b'_1, with the convention that (the copy of) a is closer to b_1 in T_1;
2. symmetrically, consider a second copy T_2 of T, subdivide once the edge corresponding to a, call s_2 the subdivision vertex, and call the two created edges a_2 and a'_2, with the convention that (the copy of) b is closer to a_2 in T_2;
3. in the disjoint union of these two trees, add an edge joining s_1 and s_2: this gives U;
4. for every $t \in V(U)$, let $Y_t = \begin{cases} X_t \cap V(G_A) & \text{if } t \in V(T_1) \setminus \{s_1\} \\ X_t \cap V(G_B) & \text{if } t \in V(T_2) \setminus \{s_2\}, \text{ and} \\ \emptyset & \text{if } t \in \{s_1, s_2\}. \end{cases}$

The construction of an (a, b, F)-segregation is depicted in Fig. 2. It is not hard to see that a segregation of a tree-cut decomposition is still a tree-cut decomposition.

For every non-lean tree-cut decomposition $(T, \mathcal{X})$ there are edges $a, b, e \in E(T)$ and subsets $A \subseteq \mathsf{adh}(a)$, $B \subseteq \mathsf{adh}(b)$ violating the definition of leanness. The proof of Lemma 5 consists in showing that the (a, b, F)-segregation of $(T, \mathcal{X})$, for some cut F carefully chosen using a, b, e, A, B, has the same width as $(T, \mathcal{X})$ and smaller fatness.

4 Tidy decompositions and branch interfaces

For the rest of the proofs we need the notion of *tidiness* of a tree-cut decomposition of a graph G, with respect to some edge-cut F of G. Informally, the *tidiness* property, is satisfied in a tree-cut decomposition of G, when, given a cut in a graph, the tree-cut decomposition can be *almost* partitioned (excluding a set of nodes whose size is upper-bounded linearly by the size of F) to subtrees in such a way that the vertices of the graph residing in each subtree reside also to the same side of the cut. (See also Figure 3.)

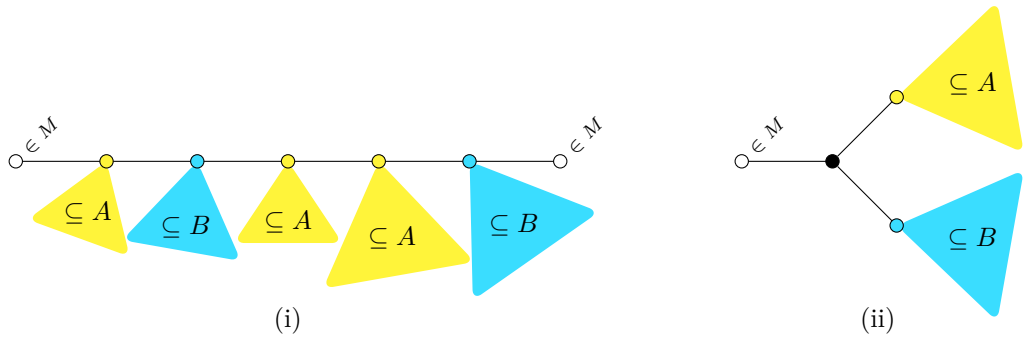


Figure 3 Parts of an (A, B)-tidy tree-cut decomposition, with subtrees colored differently depending whether the bags of their vertices are included in A or in B.

To clearly define tidy tree-cut decompositions and state the main lemma of this section we need the following.

For a tree T and a set $M \subseteq V(T)$, the *least common ancestor closure* (*lca-closure*) of M is the set $\mathsf{lca}(M) \subseteq V(T)$ obtained from M by repeatedly adding to it, for every triple x, y, z in the set, the common vertex of the paths xTy, yTz, and zTx.

▶ **Lemma 6** ([10]). *Let T be a rooted forest and $M \subseteq V(T)$. Then $|\mathsf{lca}(M)| \leq 2|M|$ and every connected component C of $T - \mathsf{lca}(M)$ has at most two neighbors in T.*

Let T be a tree, $M \subseteq V(T)$, and $v \in V(T)$. A node $u \in V(T)$ is a *M-descendant* of v if every path from u to a vertex of M contains v. We denote by $\mathsf{desc}_T^M(v)$ the set of such vertices (we drop the subscript when it is clear from the context). In particular $v \in \mathsf{desc}^M(v)$ holds for every $v \in V(T)$ and $\mathsf{desc}^M(v) = \{v\}$ holds for every $v \in M$. We also use the notation $\mathsf{desc}_T(v)$ instead of $\mathsf{desc}_T^{V(G)}(v)$.

▶ **Definition 7** (tidy tree-cut decomposition). *Let $\mathcal{D} = (T, \{X_u\}_{u \in V(T)})$ be a tree-cut decomposition of a graph G and let (A, B) be a cut of this graph. Let $M \subseteq V(T)$ be the lca-closure of the nodes of T whose bags contain endpoints of the cut (A, B). We denote by $\mathcal{C}_i^M$, $i \in [2]$ the set of connected components of $T - M$ that have exactly i neighbors in M (and drop the superscript M if it is clear from the context).*

We say that $\mathcal{D}$ is (A, B)-tidy *if the following conditions hold for every connected component F of $T - M$:*

- **(i)** *if $F \in \mathcal{C}_2^M$: for every $w \in V(F)$, all the bags at $T\,\mathsf{desc}^M(w)$ are subsets of one of A and B;*
- **(ii)** *if $F \in \mathcal{C}_1^M$: let u be the node of F adjacent to M. Then, first, u has at most two neighbors in F; let us call them v, w (if they exist). Second, X_u intersects at most one of A and B and all the bags at $T\,\mathsf{desc}^M(v)$ (resp. $T\,\mathsf{desc}^M(w)$) are subsets of A (resp. B), or the other way around.*

These two situations are again depicted on Fig. 3.

Furthermore, we show that there exists a tidy tree-cut decomposition of optimum width where the number of these subtrees is upper-bounded by a function of the size of the cut and the tree-cut width of the graph.

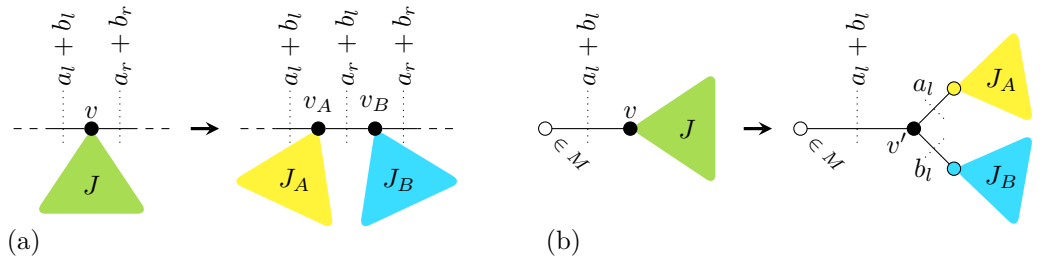


Figure 4 Improvement step towards obtaining a tidy tree-cut decomposition from a general tree-cut decomposition without increasing the width.

Before we present the lemma, we need the following definition.

Let G be a graph and let $\mathcal{D} = (T, \{X\}_{u \in V(T)})$ be a tree-cut decomposition of G. For $X \subseteq V(G)$, a *X-block* is a maximal subtree F of T such that:

- bags at nodes of F are subsets of X;
- at most two nodes of $T - V(F)$ have a neighbor in F.

If (A, B) is a cut, we refer to A- and B- blocks as *(A, B)-blocks*.

▶ **Lemma 8 (★).** *Let $\ell \in \mathbb{N}_{\geq 3}$ and G be a 3-edge-connected graph. If (A, B) is a cut of G of size ℓ, then there is an optimum tidy tree-cut decomposition $\mathcal{D} = (T, \mathcal{X})$ of G with less than $8\ell \cdot \mathbf{tcw}(G)$ (A, B)-blocks in $T - M$, where M is the lca-closure of the nodes of T whose bags contain endpoints of the cut (A, B).*

Note that from the definition of a block, two A-blocks (resp. B-blocks) cannot be connected by a link of T. This simple property is crucial in the proof of Lemma 8. It allows us, given a tree-cut decomposition of a graph G that has many (A, B)-blocks, to swap two of them (hence decreasing the total number of blocks) without increasing the width of the decomposition. Fig. 4 illustrates how we may obtain a tidy tree-cut decomposition of optimum width from a general tree-cut decomposition of optimum width while Fig. 5 illustrates the intuition behind swapping A- and B-blocks in order to decrease their total number.

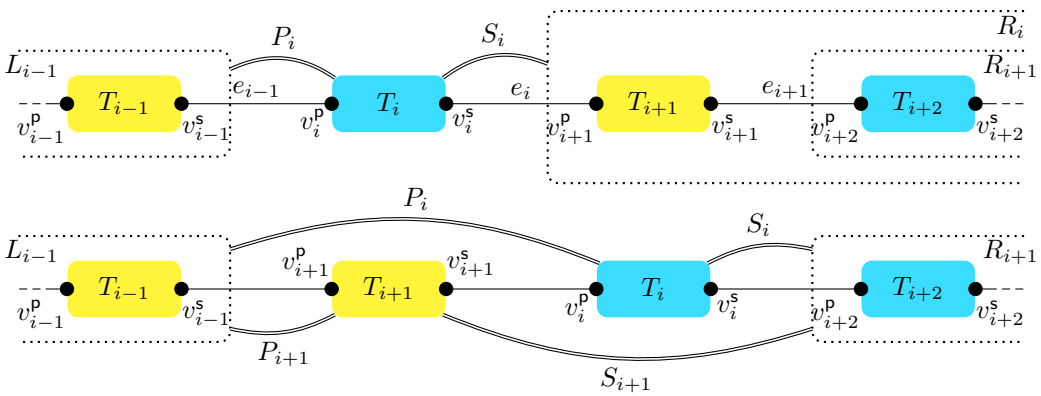


Figure 5 From $\mathcal{D}$ (top) to $\mathcal{D}'$ (bottom): swapping the blocks T_i and T_{i+1}.

5 Branch Interfaces

Tidy tree-cut decompositions will be used in this section to classify certain subgraphs of a graph into a bounded number of equivalence classes. This will be directly used to obtain our result on obstructions.

A *k-boundaried graph* is a pair $\mathbf{G} = (G, \bar{x})$ where G is a graph and $\bar{x} = (x_1, \ldots, x_k)$ is a k-tuple of the graph's vertices, called *boundary vertices* (ordered, not necessarily distinct). The *extension* of $\mathbf{G}$ is the graph G^* obtained from G by adding k new vertices $x'_1, \ldots, x'_k$ and edges $x_1x'_1, \ldots, x_kx'_k$. The *join* $\mathbf{A} \oplus \mathbf{B}$ of two k-boundaried graphs $\mathbf{A} = (A, \bar{x}), \mathbf{B} = (B, \bar{y})$ is the graph obtained from the disjoint union of A and B by adding an edge x_iy_i for $i \in [k]$. Intuitively, a boundaried graph can be seen as the subgraph corresponding to one side of an (edge) cut in a larger graph. Its boundary vertices then record the endpoints of the edges of the cut and can be glued to an other boundaried graph to recover the original graph.

We note that our definition of boundaried graph (already used in previous work [16, 15, 8, 7]) slightly differs from that usually followed in the literature (e.g. in [4]), in particular the join operation. This is because the problems that we consider are related to (edge) cuts – a setting where our definition is more natural – while other papers often deal with settings pertaining to (vertex) separators, typically graphs of bounded treewidth.

Let $\mathbf{A}$ and $\mathbf{B}$ be two k-boundaried graphs and let $G = \mathbf{A} \oplus \mathbf{B}$. (This is roughly the same as saying that G has a cut (A, B) of size k.) Let $\mathcal{D} = (T, \mathcal{X})$ be an (A, B)-tidy tree-cut decomposition of G. Let M denote the lca-closure of the vertices of T whose bags contain endpoints of the cut.

Let us construct a new pair $\mathcal{B} = (U, \mathcal{Y})$ from $\mathcal{D}$ as follows: for every (A, B)-block F of T, we contract F to a single vertex x and define Y_x as the set of vertices of G that are contained in bags at F. For every $x \in M$ we set $Y_x = X_x$. We call $\mathcal{B}$ a *bucketing* of G. In order to avoid confusion with the bags of a tree-cut decomposition, we refer to the elements of $\mathcal{Y}$ as *buckets*. Recall that by Lemma 8, the vertices of M are separated in T by a bounded number of alternations of (A, B)-blocks. Therefore the size of U is bounded by a function of k and $\mathbf{tcw}(G)$.

A *bucketing* of $\mathbf{A}$ is obtained from a bucketing of G by forgetting the content of the buckets corresponding to $\mathbf{B}$; formally it is the pair $(U, \mathcal{Y}')$ where $\mathcal{Y}' = \{Y_u \cap V(A),\ u \in V(U - M)\} \cup \{Y_u,\ u \in M\}$. Given the bucketing $(U, \mathcal{Y}')$ of $\mathbf{A}$ together with $\mathcal{D}$, we can compute:

- for each edge e of U, the size $z(e)$ of its adhesion (defined as for tree-cut decompositions);
- for each node $x \in V(T - M)$, the width $z(x)$ of the corresponding (A, B)-block in $\mathcal{D}$;[5]
- for each node $x \in V(T - M)$, the maximum $\alpha(x)$ of adhesions sizes over the subpath linking the attachement points (i.e. vertices with a neighbor outside of the block) of the corresponding (A, B)-block in $\mathcal{D}$;
- for each node $x \in M$, the sum $z(x)$ of its degree in T and bag size in $\mathcal{D}$;
- a function $f\colon [2k] \to V(U)$ specifying which buckets of U contain the endpoints of the cut.

Informally, these values encode all the relevant information about the contribution of $\mathbf{A}$ to the width of $\mathcal{D}$. If $\ell = \mathbf{tcw}(G)$, then we say that (U, z, α, f) (seen as a tuple independent of $\mathbf{A}$) is a (k, ℓ)-*branch interface*[6] and we say that $\mathbf{A}$ *conforms* with this branch interface.

[5] That is, the width of $\mathcal{D}$ restricted to this block.

[6] Note that we define here a slightly simplified notion of branch interface than in the full version [14], which is sufficient for the presentation given in this extended abstract.

Observe that the number of (k,ℓ)-branch interfaces can be upper-bounded by a function of k and ℓ. In particular we have the following.

▶ **Observation 9.** *For all $k,\ell \in \mathbb{N}$ there are $2^{O(\ell k \log k)}$ (k,ℓ)-branch interfaces.*

We denote by $\mathcal{I}_{k,\ell}(\mathbf{A})$ the set of (k,ℓ)-branch interfaces $\mathbf{A}$ conforms with. Intuitively, this set records all the possible ways for $\mathbf{A}$ to appear in a tidy tree-cut decomposition of a graph $\mathbf{A} \oplus \mathbf{B}$ of tree-cut width at most ℓ. That is, whatever k-boundaried graph $\mathbf{B}$ we consider (among those such that $\mathbf{tcw}(\mathbf{A} \oplus \mathbf{B}) \leq \ell$), all the possible ways (A,B)-blocks can interleave and contribute to the width in a tidy tree-cut decomposition of $\mathbf{A} \oplus \mathbf{B}$ is of bounded size and recorded in $\mathcal{I}_{k,\ell}(\mathbf{A})$. This is formalized by the following lemma.

▶ **Theorem 10** (edge-protrusion replacement lemma ★)**.** *Let k,ℓ be two positive integers. Let also $\mathbf{A}_1$ and $\mathbf{A}_2$ be two k-boundaried graphs with $\mathcal{I}_{k,\ell}(\mathbf{A}_1) = \mathcal{I}_{k,\ell}(\mathbf{A}_2)$. Then for any k-boundaried graph $\mathbf{B}$ where $\mathbf{tcw}(\mathbf{A}_1 \oplus \mathbf{B}) \leq \ell$ and such that both $\mathbf{A}_1 \oplus \mathbf{B}$ and $\mathbf{A}_2 \oplus \mathbf{B}$ are 3-edge connected, it holds that $\mathbf{tcw}(\mathbf{A}_2 \oplus \mathbf{B}) = \mathbf{tcw}(\mathbf{A}_1 \oplus \mathbf{B})$.*

6 Obstructions and algorithms for tree-cut width

We explain in this section how the machinery introduced in the two previous sections can be used to obtain theorems 2 and 3.

Let $\mathbf{H} = (H,\overline{x})$ and $\mathbf{G} = (G,\overline{y})$ be two k-boundaried graphs. If there is an H^*-immersion model (ϕ,ψ) of G^* where $\phi(x_i) = y_i$ and $\phi(x_i') = y_i'$ for each $i \in [k]$, then we say that $\mathbf{H}$ is a *rooted immersion* of $\mathbf{G}$ and we denote this by $\mathbf{H} \leq \mathbf{G}$.

Immersion obstructions. For every $k \in \mathbb{N}$, we denote by $\mathbf{obs}(k)$ the set of all immersion-minimal graphs whose tree-cut width is strictly greater than k. For instance, it is easy to see that $\mathbf{obs}(1)$ contains only the graph with two vertices and a double edge between them.

Our purpose is to provide a bound to the order of the graphs in $\mathbf{obs}(k)$, as a function of k. It follows from [21, Lemmata 3 and 4] that all graphs in $\mathbf{obs}(k)$ are 3-edge-connected for $k \geq 2$.

We use the following lemmata.

▶ **Proposition 11** ([16])**.** *Let w,m be positive integers and let y be a word in $[w]^*$ of length m^w. Then there is a number $k \in [w]$ and a subword u of y such that all numbers in u are at least k and u contains the number k at least m times.*

▶ **Lemma 12** (★)**.** *Let $k \in \mathbb{N}$ and let $\mathbf{H} = (H,\overline{x})$ and $\mathbf{G} = (G,\overline{y})$ be two k-boundaried graphs. For every $\ell \in \mathbb{N}$, if $\mathbf{H} \leq \mathbf{G}$, then $\mathcal{I}_{k,\ell}(\mathbf{G}) \subseteq \mathcal{I}_{k,\ell}(\mathbf{H})$.*

▶ **Lemma 13.** *There exists a function $g : \mathbb{N}^2 \to \mathbb{N}$ such that for every $w,r \in \mathbb{N}$ and every 3-edge-connected graph G where $\mathbf{tcw}(G) \leq w$ and $|V(G)| > g(r,w)$, G contains as a proper immersion a graph H for which $\mathbf{tcw}(H) \leq r$ implies $\mathbf{tcw}(G) \leq r$. Moreover such a function can be chosen so that $g(r,w) = 2^{2^{O(r \cdot w^3)}}$.*

Proof. By Lemma 5, there exists a lean tree-cut decomposition $\mathcal{D} = (T,\mathcal{X})$ of G with width at most w. We also assume that for every node t of T of degree at most 2, the bag X_t is non-empty, otherwise we revise $\mathcal{D}$ by contracting a link incident to t. The 3-edge connectivity of G implies that $\Delta(T) \leq w$. We root T on some, arbitrarily chosen, node s. This permits us to see T as a directed tree where all links are oriented towards the root s. In particular, if (v,u) is a link of T, we always assume that u is closer than v to the root.

From Observation 9 there is a function $\alpha : \mathbb{N}^2 \to \mathbb{N}$, with $\alpha(w,\ell) = 2^{O(\ell \cdot w \cdot \log w)}$, such that the number of the different (w,ℓ) interfaces of w-boundaried graphs is upper-bounded by $\alpha(w,\ell)$. We define $d = m^w$ where $m = \alpha(w,\ell)+1$ and $\ell = 8wr$. Let z be a node of T of maximum distance from the root s. Notice that z is a leaf of T. Let also $f : \mathbb{N}^2 \to \mathbb{N}$ be a function such that if $|T| > f(d,w)$, then T contains a path P of length d from z to some node, say u that is different than the root s. Notice that, as $\Delta(T) \leq w$, we can choose f such that $f(d,w) = 2^{O(d \log w)}$. We also set up the function $g : \mathbb{N}^2 \to \mathbb{N}$ where $g(r,w) = f(d,w) \cdot w$. Clearly, $g(r,w) = f(m^w,w) \cdot w = 2^{O((2^{O(\ell \cdot w^2 \cdot \log w)})^w \cdot \log w)} \cdot w = 2^{2^{O(r \cdot w^3)}}$.

Notice that if $|G| > g(r,w)$, then $|T| > f(d,w)$ and the aforementioned path P exists in T. Let $e_1, \ldots, e_d$ be the links of P ordered so that e_1 is the one that is closer to the root and let $w_{e_i} = |\mathsf{adh}_{\mathcal{D}}(e_i)|$. We now see the string $y = w_{e_1} \ldots w_{e_d}$ as a word in $[w]^*$. As $d = m^w$, from Proposition 11 there is some $k \in [w]$ and a subword $y' = w_{e_i} \ldots w_{e_j}$ of y such that $w_{e_h} \geq k$ for $h \in [i,j]$ and there is some $I \subseteq [i,j]$ such that $m = |I|$ and $\forall h \in I\ w_{e_j} = k$. Let $f_1, \ldots, f_m$ be the links in $\{e_h \mid h \in I\}$ ordered as a subsequence of $e_1, \ldots, e_d$. As $\mathcal{D}$ is lean, we know that there exist k edge-disjoint paths from $\mathsf{adh}_{(T,\mathcal{X})}(f_1)$ to $\mathsf{adh}_{(T,\mathcal{X})}(f_m)$. We denote these paths by $P_1, \ldots, P_k$. Notice also that for every $q \in [m]$, $\mathsf{adh}_{(T,\mathcal{X})}(f_m)$ contains *exactly one* edge from each of the paths in $P_1, \ldots, P_k$. We denote by $e_{q,p}$ the unique edge in $E(P_k) \cap \mathsf{adh}_{(T,\mathcal{X})}(f_q)$, for $(p,q) \in [k] \times [m]$. for every $q \in [m]$, we define $\lambda_q : \mathsf{adh}_{(T,\mathcal{X})}(f_q) \to [w_{f_q}]$ so that $\lambda_q(e_{q,p}) = p$ for $p \in [k]$.

For $q \in [m]$, we denote by t_q the tail of the edge f_q and we define $\mathbf{A}_q = (A_q, \bar{x}_q)$ where $A_q = G[\bigcup_{t \in \mathsf{desc}_T(t_q)} X_t]$ and $\bar{x}_q$ are the endpoints of the edges in $\mathsf{adh}_{(T,\mathcal{X})}(f_q)$ that belong to A_1, ordered according to λ_q (recall that $\mathsf{desc}_T(t_q)$ consists of the descendants of t_q, including t_q, in the rooted tree T). We also set $\mathbf{B}_q = (B_q, \bar{x}'_q)$ where $B_q = G - V(A_q)$ and $\bar{x}'_q$ are the endpoints of the edges in $\mathsf{adh}_{(T,\mathcal{X})}(f_q)$ that belong to B_q, again ordered according to λ_q. Notice that for every $q \in [m]$, $\mathbf{B}_q \oplus \mathbf{A}_q = G$. Moreover for every $(q,q') \in [m]^2$, where $q \leq q'$, there are k edge-disjoint paths in $\mathbf{A}_q$ that are joining, for each $p \in [k]$ the edge $e_{q,p}$ with the edge $e_{q',p}$. This implies that $\mathbf{A}_q \leq \mathbf{A}_{q'}$. This, combined with Lemma 12, yields that

$$\mathcal{I}_{k,\ell}(\mathbf{A}_1) \subseteq \cdots \subseteq \mathcal{I}_{k,\ell}(\mathbf{A}_m)$$

and, as $m = \alpha(r,w)+1$, there is a pair $(q,q') \in [m]^2$, where $q < q'$ and $\mathcal{I}_{k,\ell}(\mathbf{A}_q) = \mathcal{I}_{k,\ell}(\mathbf{A}_{q'})$. We set $H = \mathbf{A}_{q'} \oplus \mathbf{B}_q$. Since $q \neq q'$, H is a proper immersion of G. From Lemma 10, the fact that $\mathbf{tcw}(\mathbf{A}_q \oplus \mathbf{B}_q) = \mathbf{tcw}(G) \leq r$ implies that $\mathbf{tcw}(H) = \mathbf{tcw}(\mathbf{A}_q \oplus \mathbf{B}_{q'}) = \mathbf{tcw}(\mathbf{A}_q \oplus \mathbf{B}_q) = \mathbf{tcw}(G) \leq r$. ◀

▶ **Theorem 14** (Restatement of Theorem 2). *For every $k \in \mathbb{N}$, every graph in $\mathbf{obs}(k)$ has $2^{2^{O(k^4)}}$ vertices.*

Proof. Let $G \in \mathbf{obs}(k)$. This means that $\mathbf{tcw}(G) \geq k+1$ and that if H is a proper immersion of G, then $\mathbf{tcw}(H) < \mathbf{tcw}(G)$ and hence, $\mathbf{tcw}(H) \leq k$. It is easy to verify that if H is a proper immersion of G with $\|H\| = \|G\| - 1$ then $\mathbf{tcw}(H) \geq \mathbf{tcw}(G) - 1$. This last observation implies that $\mathbf{tcw}(G) \leq k+1$. If $|V(G)| > g(k,k+1)$ (where g is the function of Lemma 13) then Lemma 13 implies that G contains as a proper immersion a graph H for which $\mathbf{tcw}(H) \leq k$ implies that $\mathbf{tcw}(G) \leq k$. As $\mathbf{tcw}(H) \leq k$ for every proper immersion H of G, we deduce that $\mathbf{tcw}(G) \leq k$, a contradiction to the fact that $\mathbf{tcw}(G) \geq k+1$. The result follows as $g(k,k+1) = 2^{2^{O(k^4)}}$. ◀

Tree decompositions. A *tree decomposition* of a graph G is a pair $\mathcal{D} = (T, \mathcal{X})$, where T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of $V(G)$ such that:

- $\bigcup_{t \in V(T)} X_t = V(G)$,
- for every $uv \in E$, there is a $t \in V(T)$ such that $\{u, v\} \subseteq X_t$, and
- for each $\{x, y, z\} \subseteq V(T)$ such that z lies on the unique path between x and y in T, $X_x \cap X_y \subseteq X_z$.

The *width* of a tree decomposition $\mathcal{D} = (T, \mathcal{X})$ is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of a graph G, denoted by $\mathbf{tw}(G)$, is the smallest integer w such that there exists a tree decomposition of G of width at most w.

The proof of Theorem 3 comes as a direct consequence of Theorem 14.

Proof of Theorem 3. We set $n = |V(G)|$. Observe that $\Delta(G) = O_r(1)$. Moreover, if an edge has multiplicity strictly greater than r, then both its endpoints should be in the same bag of a tree-cut decomposition with width at most r. Therefore, we may replace each edge in G of multiplicity bigger than $r+1$ by one of multiplicity $r+1$ and this modification can be implemented in $O_r(n)$ steps. By applying this preprocessing we may assume that G has $O_r(n)$ edges.

In [11] it was proved that there exists a constant c_1 such that, for every graph H, $\mathbf{tw}(H) \leq c_1 \cdot (\mathbf{tcw}(H))^2$. We run the algorithm in [3] that, given a graph H and an integer q, either outputs a tree decomposition of H of width at most $5q$ or reports that $\mathbf{tw}(G) > q$, setting $H = G$ and $q = c_1 \cdot r^2$. If $\mathbf{tw}(G) > q$ we know that $\mathbf{tcw}(G) > r$ and we readily have an answer. Therefore, we can assume that we have a tree decomposition of G of width at most $5c_1 \cdot r^2$. As the property of the immersion-containment of a graph H can be expressed in Monadic Second Order Logic, by using Courcelle's theorem, it is possible to construct an algorithm that, given two graphs H_1 and H_2, outputs whether $H_1 \leq H_2$ in $O_{|V(H_1)|+\mathbf{tw}(G)}(|V(H_2)|)$ steps. By Theorem 14, the set $\mathbf{obs}(r)$ can be constructed in $O_r(1)$ steps. Then, by testing immersion-containment for every graph $H \in \mathbf{obs}(r)$, we can decide whether $\mathbf{tcw}(G) \leq r$ in $O_r(n)$ steps. ◀

7 Discussion

An interesting direction is to specify the parameterized dependence of the algorithm in Theorem 3. We argue below on why the proof of Lemma 13 can already give a first bound.

Notice that Lemma 10 defines an equivalence relation of w-boundaried graphs. We say that two k-boundaried graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ are (w, r)-*equivalent*, namely $\mathbf{G}_1 \equiv_{w,r} \mathbf{G}_2$ if $\mathcal{I}_{w,8wr}(\mathbf{G}_1) = \mathcal{I}_{w,8wr}(\mathbf{G}_2)$. Notice that the proof of Lemma 13 already implies that a minimum-size of a representative of any of the equivalence classes of $\equiv_{w,r}$ has $g(r, w) = 2^{2^{O(r \cdot w^3)}}$ vertices. By brute-force checking on all graphs on $g(r, w) = 2^{2^{O(r \cdot w^3)}}$ vertices, we may construct a set $\mathcal{R}_{w,r}$ of representatives for $\equiv$. Moreover $|\mathcal{R}_{w,r}| = 2^{2^{2^{O(r \cdot w^3)}}}$. The knowledge of $\mathcal{R}_{w,r}$ permits us to avoid the use of Courcelle's theorem in Theorem 3. For this, we first transform the tree decomposition of the proof of Theorem 3 to a tree-cut decomposition with width $w = O(r^4)$ using [32, Lemma 12]. Then we implement an algorithmic version of the proof of Lemma 13: as long as the sub-tree of T rooted on u of height $2^{2^{r \cdot w^3}}$ exists, we reduce $G = \mathbf{A}_q \oplus \mathbf{B}_q$ to the smaller equivalent instance $G = \mathbf{A}_{q'} \oplus \mathbf{B}_q$. Given that $|\mathcal{R}_{w,r}| = 2^{2^{2^{O(r \cdot w^3)}}}$ and using suitable data-structures, this compression can be implemented in $2^{2^{2^{O(r^{13})}}}$ steps and as it is repeated at most n times, it will report a correct answer in $2^{2^{2^{O(r^{13})}}} \cdot n$ steps.

Clearly, the parameterized dependence of the above argumentation is still too heavy. We believe that a detailed dynamic programming based on how collections of branch interfaces are updated along a rooted tree-cut decomposition may yield a bound $f(r) = 2^{\text{poly}(r)}$ to the function of Theorem 3.

A second direction for further research is to obtain an algorithm that, besides computing the tree-cut width of a graph, also provides a tree-cut decomposition of optimal width.

References

1 Jeffrey Azzato. Linked tree-decompositions of represented infinite matroids. *Journal of Combinatorial Theory, Series B*, 101(3):123–140, 2011. doi:10.1016/j.jctb.2010.12.003.

2 Patrick Bellenbaum and Reinhard Diestel. Two short proofs concerning tree-decompositions. *Combinatorics, Probability and Computing*, 11(6):541–547, 2002.

3 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $(c^k n)$ 5-Approximation Algorithm for Treewidth. *SIAM Journal of Computing*, 45(2):317–378, 2016.

4 Hans L Bodlaender, Fedor V Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M Thilikos. (Meta) kernelization. *Journal of the ACM*, 63(5):44, 2016.

5 Heather Booth, Rajeev Govindan, Michael A. Langston, and Siddharthan Ramachandramurthi. Cutwidth approximation in linear time. In *Proceedings of the Second Great Lakes Symposium on VLSI*, pages 70–73. IEEE, 1992.

6 Johannes Carmesin, Reinhard Diestel, M. Hamann, and Fabian Hundertmark. k-Blocks: A Connectivity Invariant for Graphs. *SIAM Journal on Discrete Mathematics*, 28(4):1876–1891, 2014.

7 Dimitris Chatzidimitriou, Jean-Florent Raymond, Ignasi Sau, and Dimitrios M. Thilikos. An $O(\log OPT)$-approximation for covering and packing minor models of θ_r. *Algorithmica*, April 2017. doi:10.1007/s00453-017-0313-5.

8 Dimitris Chatzidimitriou, Jean-Florent Raymond, Ignasi Sau, and Dimitrios M Thilikos. Minors in graphs of large θ_r-girth. *European Journal of Combinatorics*, 65:106–121, 2017.

9 Joshua Erde. A unified treatment of linked and lean tree-decompositions. *Journal of Combinatorial Theory, Series B*, 130:114–143, 2018.

10 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proceedings of FOCS 2012*, pages 470–479. IEEE Computer Society, 2012.

11 Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. In *Mathematical foundations of computer science 2015. Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 348–360. Springer, Heidelberg, 2015.

12 James F. Geelen, A. M. H. Gerards, and Geoff Whittle. Branch-width and well-quasi-ordering in matroids and graphs. *Journal of Combinatorial Theory, Series B*, 84(2):270–290, 2002.

13 Jim Geelen and Benson Joeris. A generalization of the Grid Theorem. *arXiv preprint*, 2016. arXiv:1609.09098.

14 Archontia C. Giannopoulou, O-joung Kwon, Jean-Florent Raymond, and Dimitrios M. Thilikos. Lean tree-cut decompositions: obstructions and algorithms. *arXiv e-prints*, August 2018. arXiv:1808.00863.

15 Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Linear Kernels for Edge Deletion Problems to Immersion-Closed Graph Classes. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 57:1–57:15, 2017.

16 Archontia C. Giannopoulou, Michał Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Cutwidth: Obstructions and Algorithmic Aspects. *Algorithmica*, March 2018. doi:10.1007/s00453-018-0424-7.

17 Rajeev Govindan and Siddharthan Ramachandramurthi. A weak immersion relation on graphs and its applications. *Discrete Mathematics*, 230(1–3):189–206, 2001.

18 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding Topological Subgraphs is Fixed-parameter Tractable. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 479–488, New York, NY, USA, 2011. ACM.

19 Tibor Grünwald. Ein Neuer Beweis Eines Mengerschen Satzes. *Journal of the London Mathematical Society*, s1-13(3):188–192, 1938. doi:10.1112/jlms/s1-13.3.188.

20 Mamadou Moustapha Kanté and O-joung Kwon. An Upper Bound on the Size of Obstructions for Bounded Linear Rank-Width. *CoRR*, abs/1412.6201, 2014. arXiv:1412.6201.

21 Eun Jung Kim, Sang-il Oum, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. An FPT 2-Approximation for Tree-Cut Decomposition. *Algorithmica*, 80(1):116–135, 2018.

22 Ilhee Kim and Paul D. Seymour. Tournament minors. *Journal of Combinatorial Theory, Series B*, 112:138–153, 2015.

23 Shiva Kintali. Directed Minors III. Directed Linked Decompositions. *CoRR*, 2014. arXiv:1404.5976.

24 Jens Lagergren. Upper Bounds on the Size of Obstructions and Intertwines. *Journal of Combinatorial Theory, Series B*, 73(1):7–40, 1998.

25 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 1(10):96–115, 1927.

26 Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

27 Neil Robertson and Paul Seymour. Graph Minors. XXIII. Nash-Williams' Immersion Conjecture. *Journal of Combinatorial Theory, Series B*, 100(2):181–205, March 2010.

28 Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic Aspects of Tree-Width. *Journal of Algorithms*, 7(3):309–322, 1986.

29 Neil Robertson and Paul D. Seymour. Graph Minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(2):92–114, 1986.

30 Neil Robertson and Paul D Seymour. Graph Minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

31 Robin Thomas. A menger-like property of tree-width: The finite case. *Journal of Combinatorial Theory, Series B*, 48(1):67–76, 1990.

32 Paul Wollan. The structure of graphs not admitting a fixed immersion. *Journal of Combinatorial Theory, Series B*, 110:47–66, 2015.

Dispersing Obnoxious Facilities on a Graph

Alexander Grigoriev
Department of Quantitative Economics, Maastricht University, Maastricht, The Netherlands
a.grigoriev@maastrichtuniversity.nl

Tim A. Hartmann
Department of Computer Science, RWTH Aachen, Aachen, Germany
hartmann@algo.rwth-aachen.de

Stefan Lendl
Institute of Discrete Mathematics, TU Graz, Graz Austria
lendl@math.tugraz.at

Gerhard J. Woeginger
Department of Computer Science, RWTH Aachen, Aachen Germany
woeginger@algo.rwth-aachen.de

Abstract

We study 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 any two facilities have at least distance δ from each other.

We investigate the complexity of this problem in terms of the rational parameter δ. The problem is polynomially solvable, if the numerator of δ is 1 or 2, while all other cases turn out to be NP-hard.

2012 ACM Subject Classification Mathematics of computing; Theory of computation → Graph algorithms analysis; Theory of computation → Discrete optimization

Keywords and phrases algorithms, complexity, optimization, graph theory, facility location

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.33

Related Version A full version of the paper is available at `https://arxiv.org/abs/1811.08918`.

Funding *Stefan Lendl*: Supported by the Austrian Science Fund (FWF): W1230, Doctoral Program "Discrete Mathematics".
Gerhard J. Woeginger: Supported by the DFG RTG 2236 "UnRAVeL".

Acknowledgements The research in this paper has been started during the Lorentz Center Workshop on Fixed-Parameter Computational Geometry, which was organized in May 2018 in Leiden. We thank the Lorentz Center for its hospitality. Alexander Grigoriev and Gerhard Woeginger acknowledge helpful discussions with Fedor Fomin, Petr Golovach and Jesper Nederlof.

1 Introduction

A large part of the facility location literature deals with *desirable* facilities that people like to have nearby, such as service centers, police departments, fire stations, and warehouses. However, there also do exist facilities that are *undesirable* and *obnoxious*, such as nuclear reactors, garbage dumps, chemical plants, military installations, and high security penal institutions. A standard goal in location theory is to spread out such obnoxious facilities and to avoid their accumulation and concentration in a small region; see for instance Erkut & Neuman [6] and Cappanera [2] for comprehensive surveys on this topic.

In this paper, we investigate the location of obnoxious facilities in a metric space whose topology is determined by a graph. Formally, let $G = (V, E)$ be an undirected connected graph, where every edge is rectifiable and has unit length. Let $P(G)$ denote the continuum set

of points on all the edges in E together with all the vertices in V. For two points $p, q \in P(G)$, we denote by $d(p, q)$ the length of a shortest path connecting p and q in the graph. A subset $S \subset P(G)$ is said to be *δ-dispersed* for some positive real number δ, if any two points $p, q \in S$ with $p \neq q$ are at distance $d(p, q) \geq \delta$ from each other. Our goal is to compute for a given graph $G = (V, E)$ and a given positive real number δ a maximum cardinality subset $S \subset P(G)$ that is δ-dispersed. Such a set S is called an *optimal* δ-dispersed set, and $|S|$ is called the *δ-dispersion number* δ-Disp(G) of the graph G.

Known and related results

Obnoxious facility location goes back to the seminal articles of Goldman & Dearing [10] from 1975 and Church & Garfinkel [3] from 1978. The area actually covers a wide variety of problem variants and models; some models specify a geometric setting, while other models use a graph-theoretic setting.

For example, Abravaya & Segal [1] consider a purely geometric variant of obnoxious facility location, where a maximum cardinality set of obnoxious facilities has to be placed in a rectangular region, such that their pairwise distance as well as the distance to a fixed set of demand sites is above a given threshold. As another example we mention the graph-theoretic model of Tamir [15], where every edge $e \in E$ of the underlying graph $G = (V, E)$ is rectifiable and has a given edge-dependent length $\ell(e)$. Tamir discusses the complexity and approximability of various optimization problems with various objective functions. One consequence of [15] is that if the graph G is a tree, then the value δ-Disp(G) can be computed in polynomial time. Segal [14] locates a single obnoxious facility on a network under various objective functions, such as maximizing the smallest distance from the facility to the clients on the network or maximizing the total sum of the distances between facility and clients.

Megiddo & Tamir [13] consider the covering problem that is dual to the δ-dispersion packing problem: Given a graph $G = (V, E)$ with rectifiable unit-length edges, find a minimum cardinality subset $S \subset P(G)$ such that every point in $P(G)$ is at distance at most δ from one of the facilities in S. Among many other results [13] shows that this covering problem is NP-hard for $\delta = 2$.

Our results

We provide a complete picture of the complexity of computing the δ-dispersion number for connected graphs $G = (V, E)$ and positive rational numbers δ.

- If $\delta = 1/b$ for some integer b, then the δ-dispersion number of G can be written down without really looking at the structure of the graph: If G is a tree then δ-Disp$(G) = b|E|+1$, and if G is not a tree then δ-Disp$(G) = b|E|$.
- If $\delta = 2/b$ for some integer b, then δ-Disp(G) can be computed in polynomial time. The algorithm uses the Edmonds-Gallai decomposition of G and reformulates the problem as a submodular optimization problem.
- If $\delta = a/b$ for integers a and b with $a \geq 3$ and $\gcd(a, b) = 1$, then the computation of δ-Disp(G) is an NP-hard problem.

The rest of the paper is organized as follows. Section 2 summarizes the basic notations and states several technical observations. Section 3 presents the NP-hardness results. The reductions are essentially based on routine methods, but need to resolve certain number-theoretic issues. Our technical main contribution is the polynomial time algorithm for the case $\delta = 2$ as developed in Section 4; this result is heavily based on tools from matching theory. Section 5 summarizes the polynomially solvable special cases and provides additional structural insights.

2 Notation and technical preliminaries

All graphs in this paper are undirected and connected, and all edges have unit length. Throughout the paper we use the word *vertex* in the graph-theoretic sense, and we use the word *point* to denote the elements of the geometric structure $P(G)$. For a graph $G = (V, E)$ and a subset $V' \subseteq V$, we denote by $G[V']$ the subgraph induced by V'. For an integer $c \geq 1$, the *c-subdivision* of G is the graph that results from G by subdividing every edge in E by $c - 1$ new vertices into c new edges.

For an edge $e = \{u, v\}$ and a real number λ with $0 \leq \lambda \leq 1$, we denote by $p(u, v, \lambda)$ the point on e that has distance λ from vertex u. Note that $p(u, v, 0) = u$ and $p(u, v, 1) = v$, and note that point $p(u, v, \lambda)$ coincides with point $p(v, u, 1 - \lambda)$; hence we will sometimes assume without loss of generality that $\lambda \leq 1/2$.

▶ **Lemma 2.1.** *Let G be a graph, let $c \geq 1$ be an integer, and let G' be the c-subdivision of G. Then for every $\delta > 0$, the δ-dispersed sets in G are in one-to-one correspondence with the $(c \cdot \delta)$-dispersed sets in G'. In particular, δ-Disp$(G) = (c \cdot \delta)$-Disp(G').*

Proof. Every point $p(u, v, \lambda)$ in $P(G)$ translates into a corresponding point in $P(G')$ that lies on the subdivided edge between u and v and is at distance $c \cdot \lambda$ from vertex u. ◀

Lemma 2.1 has many useful consequences, as for instance the following:

▶ **Lemma 2.2.** *Let $\delta > 0$ and let $c \geq 1$ be an integer.*

- *If the problem of computing the δ-dispersion number is NP-hard, then also the problem of computing the $(c \cdot \delta)$-dispersion number is NP-hard.*
- *If the problem of computing the $(c \cdot \delta)$-dispersion number is polynomially solvable, then also the problem of computing the δ-dispersion number is polynomially solvable.*

Proof. By Lemma 2.1 the c-subdivision of a graph yields a polynomial time reduction from computing δ-dispersions to computing $(c \cdot \delta)$-dispersions. ◀

For integers ℓ and k, the rational number ℓ/k is called *k-simple*. A set $S \subseteq P(G)$ is k-simple, if for every point $p(u, v, \lambda)$ in S the number λ is k-simple.

▶ **Lemma 2.3.** *Let $\delta = a/b$ with integers a and b, and let $G = (V, E)$ be a graph. Then there exists an optimal δ-dispersed set S^* that is $2b$-simple.*

Proof. We first handle the cases with $b = 1$, so that δ is integer. Consider an optimal δ-dispersed set S for graph G. Note that for every vertex u, at most one point $p(u, v, \lambda)$ with $v \in V$ and $0 \leq \lambda < 1/2$ is in S. For every point $p = p(u, v, \lambda)$ with $0 \leq \lambda \leq 1/2$ in S, we put a corresponding point p^* into set S^*: If $0 \leq \lambda < 1/2$ then $p^* = p(u, v, 0)$, and if $\lambda = 1/2$ then $p^* = p(u, v, 1/2)$. As all points in the resulting set S^* are either vertices or midpoints of edges, we get that S^* is 2-simple. We claim that S^* is still δ-dispersed: Consider two distinct points p^* and q^* in S^*. Note that $d(p, p^*) < 1/2$ and $d(q, q^*) < 1/2$ by construction.

- If p^* and q^* both are vertices in V, then the distance $d(p^*, q^*)$ is integer. Suppose for the sake of contradiction that $d(p^*, q^*) < \delta$, which by integrality implies $d(p^*, q^*) \leq \delta - 1$ The triangle inequality yields $d(p, q) \leq d(p, p^*) + d(p^*, q^*) + d(q^*, q)$. The left hand side in this inequality is at least δ, wheras its right hand side is strictly smaller than $(1/2) + (\delta - 1) + (1/2)$. This contradiction shows $d(p^*, q^*) \geq \delta$.
- If p^* and q^* both are midpoints of edges, then $p = p^*$ and $q = q^*$ yields $d(p^*, q^*) \geq \delta$.

- If p^* is the midpoint of some edge and q^* is a vertex, then $d(p^*, q^*) = D + 1/2$ for some integer D. The triangle inequality together with $p = p^*$ implies $\delta \leq d(p, q) = d(p^*, q) \leq d(p^*, q^*) + d(q^*, q) < D + 1$. This implies $D \geq \delta$, so that the desired $d(p^*, q^*) \geq \delta + 1/2$ holds.

Since S and S^* have the same cardinality, we conclude that S^* is an optimal δ-dispersed set that is 2-simple, exactly as desired.

In the cases where $\delta = a/b$ for some integer $b \geq 2$, we consider the b-subdivision G' of G. By the above discussion, G' possesses an optimal a-dispersed set S' that is 2-simple. Then Lemma 2.1 translates S' into an optimal δ-dispersed set S for G that is $2b$-simple. ◀

3 NP-completeness results

In this section we present our NP-hardness proofs for computing the δ-dispersion number. All proofs are done through polynomial time reductions from the following NP-hard variant of the independent set problem; see Garey & Johnson [9].

> Problem: Independent Set in Cubic Graphs (Cubic-Ind-Set)
>
> Instance: An undirected, connected graph $H = (V_H, E_H)$ in which every vertex is adjacent to exactly three other vertices; an integer bound k.
>
> Question: Does H contain an independent set I with $|I| \geq k$ vertices?

Throughout this section we consider a fixed rational number $\delta = a/b$, where a and b are positive integers that satisfy $\gcd(a, b) = 1$ and $a \geq 3$. Section 3.1 discusses the cases with odd numerators $a \geq 3$, and Section 3.2 discusses the cases with even numerators $a \geq 4$. It is instructive to verify that our arguments do not work for the cases with $a = 1$ and $a = 2$, as our gadgets and our arguments break down at various places.

3.1 NP-hard cases with odd numerator

Throughout this section we consider a fixed rational number $\delta = a/b$ where $\gcd(a, b) = 1$ and where $a \geq 3$ is an odd integer. For the NP-hardness proof, we first determine four positive integers x_1, y_1, x_2, y_2 that satisfy the following equations (1) and (2).

$$2b \cdot x_1 - 2a \cdot y_1 = a - 1 \tag{1}$$

$$b \cdot x_2 - a \cdot y_2 = 1 \tag{2}$$

Note that the value $a - 1$ on the right hand side of equation (1) is even, and hence is divisible by the greatest common divisor $\gcd(2b, 2a) = 2$ of the coefficients in the left hand side. With this, Bézout's lemma yields the existence of positive integers x_1 and y_1 that satisfy (1). Bézout's lemma also yields the existence of positive integers x_2 and y_2 in equation (2), as the coefficients in the left hand are relatively prime.

Our reduction now starts from an arbitrary instance $H = (V_H, E_H)$ and k of Cubic-Ind-Set, and constructs a corresponding dispersion instance $G = (V_G, E_G)$ from it.

- For every vertex $v \in V_H$, we create a corresponding vertex v^* in V_G.
- For every edge $e = \{u, v\} \in E_H$, we create a corresponding vertex e^* in V_G.
- For every edge $e = \{u, v\} \in E_H$, we create (i) a path with x_1 edges that connects vertex u^* to vertex e^*, (ii) another path with x_1 edges that connects v^* to e^*, and (iii) a cycle $C(e)$ with x_2 edges that runs through vertex e^*.

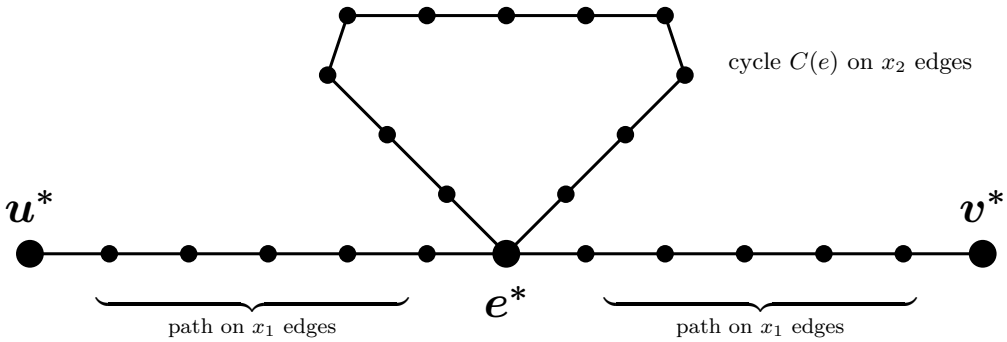


Figure 1 The edge $e = \{u, v\}$ in the instance of Cubic-Ind-Set translates into three vertices u^*, e^*, v^* in the dispersion instance, together with two paths and one cycle.

This completes the description of the graph $G = (V_G, E_G)$; see Figure 1 for an illustration. We claim that graph H contains an independent set of size k, if and only if (a/b)-Disp$(G) \geq k + (2y_1 + y_2)|E_H|$.

▶ **Lemma 3.1.** *If graph H contains an independent set of size k, then the (a/b)-dispersion number of graph G is at least $k + (2y_1 + y_2)|E_H|$.*

Proof. Let I be an independent set of size k in graph $H = (V_H, E_H)$. We construct from I a δ-dispersed set $S \subset P(G)$ as follows. Let $u \in V_H$ be a vertex, and let e_1, e_2, e_3 be the three edges in E_H that are incident to u.

- If $u \in I$, then we put point u^* into S. On each of the three paths that connect vertex u^* respectively to vertex e_i^* $(i = 1, 2, 3)$, we select y_1 further points for S. The first selected point is at distance δ from u^*, and every further selected point is at distance $\delta = a/b$ from the preceding selected point. By equation (1), on each of the three paths the distance from the final selected point to point e_i^* $(i = 1, 2, 3)$ then equals $(a - 1)/(2b)$.
- If $u \notin I$, then on each of the three paths between u^* and e_i^* $(i = 1, 2, 3)$ we select y_1 points for S. The first selected point is at distance $\delta/2 = a/(2b)$ from u^*, and every further selected point is at distance δ from the preceding selected point. By equation (1), the distance from the final selected point to point e^* then equals $(2a - 1)/(2b)$.

Furthermore, for every edge $e \in E_H$ we select y_2 points from the cycle $C(e)$ for S:

- We start in point e^* and traverse $C(e)$ in clockwise direction. The first selected point is at distance $(a + 1)/(2b)$ from point e^*, and every further selected point is at distance δ from the preceding selected point. By equation (2), the distance from the final selected point to point e^* then equals $(a + 1)/(2b)$.

This completes the construction of set S. Now let us count the points in S. First, there are the k points $u^* \in S$ for which $u \in I$. Furthermore, for every edge $e = \{u, v\} \in E_H$ there are $2y_1$ points in S that lie on the two paths from u^* to e^* and from e^* to v^*. Finally, for every edge $e \in E_H$ there are y_2 points that lie on the cycle $C(e)$. Altogether, this yields the desired size $k + (2y_1 + y_2)|E_H|$ for S.

It remains to verify that the point set S is δ-dispersed. By construction, the points selected from each path are at distance at least δ from each other, and the same holds for the points selected from each cycle. If vertex u^* is in S, then all selected points on the three incident paths are at distance at least δ from u^*. If vertex u^* is not in S, then the first selected point on every path is at distance $\delta/2$ from u^*, so that these points are pairwise

at distance at least δ from each other. Hence the only potential trouble could arise in the neighborhood of point e^*, where paths and cycles are glued together. Every selected point on $C(e)$ is at distance at least $(a+1)/(2b)$ from point e^*. Every selected point on some path from u^* to e^* is at distance at least $(a-1)/(2b)$ from e^* if $u \in I$ and is at distance at least $(2a-1)/(2b)$ if $u \notin I$. Since for any edge $e = \{u, v\} \in E_H$ at most one of the end vertices u and v is in I, at most one selected point can be at distance $(a-1)/(2b)$ from e^*, and all other points are at distance at least $(a+1)/(2b)$ from e^*. Hence S is indeed δ-dispersed. ◀

▶ **Lemma 3.2.** *If the (a/b)-dispersion number of graph G is at least $k + (2y_1 + y_2)|E_H|$, then graph H contains an independent set of size k.*

Proof. Let S be an (a/b)-dispersed set of size $k + (2y_1 + y_2)|E_H|$. By Lemma 2.3 we assume that for every point $p(u, v, \lambda)$ in S, the denominator of the rational number λ is $2b$.

For an edge $e = \{u, v\} \in E_H$, let us consider its corresponding path π on x_1 edges that connects vertex u^* to vertex e^*. Suppose that there is some point p in $S \cap \pi$ with $d(p, e^*) \leq (a-2)/(2b)$. Then by Equation (2), set S will contain at most $y_2 - 1$ points from the cycle $C(e)$. In this case we restructure S as follows: We remove point p together with the at most $y_2 - 1$ points on cycle $C(e)$ from S, and instead insert y_2 points into S that are δ-dispersed on $C(e)$ and that all are at distance at least $(a+1)/(2b)$ from e^*. As this restructuring does not decrease the size of S, we will from now on assume without loss of generality that $d(p, e^*) \geq (a-1)/(2b)$ holds for every point $p \in S \cap \pi$.

Now let us take a closer look at the points in $S \cap \pi$. Equation (1) can be rewritten into $x_1 = y_1\delta + (a-1)/(2b)$, which yields $|S \cap \pi| \leq y_1 + 1$.

- In the equality case $|S \cap \pi| = y_1 + 1$, we must have $u^* \in S$ and also the point on π at distance $(a-1)/(2b)$ from e^* must be in S.
- In case $|S \cap \pi| \leq y_1$, there is ample space for picking y_1 points from π that are δ-dispersed and that are at distance at least $\delta/2$ from u^* and at distance at least $\delta/2$ from e^*. Hence we will from now on assume $|S \cap \pi| = y_1$ in these cases.

Now let us count: Set S contains exactly y_1 interior points from every path π, and altogether there are $2|E_H|$ such paths. Set S contains exactly y_2 points from every cycle $C(e)$, and altogether there are $|E_H|$ such cycles. Since $|S| \geq k + (2y_1 + y_2)|E_H|$, this means that S must contain at least k further points on vertices u^* with $u \in V_H$. The corresponding subset of V_H is called I.

Finally, we claim that this set I with $|I| \geq k$ forms an independent set in graph H. Suppose for the sake of contradiction that there is an edge $e = \{u, v\} \in E_H$ with $u^* \in I$ and $v^* \in I$. Consider the two paths that connect u^* to e^* and v^* to E^*. By the above discussion, S then contains two points at distance $(a-1)/(2b)$ from e^*. As these two points are then at distance at most $(a-1)/b < \delta$ from each other, we arrive at the desired contradiction. ◀

The statements in Lemma 3.1 and in 3.2 yield the following theorem.

▶ **Theorem 3.3.** *Let a and b be positive integers with $\gcd(a, b) = 1$ and odd $a \geq 3$. Then it is NP-hard to compute the (a/b)-dispersion number of a graph G.*

3.2 NP-hard cases with even numerator

In this section we consider a fixed rational number $\delta = a/b$ where $\gcd(a, b) = 1$ and where $a \geq 4$ is an even integer. The NP-hardness argument is essentially a minor variation of the argument in Section 3.1 for the cases with odd numerators. Therefore, we will only explain the modifications, and leave all further details to the reader.

The NP-hardness proof in Section 3.1 is centered around the four positive integers x_1, y_1, x_2, y_2 introduced in equations (1) and (2). We perform the same reduction from Cubic-Ind-Set as in Section 3.1 but with positive integers x_1, y_1, x_2, y_2 that satisfy the following equations (3) and (4).

$$2b \cdot x_1 - 2a \cdot y_1 = a - 2 \quad (3)$$

$$b \cdot x_2 - a \cdot y_2 = 2 \quad (4)$$

In (3), the right hand side $a-2$ is even and divisible by the greatest common divisor of the coefficients in the left hand side. In (4), the coefficients in the left hand are relatively prime. Therefore Bézout's lemma can be applied to both equations.

The graph $G = (V_G, E_G)$ is defined as before, with a vertex v^* for every $v \in V_H$ and a vertex e^* for every $e \in E_H$, with paths on x_1 edges and cycles $C(e)$ on x_2 edges. The arguments in Lemma 3.1 and 3.2 can easily be adapted and yield the following theorem.

▶ **Theorem 3.4.** *Let a and b be positive integers with $\gcd(a,b) = 1$ and even $a \geq 4$. Then it is NP-hard to compute the (a/b)-dispersion number of a graph G.*

3.3 Containment in NP

In this section we consider the decision version of δ-dispersion: *"For a given graph $G = (V, E)$, a positive real δ, and a bound k, decide whether δ-Disp$(G) \leq k$."* Our NP-certificate specifies the following partial information on a δ-dispersed set S in a graph $G = (V, E)$:

- The certificate specifies the set $W := V \cap S^*$.
- For every edge $e \in E$, the certificate specifies the number n_e of facilities that are located in the interior of e.

As every edge accommodates at most $1/\delta$ points from S, the encoding length of our certificate is polynomially bounded in the instance size. For verifying the certificate, we introduce for every vertex u and for every incident edge $e = \{u, v\} \in E$ with $n_e > 0$ a corresponding real variable $x(u, e)$, which models the distance between vertex u and the closest point from S in the interior of edge e. Finally, we introduce the following linear constraints:

- The non-negativity constraints $x(u, e) \geq 0$.
- For every edge $e = \{u, v\} \in E$, the inequality $x(u, e) + (n_e - 1)\delta + x(v, e) \leq 1$.
- For all $u, v \in W$ with $u \neq v$, the inequality $d(u, v) \geq \delta$.
- For all $w \in W$ and $e = \{u, v\} \in E$, the inequality $x(u, e) + d(u, w) \geq \delta$.
- For all $e = \{u, v\} \in E$ and $e' = \{u', v'\} \in E$, the inequality $x(u, e) + d(u, u') + x(u', e') \geq \delta$.

These inequalities enforce that on every edge the variables properly work together, and that the underlying point set indeed is δ-dispersed. For verifying the certificate, we simply check in polynomial time whether the resulting linear program has a feasible solution, and whether $|W| + \sum_{e \in E} n_e \geq k$ holds.

▶ **Theorem 3.5.** *The decision version of δ-dispersion lies in NP, even if the value δ is given as part of the input.*

4 The polynomial time result for $\delta = 2$

This section derives a polynomial time algorithm for computing the 2-dispersion number of a graph. This algorithm is heavily based on tools from matching theory, as for instance developed in the book by Lovász & Plummer [12]. As usual, the size of a maximum cardinality matching in graph G is denoted by $\nu(G)$.

▶ **Lemma 4.1.** *Every graph $G = (V, E)$ satisfies* 2-Disp$(G) \geq \nu(G)$.

Proof. The midpoints of the edges in every matching form a 2-dispersed set. ◀

A 2-dispersed set is in *canonical* form, if it entirely consists of vertices and of midpoints of edges. Recall that by Lemma 2.3 every graph $G = (V, E)$ possesses an optimal 2-dispersed set in canonical form. Throughout this section, we will consider 2-dispersed (but not necessarily optimal) sets S^* in canonical form; we always let V^* denote the set of vertices in S^*, and we let E^* denote the set of edges whose midpoints are in S^*. Finally, $N^* \subseteq V$ denotes the set of vertices in $V - V^*$ that have a neighbor in V^*. As S^* is 2-dispersed, the vertex set V^* forms an independent set in G, and the edge set E^* forms a matching in G. Furthermore, the vertex set N^* separates the vertices in V^* from the edges in E^*; in particular, no edge in E^* covers any vertex in N^*. We start with two technical lemmas that will be useful in later arguments.

▶ **Lemma 4.2.** *Let $G = (V, E)$ be a graph with a perfect matching, and let S^* be some 2-dispersed set in canonical form in G. Then $|S^*| \leq \nu(G)$.*

Proof. Let $M \subseteq E$ denote a perfect matching in G, and for every vertex $v \in V$ let $e(v)$ denote its incident edge in matching M. Consider the vertex set V^* and the edge set E^* that correspond to set S^*. Then E^* together with the edges $e(v)$ with $v \in V^*$ forms another matching M' of cardinality $|E^*| + |V^*| = |S^*|$ in G. Now $|S^*| = |M'| \leq \nu(G)$ yields the desired inequality. ◀

A graph G is *factor-critical* [12], if for every vertex $x \in V$ there exists a matching that covers all vertices except x. A *near-perfect* matching in a graph covers all vertices in V except one. Note that the statement in the following lemma cannot be extended to graphs that consist of a single vertex.

▶ **Lemma 4.3.** *Every 2-dispersed set S^* in a factor-critical graph $G = (V, E)$ with $|V| \geq 3$ satisfies $|S^*| \leq \nu(G)$.*

Proof. Without loss of generality we assume that S^* is in canonical form, and we let V^* and E^* denote the underlying vertex set and edge set, respectively. If V^* is empty, we have $|S^*| = |E^*| \leq \nu(G)$ since E^* is a matching. If V^* is non-empty, then also N^* is non-empty (here we use the condition $|V| \geq 3$) and we pick some vertex $x \in N^*$. We consider a near-perfect matching M that covers all vertices except x, and we let $e(v)$ denote the edge incident to $v \in V$ in matching M. Then E^* together with the edges $e(v)$ with $v \in V^*$ forms another matching M' of cardinality $|E^*| + |V^*| = |S^*|$ in G. The claim follows from $|S^*| = |M'| \leq \nu(G)$. ◀

The following theorem goes back to Edmonds [5] and Gallai [7, 8]; see also Lovász & Plummer [12]. Figure 2 gives an illustration.

▶ **Theorem 4.4.** *(Edmonds-Gallai structure theorem) Let $G = (V, E)$ be a graph. The following decomposition of V into three sets X, Y, Z can be computed in polynomial time.*

$$\begin{aligned} X &= \{v \in V \mid \textit{ there exists a maximum matching that misses } v\} \\ Y &= \{v \in V \mid \textit{ } v \notin X \textit{ and } v \textit{ is adjacent to some vertex in } X\} \\ Z &= V - (X \cup Y) \end{aligned}$$

The Edmonds-Gallai decomposition has the following properties:

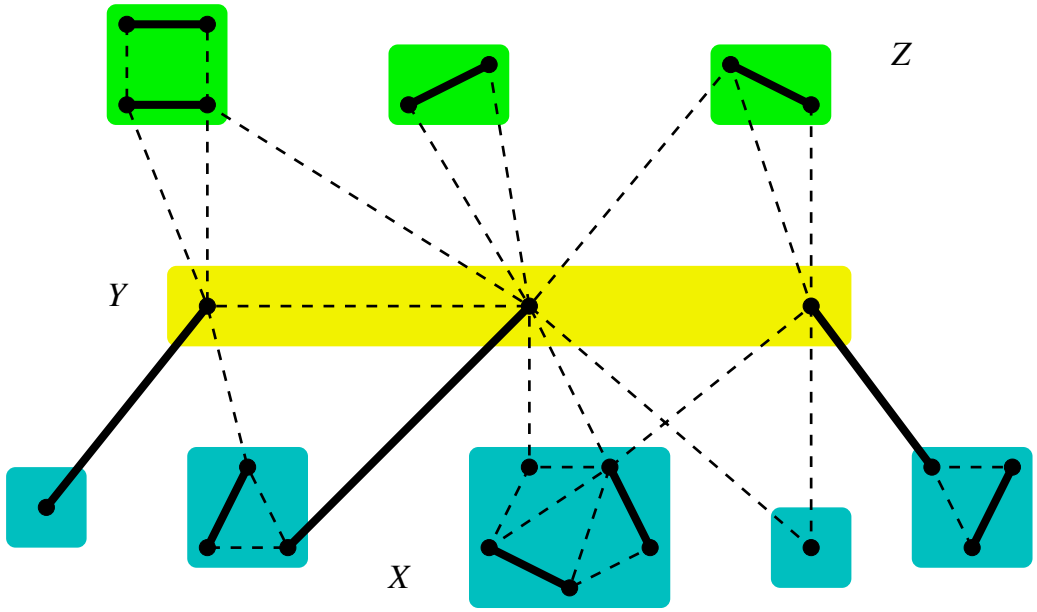


Figure 2 An illustration for the Edmonds-Gallai structure theorem. A maximum matching is shown with fat edges, and the non-matching edges are dashed.

- *Set X is the union of the odd-sized components of $G - Y$; every such odd-sized component is factor-critical. Set Z is the union of the even-sized components of $G - Y$.*
- *Every maximum matching in G induces a perfect matching on every (even-sized) component of Z and a near-perfect matching on every (odd-sized) component of X. Furthermore, the matching matches the vertices in Y to vertices that belong to $|Y|$ different components of X.*

We further subdivide the set X in the Edmonds-Gallai decomposition into two parts: Set X_1 contains the vertices of X that belong to components of size 1, and set $X_{\geq 3}$ contains the vertices that belong to (odd-sized) components of size at least 3. The *vicinity* $\mathrm{vic}(v)$ of a vertex $v \in V$ consists of vertex v itself and of the midpoints of all edges incident to v.

▶ **Lemma 4.5.** *There exists an optimal 2-dispersed set S^* in canonical form (with underlying edge set E^*) that additionally satisfies the following three properties.*

- **P1.** *In every component of $X_{\geq 3}$, the set E^* induces a near-perfect matching.*
- **P2.** *For every vertex $y \in Y$, the set $\mathrm{vic}(y) \cap S^*$ is either empty or consists of the midpoint of some edge between X and Y.*
- **P3.** *In every component of Z, the set E^* induces a perfect matching.*

Proof. We start from an arbitrary optimal 2-dispersed set S^* (in canonical form, with corresponding sets V^* and E^*) and transform it in two steps into an optimal 2-dispersed set of the desired form.

In the first transformation step, we exploit a matching M between sets Y and X that matches every vertex $y \in Y$ to some vertex $M(y)$, so that for $y_1 \neq y_2$ the vertices $M(y_1)$ and $M(y_2)$ belong to different components of X; see Theorem 4.4. A vertex $y \in Y$ is called *blocked*, if it is adjacent to some $x \in X_1 \cap S^*$. As for a blocked vertex the set $\mathrm{vic}(y) \cap S^*$ is already empty (and hence already satisfies property P2), we will not touch it at the moment. We transform S^* in the following way.

- For every non-blocked vertex $y \in Y$, the set $\text{vic}(y) \cap S^*$ contains at most one point. We remove this point from S^*, and we insert instead the midpoint of the edge between y and $M(y)$ into S^*. These operations cannot decrease the size of S^*.
- Every (odd-sized) component C of $X_{\geq 3}$ contains at most one point $M(y)$ with $y \in Y$. We compute a near-perfect matching M_C for C that misses this vertex $M(y)$ (and if no such vertex is in C, matching M_C misses an arbitrary vertex of C). We remove all points in C from S^*, and we insert instead the midpoints of the edges in M_C. As by Lemma 4.3 we remove at most $\nu(C)$ points and as we insert exactly $\nu(C)$ points, these operations will not decrease the size of S^*.

The resulting set S^* is of course again in canonical form, and it is also easy to see that S^* is still 2-dispersed. Furthermore, S^* now satisfies properties P1 and P2.

In the second transformation step, we note that the current S^* does neither contain vertices from Y nor midpoints of edges between Y and Z. For every (even-sized) component C of Z, we compute a perfect matching M_C. We remove all points in C from S^*, and we insert instead the midpoints of the edges in M_C. As by Lemma 4.3 we remove at most $\nu(C)$ points and as we insert exactly $\nu(C)$ points, these operations will not decrease the size of S^*. The resulting set S^* is 2-dispersed and satisfies properties P1, P2, and P3. ◀

The optimal 2-dispersed sets in Lemma 4.5 are strongly structured and fairly easy to understand: The perfect matchings in set Z contribute exactly $|Z|/2$ points to S^*. Every (odd-sized) component C in $X_{\geq 3}$ contributes exactly $(|C|-1)/2$ points to S^*. The only remaining open decisions concern the points in X_1 and the midpoints of the edges $\{y, M(y)\}$ for $y \in Y$. So let us consider the set $T := S^* \cap X_1$, and let $\Gamma(T) \subset Y$ denote the vertices in Y that are adjacent to some vertex in T. Then every vertex y in $Y - \Gamma(T)$ contributes the midpoint of $\{y, M(y)\}$ to S^*, and every vertex $x \in T$ contributes itself to S^*.

Hence the remaining optimization problem boils down to finding a subset $T \subseteq X_1$ that maximizes the function value $f(T) := |Y - \Gamma(T)| + |T|$, which is equivalent to minimizing the function value

$$g(T) := |\Gamma(T)| - |T|. \tag{5}$$

The set function $g(T)$ in (5) is a *submodular* function, as it satisfies $g(A)+g(B) \geq g(A \cup B)+g(A \cap B)$ for all $A, B \subseteq X_1$; see for instance Grötschel, Lovász & Schrijver [11]. Therefore, the minimum value of $g(T)$ can be determined in polynomial time by the ellipsoid method [11], or by Cunningham's combinatorial algorithm [4].

We finally summarize all our insights and formulate the main result of this section.

▶ **Theorem 4.6.** *The 2-dispersion number of a graph G can be computed in polynomial time.*

5 The polynomially solvable cases

Theorem 4.6 and Lemma 2.2 together imply that for every rational number $\delta = a/b$ with numerator $a \leq 2$, the δ-dispersion number of a graph can be computed in polynomial time. We now present some results that provide additional structural insights into these cases. The cases where the numerator is $a = 1$ are structurally trivial, and the value of the corresponding δ-dispersion number can be written down with the sole knowledge of $|V|$ and $|E|$.

The proofs for the results of this section can be found in the full version of this paper.

▶ **Lemma 5.1.** *Let $\delta = 1/b$ for some integer b, and let $G = (V, E)$ be a connected graph.*

- *If G is a tree then δ-Disp$(G) = b|E| + 1$.*
- *If G is not a tree then δ-Disp$(G) = b|E|$.*

The following lemma derives an explicit (and very simple) connection between the 2-dispersion number and the $(2/b)$-dispersion number (with odd denominator b) of a graph.

▶ **Lemma 5.2.** *Let $G = (V, E)$ be a graph, let $z \geq 1$ be an integer, and let $\delta = 2/(2z + 1)$. Then the dispersion numbers satisfy* $\delta\text{-Disp}(G) = 2\text{-Disp}(G) + z|E|$.

References

1 S. Abravaya and M. Segal. Maximizing the number of obnoxious facilities to locate within a bounded region. *Computers and Operations Research*, 37:163–171, 2010.
2 P. Cappanera. A survey on obnoxious facility location problems. Technical report, Dipartimento di Informatica, Universitá di Pisa, Italy, 2010.
3 R.L. Church and R.S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12:107–118, 1978.
4 W.R. Cunningham. On submodular function minimization. *Combinatorica*, 5:185–192, 1985.
5 J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
6 E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40:275–291, 1989.
7 T. Gallai. Kritische Graphen II. *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 8:373–395, 1963.
8 T. Gallai. Maximale Systeme unabhängiger Kanten. *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 9:401–413, 1964.
9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
10 A.J. Goldman and P.M. Dearing. Concepts of optimal location for partially noxious facilities. *ORSA Bulletin*, 23:B–31, 1975.
11 M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Verlag, 1988.
12 L. Lovász and M.D. Plummer. *Matching Theory*. Annals of Discrete Mathematics 29, North-Holland, Amsterdam, 1986.
13 N. Megiddo and A. Tamir. New results on the complexity of p-center problems. *SIAM Journal on Computing*, 12:751–758, 1983.
14 M. Segal. Placing an obnoxious facility in geometric networks. *Nordic Journal of Computing*, 10:224–237, 2003.
15 A. Tamir. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4:550–567, 1991.

Reachability in $O(\log n)$ Genus Graphs is in Unambiguous Logspace

Chetan Gupta
Indian Institute of Technology, Kanpur, India
gchetan@cse.iitk.ac.in

Vimal Raj Sharma
Indian Institute of Technology, Kanpur, India
vimalraj@cse.iitk.ac.in

Raghunath Tewari
Indian Institute of Technology, Kanpur, India
rtewari@cse.iitk.ac.in

Abstract

We show that given an embedding of an $O(\log n)$ genus graph G and two vertices s and t in G, deciding if there is a path from s to t in G is in unambiguous logarithmic space.

Unambiguous computation is a restriction of nondeterministic computation where the nondeterministic machine has at most one accepting computation path on each input. An important fundamental question in computational complexity theory is whether this is an actual restriction or are unambiguous computations as powerful as general nondeterminism. We investigate this problem in the domain of logarithmic space bounded computations, where the corresponding unambiguous and general nondeterministic classes are UL and NL respectively.

In 1997 Reinhardt and Allender showed that NL and UL are equal in a non-uniform model. More specifically they showed that if one can efficiently construct an $O(\log n)$-bit min-unique weight function for a graph, then these classes are equal unconditionally as well. In other words, they gave a UL algorithm to solve reachability in graphs with a min-unique weight assignment. Using this approach reachability in various classes of graphs such as planar graphs, constant genus graphs, minor free graphs, etc., have been shown to be in UL by devising min-unique weight functions for those classes.

In this paper we improve these results by constructing a min-unique weight function for $O(\log n)$ genus graphs. We define signature of a path in a graph as the parity of the number of crossings of that path with respect to each handle of the surface on which the graph is embedded. We construct our weight function in two steps. First we ensure that between any pair of vertices, amongst all paths having the same signature, the minimum weight path is unique. Now since in a genus g graph there are 2^{2g} many possible signatures, we use the hashing scheme of Fredman, Komlós and Szemerédi to isolate a unique minimum weight path among these 2^{2g} many paths isolated in the first step.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases logspace unambiguity, high genus, path isolation

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.34

Acknowledgements The authors would like to thank Ministry of Electronics and IT, India for supporting this research through the Visvesvaraya PhD and YFRF program. The third author would also like to thank DST for providing funding support.

1 Introduction

Deciding reachability between a pair of vertices in a graph is an important problem in computational complexity theory. Directed graph reachability characterizes the complexity of the class nondeterministic logspace (NL) and undirected graph reachability characterizes the

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 34; pp. 34:1–34:13

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


complexity of the class deterministic logspace (L). The latter follows due to a seminal result by Reingold in 2005 [24]. Several other variants of this problem characterize the complexity of other complexity classes [6, 13, 7].

Unambiguous computation is a natural restriction of nondeterministic computation where for every input the Turing machine can have at most one accepting computation path. In the domain of logarithmic space, this defines the class unambiguous logspace (UL) of languages for which there are nondeterministic logspace bounded Turing machines that have exactly one accepting path for every input in the language and zero accepting path otherwise. The class UL was introduced in [9] and subsequently its properties were also studied in the paper [4]. The relation between NL and UL was not well understood till that point. In 1997, Reinhardt and Allender showed that NL and UL are equal in a non-uniform setting [25]. Subsequently, it was shown that if deterministic linear space functions cannot be computed by $2^{\epsilon n}$ sized circuits then NL = UL [3]. Both these results gave evidence that most likely the classes are the same unconditionally. Recently directed graph reachability was shown to be decidable by an unambiguous algorithm running in polynomial time and using $O(\log^2 n)$ space [20]. The space bound was improved to $O(\log^{1.5} n)$ in a subsequent result [29].

A graph G is said to be *min-unique* with respect to a weight function w if for every pair of vertices in G there is at most one minimum weight path from one vertex to the other with respect to w. We will call such a weight function a *path isolating* weight function. Min-uniqueness has been studied in several papers [30, 16, 25]. Reinhardt and Allender showed that if graphs in a class of graphs are min-unique with respect to an $O(\log n)$ bit weight function then deciding reachability for that class of graphs is in UL [25]. They also gave a UL algorithm to check if a graph is min-unique.

The technique of assigning weights to isolate a combinatorial structure is not specific to the graph reachability problem and has been applied to other computational problems as well. In a sequence of results it has been shown that assigning an efficiently computable weight function to a graph class such that the minimum weight perfect matching is unique with respect to the weight function, results in an efficient parallel algorithm for computing matching for the respective class of graphs [11, 12, 5, 14]. Another area which has successfully used the idea of isolating structures to obtain better upper bounds is polynomial identity testing. More specifically, researchers have used basis isolating weight assignment to obtain polynomial and quasi-polynomial time algorithms for certain restrictions of the polynomial identity testing problem [1, 19, 18].

Observe that devising a UL algorithm for directed graph reachability would show that NL = UL, since directed graph reachability is complete for the class NL. Although the NL versus UL has been elusive so far, partial progress has been made towards solving this problem. For several classes of directed graphs, the reachability problem has been shown to be in UL – such as layered grid graphs [2], planar graphs [8], constant genus graphs [21, 12], graphs with polynomially many paths from the source to all other vertices [23], $K_{3,3}$-free and K_5-free graphs [27, 5]. The techniques involve either an efficient construction of a path isolating weight function or reduction to reachability in a graph class for which the problem is already known to be in UL.

Reachability in positive genus graphs is a natural extension of planar reachability. Allender et al. showed that reachability in 1 genus graphs can be reduced to planar reachability [2]. After planar reachability was shown to be in UL, reachability in constant genus graphs was reduced to reachability in planar graphs [21]. Later a path isolating weight function was also given for constant genus graphs [12]. Prior to our result, the best known nondeterministic space upper bound for reachability in non-constant genus graphs was nothing better than general directed graphs. The question of whether reachability in $\omega(1)$ genus graphs belongs to UL or not has been open for almost a decade.

1.1 Our Result

In this paper, we make progress towards understanding the space complexity of directed graph reachability and show the following result.

▶ **Theorem 1.** *Given a polygonal schema of an $O(\log n)$ genus directed graph G, deciding reachability in G is in* $\mathsf{UL} \cap \mathsf{coUL}$.

Given a genus g graph, in the first stage, we give an $O(\log n)$ bit weight function w_{pl} which is essentially the same weight function as defined in [26] and another weight function w_{len} which gives weight 1 to every edge in the graph. Weight function w_{len} ensures that minimum weight paths among all pairs of vertices are of minimum length as well. We then show that between every pair of vertices in the graph, the number of minimum weight "topologically unequivalent" paths is at most $2^{O(g)}$. For this, we define a notion called *signature* which allows us to classify topologically equivalent paths. We show that topologically equivalent paths are very similar to paths in planar graphs and therefore we can borrow the machinery for path isolation in planar graphs here as well. In the second stage, we use the hashing scheme of Fredman, Komlós and Szemerédi [15] to compute an $O(\log n + g)$ bit weight function w_{fks} with respect to which only one among the $2^{O(g)}$ many paths of the first stage gets the minimum weight value.

When g is $O(\log n)$ the number of such minimum weight paths produced in the first stage is at most polynomial in n. Thereafter by combining the weight functions w_{len}, w_{pl} and w_{fks} we get an $O(\log n)$ bit weight function with respect to which the graph is min-unique. We then apply Reinhardt and Allender's algorithm to get a UL algorithm for $O(\log n)$ genus reachability.

1.2 Organization of the Paper

The rest of the paper is organized as follows. In Section 2 we define the notations and framework of our problem. We discuss different representations of high genus graphs and how to efficiently obtain a representation that is suitable for our purpose. We also state results from earlier work that we use in this paper. In Section 3 we prove the main result by giving a min-unique weight function.

2 Preliminaries

Let $G = (V, E)$ be a directed graph on n vertices and m edges. Let uv denote an edge directed from u to v. A *weight function* is a map $w : E \to \mathbb{Z}$ which maps every edge in G to an integer. A weight function w is said to be *skew-symmetric* if for every edge uv, $w(uv) = -w(vu)$. For a set of edges S, $w(S) = \sum_{e \in S} w(e)$. We can think of different structures in a graph such as path, walk, cycle as sets of edges and define the weight of the structure accordingly.

2.1 Representation of High Genus Graph

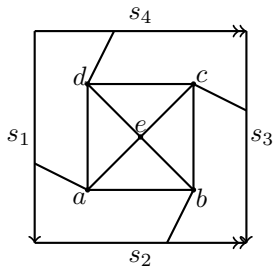
A genus g surface is a sphere with g handles on it. The *genus of a graph* is the minimum genus surface on which the graph can be embedded without any edge crossings. Such an embedding is also called a *2-cell embedding*. Since we are dealing with graphs embedded on surfaces, it is important to specify how the input graph is represented. Given a graph, computing its genus is NP-hard [28]. To the best of our knowledge, no PTAS is known either to compute the genus of a graph. So in accordance with the convention followed by

earlier papers that deals with problems on bounded genus graphs, we also assume a suitable representation of the input graph [22, 17]. We use a representation similar to the one used by Mahajan and Varadarajan [22].

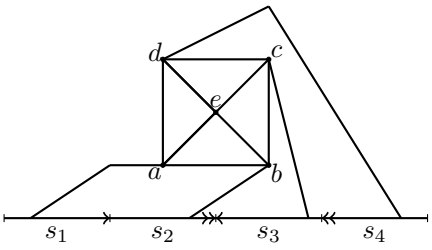
Given a genus g graph G we consider an embedding of G inside a polygon S with $4g$ sides, $s_1, s_2, \ldots, s_{4g}$. We refer to these as the *segments* of S. Moreover, we assume there is no vertex on the boundary of the polygon. The segments s_{4k+1} and s_{4k+2} are directed in anti-clockwise and segments s_{4k+3} and s_{4k+4} are directed in clockwise direction. The segments s_{4k+1} and s_{4k+3} form a pair together such that an edge can come into one of them and go out from another. Similarly, segments s_{4k+2} and s_{4k+4} form a pair. Also, if an edge is the jth edge crossing a segment s_i from head to tail then it will be the jth edge crossing the paired segment of s_i from tail to head. Pairs (s_{4k+1}, s_{4k+3}) and (s_{4k+2}, s_{4k+4}) together constitute the ith handle of the sphere. We assume that we are provided with the combinatorial embedding of the graph G inside S and an ordering of the edges crossing each segment s_i. We also assume without loss of generality that an edge can cross a segment of the polygonal schema at most once. This is because an edge crossing multiple segments can be subdivided into several edges in logspace so that reachability is still preserved. We call this representation the *polygonal schema* of G.

Let E_S be the set of edges in G that cross some segment s_i. Then observe that $G_{\text{planar}} = G \setminus E_S$ is a planar graph. A *piecewise straight line embedding* of a planar graph is an embedding where vertices are integral coordinates and an edge is a piecewise straight line segment connecting its two end points such that no two edges intersect. Given the combinatorial embedding of a planar graph a piecewise straight line embedding of it can be constructed in logspace such that each edge consists of at most 4 segments [26].

For a genus g graph G, a *flat schema* is an embedding of G such that the polygon S is represented as a straight line segment parallel to the x-axis, the internal planar graph G_{planar} is given as a piecewise straight line embedding and each edge in E_S is drawn as a piecewise straight line segment such that no two edges cross each other. Moreover, all vertices and points where an edge crosses a segment are integral coordinates. See Figure 1 for an example of a flat schema of K_5.



(a) Polygonal schema of K_5.



(b) Flat schema of K_5.

Figure 1 Embeddings of K_5.

Given a polygonal schema of G, we can compute a piecewise straight line embedding of G_{planar} in logspace. Now using a similar idea we draw each edge in E_S as a piecewise straight line segment from its end vertices to the corresponding segments of S. We summarize this process in Lemma 2.

▶ **Lemma 2.** *Given a polygonal schema of a graph G with $4g$ segments we can construct a flat schema of G with $4g$ segments, having polynomially bounded integer coordinates in logspace.*

The weight function w_{pl} that we define in Section 3 is similar to the weight function in [26]. That is why we need edges to be piecewise straight line segments. Also the flat schema embedding is necessary because we want all edges parallel to the x-axis to have weight 0 with respect to w_{pl}.

2.2 Previous Work

Consider a genus g graph G embedded on a surface of genus g say Γ. A simple cycle C in G is called a *separating cycle* if cutting along C divides the surface into at least two parts. Otherwise C is called a *non-separating cycle*. We state a characterization of these cycles from Lemma 4 of Cabello and Mohar [10] and Lemma 10 of Datta et al. [12].

▶ **Theorem 3** ([10, 12]). *Consider a polygonal schema of a genus g graph. A cycle C in G is said to be surface separating if and only if C crosses each segment of the polygonal schema an even number of times. Moreover, if C is surface separating then with respect to each segment s_i, the cycle C alternates between coming into s_i and going out of it (if C crosses s_i at all).*

We next state the popular hashing result by Fredman, Komlós and Szemerédi.

▶ **Theorem 4** ([15]). *Let $S = \{x_1, x_2, \ldots, x_k\}$ be a set of $n-$bit integers. Then there exists a $O(\log n + \log k)$ bit prime number p so that for all $x_i \neq x_j \in S$, $x_i \bmod p \neq x_j \bmod p$.*

In Theorem 5 we state a slightly modified version of Reinhardt and Allender's result that would be useful for our purpose.

▶ **Theorem 5** ([25]). *There is a nondeterministic logspace Turing machine M that takes a tuple $\langle G, s, t, w\rangle$ as input where G is a directed graph on n vertices, s and t are vertices in G and w is an $O(\log n)$ bit edge weight function and outputs the following along a unique computation path while all other computation paths halt and reject:*

- Not Min-unique *if G is not min-unique with respect to w,*
- Yes *if G is min-unique with respect to w and there is a path from s to t in G, and*
- No *if G is min-unique with respect to w and there is no path from s to t in G.*

Finally, in Theorem 6 we state the relation between the area of a simple cycle in a planar graph and weight of the cycle with respect to a suitable weight function as shown by Tewari and Vinodchandran.

▶ **Theorem 6** ([26]). *Given a piece-wise straight line embedding of a planar graph G, there exists a logspace computable weight function w such that for any cycle C in G, we have $w(C) = 2 \cdot \text{Area}(C)$ if C is a counter-clockwise cycle and $w(C) = -(2 \cdot \text{Area}(C))$ if C is a clockwise cycle, where $\text{Area}(C)$ is the area of the region enclosed by C.*

3 Isolating Paths in High Genus Graphs

In this section we show that graphs of logarithmic genus are min-unique with respect to an $O(\log n)$-bit weight function that can be computed by an unambiguous logspace machine. Using this weight function in combination with Theorem 5 we get a $\mathsf{UL} \cap \mathsf{coUL}$ algorithm for directed graph reachability in $O(\log n)$ genus graphs. Theorem 7 is the main technical result of this paper where we show the existence and computability of such a weight function.

▶ **Theorem 7.** *Given a genus g directed graph $G = (V, E)$ in terms of its flat schema, there exists an $O(\log n + g)$ bit weight function $w : E \to \mathbb{Z}$, such that for every $u, v \in V$, there exists a unique minimum weight path from u to v with respect to w, if v is reachable from u. Moreover, there is a nondeterministic $O(\log n + g)$ space algorithm that given G as input, outputs the weight function w along a unique computation path while all other paths halt and reject.*

Let $\mathcal{S} = \{s_1, s_2, \ldots, s_{4g}\}$ be the set of segments of the flat schema of G. We define a skew-symmetric weight function w_{pl} that gives non-zero weight to every surface separating cycle in G. For edges which do not cross any segment of the flat schema (we refer to them as *planar edges*), w_{pl} is same as the weight function defined in [26], and for edges which do cross some segment of the flat schema (we refer to them as *crossing edges*) we modify the weight function to be the sum of the weights of the two line segments of the edge. Formally, for an edge $e = uv$ we define the $w_{\text{pl}}(e)$ as:

$$w_{\text{pl}}(e) = \begin{cases} (y_{v_e} - y_{u_e})(x_{v_e} + x_{u_e}) & \text{if } e \text{ is a planar edge} \\ (y_{u'_e} - y_{u_e})(x_{u'_e} + x_{u_e}) + (y_{v_e} - y_{v'_e})(x_{v_e} + x_{v'_e}) & \text{if } e \text{ is a crossing edge} \end{cases}$$

where (x_{u_e}, y_{u_e}) and (x_{v_e}, y_{v_e}) are the coordinates of u and v respectively and $(x_{u'_e}, y_{u'_e})$ and $(x_{v'_e}, y_{v'_e})$ are the coordinates of intersection points of edge e with segments s_i and s_j respectively, assuming that edge e comes into s_i and goes out of s_j.

We also define another weight function w_{len} that assigns value one to every edge in the graph. That is $w_{\text{len}}(e) = 1$ for every edge $e \in G$. Let $w_{\text{comb}} = w_{\text{len}} \cdot n^{k_1} + w_{\text{pl}}$ be the weight function defined by combining w_{pl} and w_{len} for a large enough constant k_1. As a result the minimum weight path with respect to w_{comb} also has the minimum length.

We first show that every surface separating cycle has non-zero weight with respect to w_{pl}. The idea is to decompose every surface separating cycle into a set of planar cycles having the same orientation such that the weight of the original cycle is the sum of the weights of the planar cycles.

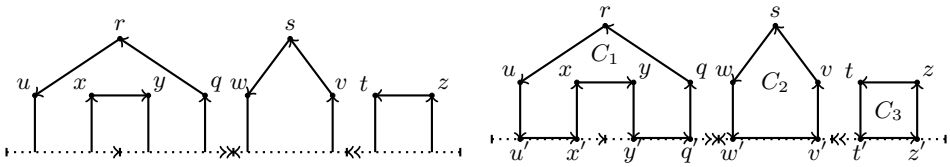
▶ **Lemma 8.** *Let C be a simple surface separating cycle of length at least 3 in G, then $w_{pl}(C) \neq 0$.*

Proof. A surface separating cycle can be of two types – one which does not intersect with any segment of the flat schema and the one which does. If C does not intersect any segment of the flat schema then C is a planar cycle. Hence $w_{\text{pl}}(C) \neq 0$ by [26].

Now consider the case where some edges of C cross the flat schema. From Theorem 3 we know that since C is a surface separating cycle, therefore, C alternates between going out and coming into the segments of the flat schema. Without loss of generality assume that the first edge of C crossing the flat schema going left to right, is coming into it. The other case is analogous.

For every edge $e = uv$ which crosses the boundary S of the flat schema we subdivide e into two directed edges $u_e u'_e$ and $v'_e v_e$, such that u'_e is the point at which uv comes into some segment s_i and v'_e is the point at which uv goes out of some segment s_j. Let C' be the cycle corresponding to C formed by this subdivision. By definition of w_{pl} we have $w_{\text{pl}}(uv) = w_{\text{pl}}(u_e u'_e) + w_{\text{pl}}(v'_e v_e)$ and hence $w_{\text{pl}}(C) = w_{\text{pl}}(C')$.

Let $x_1, x_2, \ldots x_{2t}$ be the set of intersection points of C' and S ordered from left to right. Add t dummy directed edges from x_{2i-1} to x_{2i} for all $1 \leq i \leq t$. This decomposes C' into a set of disjoint planar cycles $C_1, C_2, \ldots, C_k$ such that each C_i has the same orientation (counter-clockwise, since we assume the first edge is coming into S). See Figure 2 for an example.



(a) Surface separating cycle C.

(b) Decomposition of C into C_1, C_2 and C_3.

Figure 2 Decomposing a surface separating cycle into planar cycles.

By Theorem 6, $w_{\text{pl}}(C_i) = 2 \cdot \text{Area}(C_i)$ for each i. Moreover, since the C_i's have all the edges of C' plus some horizontal edges (the dummy edges) of zero weight, therefore $w_{\text{pl}}(C') = \sum_{i=1}^{k} w_{\text{pl}}(C_i) = 2 \cdot \sum_{i=1}^{k} \text{Area}(C_i)$. Therefore $w_{\text{pl}}(C) \neq 0$. ◀

We now show that the number of minimum weight paths with respect to w_{comb}, between any pair of vertices is at most 2^{2g}. We define classes of paths based on the number of times a path intersects each segment of the flat schema, and show that in each such class there is at most one minimum weight path.

Given a polygonal schema of a genus g graph G, by Lemma 2 we assume that we are provided with a flat schema of G having $4g$ segments. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_{2g}\}$ be the set of segments of the flat schema such that no two elements of $\mathcal{T}$ are pairs of each other.

For a path P in G, define the *signature* of P, denoted as $\text{sign}(P)$, as a binary string $s = s_1 s_2 \ldots s_{2g}$ where $s_i = 1$ if P crosses T_i an odd number of times and $s_i = 0$ if P crosses T_i an even number of times. Clearly, the total number of different signatures are 2^{2g}. This definition can be similarly extended to cycles and walks as well.

For $0 \leq i \leq 2^l - 1$, let $\text{bin}(i)$ be the l-bit string that denotes the binary representation of i (if the binary representation has lesser than l bits then we prefix it with appropriate number of zeroes to make it l-bit long). Now consider $l = 2g$. For every pair of vertices u and v, we define 2^{2g} classes of paths $K_0^{uv}, K_1^{uv}, \ldots, K_{2^{2g}-1}^{uv}$ as follows:

$$K_i^{uv} = \{P \mid P \text{ is path from } u \text{ to } v \text{ and } \text{sign}(P) = \text{bin}(i)\}.$$

Note that if $P = P_1 P_2 \ldots P_k$ be a partition of a path P into subpaths, then $\text{sign}(P) = \text{sign}(P_1) \oplus \text{sign}(P_2) \oplus \ldots \text{sign}(P_k)$, where $\oplus$ is the bitwise XOR operator.

For a directed path P from x to y, let P^r represent the directed path from vertex y to x obtained by reversing the edges along the path P. Note that $\text{sign}(P) = \text{sign}(P^r)$.

▶ **Theorem 9.** *Let $G = (V, E)$ be a genus g graph embedded on a flat schema having $4g$ segments. Let u and v be two vertices in G and i be a non-negative integer less than or equal to $2^{2g} - 1$. Then in every class K_i^{uv} there exists at most one minimum weight path from u to v with respect to w_{comb}, that is, the weight function w_{comb} is min-isolating for each set K_i^{uv}.*

Proof. Assume that P_1 and P_2 are two minimum weight paths in K_i^{uv} with respect to w_{comb}. Then $w_{\text{pl}}(P_1) = w_{\text{pl}}(P_2)$ and $w_{\text{len}}(P_1) = w_{\text{len}}(P_2)$. Consider two cases – when P_1 and P_2 have common intermediate vertices and when they do not.

Case 1: P_1 and P_2 do not have any common intermediate vertices. We will show that P_1 and P_2^r together form a surface separating cycle. Let $C = P_1 P_2^r$ be the directed cycle formed by taking P_1 followed by P_2^r. Since P_1 and P_2 do not have any common intermediate vertices therefore C is a simple cycle. Recall that w_{pl} is a skew-symmetric

weight function so $w_{\text{pl}}(P_2^r) = -w_{\text{pl}}(P_2)$. Therefore,

$$\begin{aligned} w_{\text{pl}}(C) &= w_{\text{pl}}(P_1) + w_{\text{pl}}(P_2^r) \\ &= w_{\text{pl}}(P_1) - w_{\text{pl}}(P_2) \\ &= 0 \qquad \text{(since } P_1 \text{ and } P_2 \text{ have the same minimum weight)} \end{aligned}$$

Also since P_1 and P_2 belong to K_i^{uv} we have that $\text{sign}(P_1) = \text{sign}(P_2) = \text{sign}(P_2^r)$. Therefore we get that $\text{sign}(C) = 0$ (the all zeroes vector). By Theorem 3 we have that C is a surface separating cycle and thus by Lemma 8 $w_{\text{pl}}(C)$ cannot be zero. Thus we get a contradiction. Therefore Case 1 cannot occur.

Case 2: P_1 and P_2 have common intermediate vertices. Note that at any common intermediate vertex, the paths P_1 and P_2 can either cross each other or tangentially touch each other without crossing. We refer to the former as *crossing vertex* and the latter as *grazing vertex*.

We will show that the closed walk formed by P_1 and P_2^r reduces to a surface separating simple cycle such that the weight of the closed walk is almost equal to that of the cycle.

▶ **Lemma 10.** *Let P_1 and P_2 be two minimum weight paths from u to v with $u_1, u_2, \ldots u_l$ being the set of common intermediate vertices. Then $u_1, u_2, \ldots u_l$ must occur in the same order in both paths P_1 and P_2.*

Proof. Lemma is trivially true when $l = 1$. So, let $l > 1$. Suppose u_i occurs before u_j in P_1 and u_j occurs before u_i in P_2, for $i < j$. Let a, b and c be the lengths of path P_1 from u to u_i, u_i to u_j and u_j to v respectively. Similarly, let d, e and f be the lengths of path P_2 from u to u_j, u_j to u_i and u_i to v respectively. Since $w_{\text{len}}(P_1) = w_{\text{len}}(P_2)$ we have

$$a + b + c = d + e + f. \tag{1}$$

If $d < a + b$ then taking P_2 from u to u_j and P_1 from u_j to v gives us a shorter length path from u to v than either P_1 or P_2. Similarly, if $d > a + b$ we can construct a shorter length from u to v as well. Hence we can assume that

$$d = a + b. \tag{2}$$

Using analogous argument we can assume

$$f = b + c. \tag{3}$$

Now adding Equations 2 and 3 we have

$$a + 2b + c = d + f. \tag{4}$$

Now since b and e are non zero, Equations 1 and 4 contradict each other. Hence $u_1, u_2, \ldots u_l$ occur in the same order in paths P_1 and P_2. ◀

▶ **Lemma 11.** *Let P_1 and P_2 be two paths in K_i^{uv} having crossing vertices $v_1, v_2, \ldots v_k$, such that these vertices divide P_1 and P_2 into $k+1$ sub-paths $P_1^1, P_1^2, \ldots P_1^{k+1}$ and $P_2^1, P_2^2, \ldots P_2^{k+1}$ respectively (as shown in Figure 3). Then the paths $P' = P_1^1 P_2^2 \ldots P_i^{k+1}$ and $P'' = P_2^1 P_1^2 \ldots P_j^{k+1}$ (where $i = 1$ and $j = 2$ if k is even and $i = 2$ and $j = 1$ if k is odd) belong to the same class.*

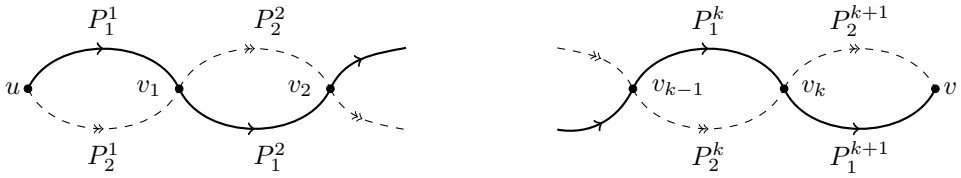


Figure 3 Crossings of paths P_1 (bold line) and P_2 (dashed line) at k many points.

The intuition of Lemma 11 is that if P_1 and P_2 cross each other then the two paths obtained by taking the "above" and "below" portions of these two paths have the same signature.

Proof. Since P_1 and P_2 belong to same class, we have

$$\begin{aligned} \text{sign}(P_1) &= \text{sign}(P_2) \\ \text{sign}(P_1^1) \oplus \text{sign}(P_1^2) \oplus \ldots \oplus \text{sign}(P_1^{k+1}) &= \text{sign}(P_2^1) \oplus \text{sign}(P_2^2) \oplus \ldots \oplus \text{sign}(P_2^{k+1}) \end{aligned}$$

We know that if a, b, c, d are binary strings of equal length then $a \oplus b = c \oplus d \Leftrightarrow a \oplus c = b \oplus d$. Therefore by rearranging the terms we get

$$\begin{aligned} \text{sign}(P_1^1) \oplus \text{sign}(P_2^2) \oplus \ldots \oplus \text{sign}(P_i^{k+1}) &= \text{sign}(P_2^1) \oplus \text{sign}(P_1^2) \oplus \ldots \oplus \text{sign}(P_j^{k+1}) \\ \text{sign}(P') &= \text{sign}(P'') \end{aligned}$$

Hence P' and P'' belong to the same class. ◀

Note that P' and P'' need not belong to the same class as P_1 and P_2. We define $P_i^{j,k} (j < k)$ as a shorthand for path $P_i^j P_i^{j+1} \ldots P_i^k$.

▶ **Lemma 12.** *Let P_1 and P_2 are two minimum weight paths in K_i^{uv} having crossing vertices $v_1, v_2, \ldots v_k$, then $w_{pl}(P_1^i) = w_{pl}(P_2^i)$, for all $i, 1 \leq i \leq k+1$. Additionally, for the closed walk $C' = P'(P'')^r$ we have that $w_{pl}(C') = 0$ and $\text{sign}(C') = 0$ (where P' and P'' are as defined in Lemma 11).*

Proof. Assume that there exists some j $(1 \leq j \leq k+1)$ such that j is the smallest index, where $w_{\text{pl}}(P_1^j) \neq w_{\text{pl}}(P_2^j)$. Without loss of generality assume that $w_{\text{pl}}(P_1^j) < w_{\text{pl}}(P_2^j)$. Now consider path $\widetilde{P} = P_2^{1,j-1} P_1^j P_2^{j+1,k+1}$. $\widetilde{P}$ is a path from u to v and by construction $w_{\text{pl}}(\widetilde{P}) < w_{\text{pl}}(P_2)$. This is a contradiction since P_2 is a minimum weight path from u to v. Therefore for all i we have $w_{\text{pl}}(P_1^i) = w_{\text{pl}}(P_2^i)$.

Now,

$$\begin{aligned} w_{\text{pl}}(C') &= w_{\text{pl}}(P') + w_{\text{pl}}((P'')^r) \\ &= w_{\text{pl}}(P_1^1) + w_{\text{pl}}(P_2^2) + w_{\text{pl}}(P_1^3) + \ldots + w_{\text{pl}}(P_i^{k+1}) + \\ &\quad w_{\text{pl}}((P_2^1)^r) + w_{\text{pl}}((P_1^2)^r) + w_{\text{pl}}((P_2^3)^r) + \ldots + w_{\text{pl}}((P_j^{k+1})^r) \\ &= (w_{\text{pl}}(P_1^1) + w_{\text{pl}}((P_2^1)^r)) + (w_{\text{pl}}(P_2^2) + w_{\text{pl}}((P_1^2)^r)) + (w_{\text{pl}}(P_1^3) + w_{\text{pl}}((P_2^3)^r)) + \\ &\quad \ldots + (w_{\text{pl}}(P_i^{k+1}) + w_{\text{pl}}((P_j^{k+1})^r)) \\ &= (w_{\text{pl}}(P_1^1) - w_{\text{pl}}(P_2^1)) + (w_{\text{pl}}(P_2^2) - w_{\text{pl}}(P_1^2)) + (w_{\text{pl}}(P_1^3) - w_{\text{pl}}(P_2^3)) + \\ &\quad \ldots + (w_{\text{pl}}(P_i^{k+1}) - w_{\text{pl}}(P_j^{k+1})) \\ &= 0. \end{aligned}$$

By Lemma 11 we have that P' and P'' belong to the same class. Hence $\text{sign}(C') = \text{sign}(P') \oplus \text{sign}(P'') = 0$. ◀

We now argue that there is a simple cycle (say $\widehat{C}$) such that C' and $\widehat{C}$ are infinitesimally separated. Hence their signatures are the same. However the weight function w_{pl} depends on the coordinates of an edge, therefore $w_{\text{pl}}(C')$ and $w_{\text{pl}}(\widehat{C})$ are nearly the same. This implies that $w_{pl}(\widehat{C})$ is close to zero. Which leads to a contradiction as we will show that $|w_{pl}(\widetilde{C})| > |w_{pl}(\widehat{C})|$ where $\widetilde{C}$ is one of the planar cycles in which $\widehat{C}$ can be decomposed. Hence P_1 and P_2 cannot be two minimum weight paths in K_i^{uv}.

Consider a graph $\widehat{G}$ that is similar to G except that in $\widehat{G}$ we split each common intermediate vertex u_i (both crossing and grazing vertices) of the paths P_1 and P_2, into two vertices u_i' and u_i'', such that u_i' and u_i'' are δ distance apart (see Figure 4). If $e = xu_i$ was an edge in P_1(or P_2) then we will have the edge $e' = xu_i'$ (or $e' = xu_i''$) in $\widehat{G}$. Let $N = cn^k$ be an upper bound on the coordinates of the embedding of G, where c and k are constants. Then by definition of w_{pl}, we have $|w_{\text{pl}}(e) - w_{\text{pl}}(e')| \leq 4N\delta + \delta^2$. Let us define $f(\delta) := 4N\delta + \delta^2$.

Let $v_1, v_2, \ldots, v_k$ be the crossing vertices of P_1 and P_2, and $Q^{1'}, Q^{2'}, \ldots, Q^{(k+1)'}$ be the paths from u to v_1', v_1' to v_2' and so on till v_k' to v respectively, such that the paths $Q^{1'}, Q^{2'}, \ldots, Q^{(k+1)'}$ correspond to the paths $P_1^1, P_2^2 \ldots, P_i^{k+1}$ respectively (see Figure 4). Note that the paths $Q^{i'}$ and P_j^i (where j is 1 if i is odd and 2 otherwise) can have at most n edges that differ and their weights w_{pl} differ by at most $f(\delta)$ for each such edge. Therefore $|w_{\text{pl}}(P_j^i) - w_{\text{pl}}(Q^{i'})| \leq n \cdot f(\delta)$. Similarly let $Q^{1''}, Q^{2''}, \ldots, Q^{(k+1)''}$ be the paths from u to v_1'', v_1'' to v_2'' and so on till v_k'' to v respectively, such that the paths $Q^{1''}, Q^{2''}, \ldots, Q^{(k+1)''}$ correspond to the paths $P_2^1, P_1^2 \ldots, P_j^{k+1}$ respectively. By an analogous argument we have, $|w_{\text{pl}}(P_j^i) - w_{\text{pl}}(Q^{i''})| \leq n \cdot f(\delta)$ (where j is 2 if i is odd and 1 otherwise).

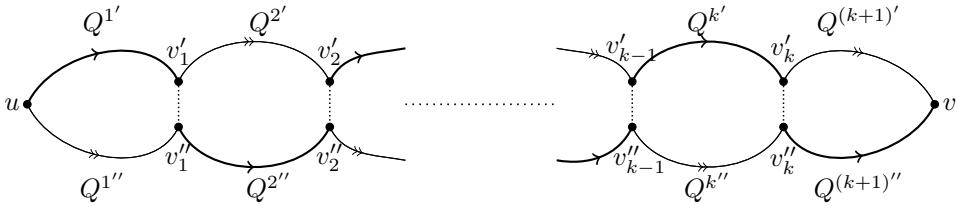


Figure 4 Splitting vertices to form the graph $\widehat{G}$ from G.

Now $Q' = Q^{1'}Q^{2'} \ldots Q^{(k+1)'}$ and $Q'' = Q^{1''}Q^{2''} \ldots Q^{(k+1)''}$ are paths from u to v (corresponding to the paths P' and P'' respectively) that do not cross each other, as shown in Figure 4. Observe that $\text{sign}(Q') = \text{sign}(P')$ since the difference between the coordinates of vertices along Q' and P' is less than 1, therefore the number of crossings with respect to each segment of the flat schema remains the same. Similarly, $\text{sign}(Q'') = \text{sign}(P'')$. Now consider the simple cycle $\widehat{C} = Q'(Q'')^r$. By Lemma 12, $\text{sign}(\widehat{C}) = 0$. Hence $\widehat{C}$ is a surface separating cycle by Theorem 3.

$$
\begin{aligned}
|w_{\text{pl}}(\widehat{C}) - w_{\text{pl}}(C')| &= |(w_{\text{pl}}(Q') + w_{\text{pl}}((Q'')^r)) - (w_{\text{pl}}(P') + w_{\text{pl}}((P'')^r))| \\
&= |\left(\left(w_{\text{pl}}(Q^{1'}) + \ldots + w_{\text{pl}}(Q^{(k+1)'})\right) - \left(w_{\text{pl}}(Q^{1''}) + \ldots + w_{\text{pl}}(Q^{(k+1)''})\right)\right) \\
&\quad - \left(\left(w_{\text{pl}}(P_1^1) + \ldots + w_{\text{pl}}(P_i^{(k+1)})\right) - \left(w_{\text{pl}}(P_2^1) + \ldots + w_{\text{pl}}(P_j^{(k+1)})\right)\right)| \\
&\leq \left|w_{\text{pl}}(Q^{1'}) - w_{\text{pl}}(P_1^1)\right| + \ldots + \left|w_{\text{pl}}(Q^{(k+1)'}) - w_{\text{pl}}(P_i^{(k+1)})\right| \\
&\quad + \left|w_{\text{pl}}(Q^{1''}) - w_{\text{pl}}(P_2^1)\right| + \ldots + \left|w_{\text{pl}}(Q^{(k+1)''}) - w_{\text{pl}}(P_j^{(k+1)})\right| \\
&\leq 2(k+1)nf(\delta)
\end{aligned}
$$

Now since by Lemma 12 $w_{\text{pl}}(C') = 0$, therefore we can choose δ small enough (say less than $1/100N^3$) so that we get $|w_{\text{pl}}(\widehat{C})| < 1/3$.

Without loss of generality assume that C' crosses some segment of the flat schema. If not then both P_1 and P_2 would not be crossing any segment of the polygon and hence with respect to w_{pl} both cannot be minimum weight paths [26]. Since C' crosses some segment, therefore, $\widehat{C}$ also must cross the same segment. Since $\widehat{C}$ is a surface separating cycle therefore by Lemma 8, $\widehat{C}$ can be decomposed into planar cycles such that the weight of $\widehat{C}$ is equal to sum of the weights of the planar cycles with respect to w_{pl}. Moreover, the weight of each planar cycle has the same sign and each planar cycle has a dummy edge (an edge that is incident on a segment of the flat schema). Let $\widetilde{C}$ be one such planar cycle, and consider a triangulation of $\widetilde{C}$ (by thinking of $\widetilde{C}$ as a polygon). There exists some triangle say $T = (a, x, y)$ in this triangulation that contains the dummy edge xy of $\widetilde{C}$ as one of its sides. Now $||x - y|| \geq 1$ since vertices in C' were integral and in $\widehat{C}$, x and y were not shifted. Moreover, a cannot be a vertex that is δ close to any segment of the flat schema. This is because for every vertex v that lies at the intersection of cycle C' and the side S of the flat schema, v was not split when forming the cycle $\widehat{C}$. Hence the distance of a from the line joining x and y is at least $1 - \delta$. Therefore the area of the triangle $\text{Area}(T) > 1/2 - 1/200n$. Now, $\text{Area}(T) \leq \text{Area}(\widetilde{C}) \leq |w_{\text{pl}}(\widetilde{C})| \leq |w_{\text{pl}}(\widehat{C})| \leq 1/3$, where the second inequality follows from Theorem 6. This contradicts that P_1 and P_2 are two minimum weight paths in G with respect to w_{comb}.

Therefore the class K_i^{uv} has at most one minimum weight path from u to v with respect to w_{comb}. This completes the proof of Theorem 9. ◀

For a fixed pair of vertices u, v, the number of classes K_i^{uv} is at most 2^{2g}. Since by Theorem 9 there is at most one minimum weight path from u to v in each class K_i^{uv}, therefore we have the following the result.

▶ **Theorem 13.** *Let $G = (V, E)$ be a genus g graph embedded on a flat schema having $4g$ segments. Then there exists at most 2^{2g} minimum weight paths between any pair of vertices in G, with respect to weight function w_{comb}.*

Now we are ready to prove Theorem 7 which says that there is a weight function with respect to which there is at most one minimum weight path between any pair of vertices in G.

Proof of Theorem 7. Let M_{uv} be the set of all minimum weight paths from u to v with respect to w_{comb} and let $M = \bigcup_{(u,v)\in V^2} M_{uv}$. By Theorem 13, $|M_{uv}|$ is at most 2^{2g}. Hence $|M| \leq n^2 \cdot 2^{2g}$.

Assume that edges of the graph are labeled $e_1, e_2, \ldots, e_m$. Define a weight function w_{index} such that it assigns weight 2^i to the edge e_i. It is easy to see that every path in graph gets unique weight with respect to w_{index}. Thus every path in M also gets a unique weight with respect to w_{index}. However w_{index} is an exponential valued weight function.

Now by Theorem 4 there exists an $O(\log n + g)$ bit prime p such that with respect to weight function $w_{\text{fks}} \coloneqq (w_{\text{index}}) \bmod p$, every path in M gets a unique weight. Therefore with respect to the weight function $w \coloneqq w_{\text{comb}} \cdot n^{k_2} + w_{\text{fks}}$, where k_2 is a sufficiently large constant, the minimum weight path between every pair of vertices in graph is unique. Note that w_{comb} and w_{fks} are $O(\log n)$ bit and $O(\log n + g)$ bit weight functions respectively.

Computing w_{comb} can be done in logspace since it is a simple function of the coordinates of the end points of an edge. To compute w_{fks} one needs to find the appropriate prime whose existence is shown in Theorem 4. For each prime, we check if G is min-unique with respect

to the corresponding weight function and if not we move to the next prime. This can be done by a nondeterministic $O(\log n + g)$ space algorithm along a unique computation path as shown in [20]. ◀

Proof of Theorem 1. Now given a graph G on n vertices and two vertices s and t in G we cycle through all primes less than n', and for each prime, we compute the weight function w given in Theorem 7. Using Theorem 5 we check if G is min-unique with respect to w and if so we check if there is a path from s to t in G. If G is not min-unique with respect to w then we move to the next prime. Theorem 4 guarantees that there is an $n' = n^{O(1)}$ and a prime less than n' such that G is min-unique with respect to the corresponding prime. Hence along a unique computation path, we finally have *Yes* or *No* answer depending on whether s is reachable from t or not respectively, while all other paths halt and reject. ◀

References

1 Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-Sets for ROABP and Sum of Set-Multilinear Circuits. *SIAM J. Comput.*, 44(3):669–697, 2015. doi:10.1137/140975103.

2 Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and Grid Graph Reachability Problems. *Theory of Computing Systems*, 45(4):675–723, 2009. doi:10.1007/s00224-009-9172-z.

3 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, Matching, and Counting: Uniform and Nonuniform Upper Bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

4 Carme Àlvarez and Birgit Jenner. A Very Hard Log-space Counting Class. *Theoretical Computer Science*, 107:3–30, 1993.

5 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing Isolation Lemma for $K_{3,3}$-free and K_5-free Bipartite Graphs. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 10:1–10:15, 2016.

6 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

7 David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Volume 1373 in Lecture Notes in Computer Science, pages 74–83. Springer, 1998.

8 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed Planar Reachability Is in Unambiguous Log-Space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009. doi:10.1145/1490270.1490274.

9 Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory (FCT'91)*, Volume 529 Lecture Notes in Computer Science, pages 168–179. Springer-Verlag, 1991.

10 Sergio Cabello and Bojan Mohar. Finding Shortest Non-Separating and Non-Contractible Cycles for Topologically Embedded Graphs. *Discrete & Computational Geometry*, 37(2):213–235, February 2007. doi:10.1007/s00454-006-1292-5.

11 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically Isolating a Perfect Matching in Bipartite Planar Graphs. In *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, pages 229–240, 2008. doi:10.4230/LIPIcs.STACS.2008.1346.

12 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N.V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *Journal of Computer and System Sciences*, 78(3):765–779, 2012. In Commemoration of Amir Pnueli. doi:10.1016/j.jcss.2011.11.002.

13 Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, June 1997.

14 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite Perfect Matching is in quasi-NC. *CoRR*, 2016. arXiv:1601.06319.

15 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a Sparse Table with 0(1) Worst Case Access Time. *J. ACM*, 31(3):538–544, June 1984. doi:10.1145/828.1884.

16 Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996. doi:10.1002/(SICI)1098-2418(199608/09)9:1/2\%3C99::AID-RSA7\%3E3.0.CO;2-6.

17 Anna Galluccio and Martin Loebl. On the Theory of Pfaffian Orientations. I. Perfect Matchings and Permanents. *Electr. J. Comb.*, 6, 1999. URL: http://www.combinatorics.org/Volume_6/Abstracts/v6i1r6.html.

18 Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity Testing for Constant-Width, and Commutative, Read-Once Oblivious ABPs. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 29:1–29:16, 2016. doi:10.4230/LIPIcs.CCC.2016.29.

19 Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic Identity Testing for Sum of Read-once Oblivious Arithmetic Branching Programs. In *Proceedings of the 30th Conference on Computational Complexity*, CCC '15, pages 323–346, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CCC.2015.323.

20 Vivek Anand T. Kallampally and Raghunath Tewari. Trading Determinism for Time in Space Bounded Computations. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 10:1–10:13, 2016. doi:10.4230/LIPIcs.MFCS.2016.10.

21 Jan Kynčl and Tomáš Vyskočil. Logspace Reduction of Directed Reachability for Bounded Genus Graphs to the Planar Case. *ACM Transactions on Computation Theory*, 1(3):1–11, 2010. doi:10.1145/1714450.1714451.

22 Meena Mahajan and Kasturi R. Varadarajan. A New NC-algorithm for Finding a Perfect Matching in Bipartite Planar and Small Genus Graphs (Extended Abstract). In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 351–357, New York, NY, USA, 2000. ACM. doi:10.1145/335305.335346.

23 Aduri Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in log-space. *Computational Complexity*, 21(4):643–670, 2012. doi:10.1007/s00037-012-0047-3.

24 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.

25 Klaus Reinhardt and Eric Allender. Making Nondeterminism Unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000. doi:10.1137/S0097539798339041.

26 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012. doi:10.1016/j.ic.2012.03.002.

27 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free Graphs and K_5-free Graphs is in Unambiguous Log-Space. In *17th International Conference on Foundations of Computation Theory (FCT)*, Lecture Notes in Computer Science 5699, pages 323–334. Springer-Verlag, 2009.

28 Carsten Thomassen. The Graph Genus Problem is NP-Complete. *J. Algorithms*, 10(4):568–576, 1989. doi:10.1016/0196-6774(89)90006-0.

29 Dieter van Melkebeek and Gautam Prakriya. Derandomizing Isolation in Space-Bounded Settings. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 5:1–5:32, 2017. doi:10.4230/LIPIcs.CCC.2017.5.

30 Avi Wigderson. $NL/poly \subseteq \oplus L/poly$ (Preliminary Version). In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference, Amsterdam, The Netherlands, June 28 - July 1, 1994*, pages 59–62, 1994. doi:10.1109/SCT.1994.315817.

Dominating Sets and Connected Dominating Sets in Dynamic Graphs

Niklas Hjuler
University of Copenhagen, Denmark
hjuler@di.ku.dk

Giuseppe F. Italiano
LUISS University, Rome, Italy
gitaliano@luiss.it

Nikos Parotsidis
University of Rome Tor Vergata, Italy
nikos.parotsidis@uniroma2.it

David Saulpic
ENS Paris, France
david.saulpic@ens.fr

Abstract

In this paper we study the dynamic versions of two basic graph problems: Minimum Dominating Set and its variant Minimum Connected Dominating Set. For those two problems, we present algorithms that maintain a solution under edge insertions and edge deletions in time $O(\Delta \cdot \text{polylog } n)$ per update, where Δ is the maximum vertex degree in the graph. In both cases, we achieve an approximation ratio of $O(\log n)$, which is optimal up to a constant factor (under the assumption that $P \neq NP$). Although those two problems have been widely studied in the static and in the distributed settings, to the best of our knowledge we are the first to present efficient algorithms in the dynamic setting.

As a further application of our approach, we also present an algorithm that maintains a Minimal Dominating Set in $O(min(\Delta, \sqrt{m}))$ per update.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dominating Set, Connected Dominating Set, Dynamic Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.35

Funding This work was done while Niklas Hjuler and David Saulpic were visiting University of Rome Tor Vergata.

1 Introduction

The study of dynamic graph algorithms is a classical area in algorithmic research and has been thoroughly investigated in the past decades. Maintaining a solution of a graph problem in the case where the underlying graph changes dynamically over time is a big challenge in the design of efficient and practical algorithms. Indeed, in several applications, due to the dynamic nature of today's data, it is not sufficient to compute a solution to a graph problem only once and for all: often, it is necessary to *maintain* a solution efficiently while the input graph is undergoing a sequence of dynamic updates. More precisely, a *dynamic graph* is a sequence of graphs $G_0, ..., G_M$ on n nodes and such that G_{i+1} is obtained from G_i by adding or removing a single edge. The natural first barrier, in the study of dynamic algorithms, is to design algorithms that are able to maintain a solution for the problem at hand after each update faster than recomputing the solution from scratch. Many dynamic graph problems such as minimum spanning forests (see e.g. [22, 26]), shortest paths [12], matching [4, 27, 30] or coloring [7] have been extensively studied in the literature, and very efficient algorithms are known for those problems. Recently, a lot of attention has been devoted to the Maximal

Independent Set problem (MIS). In this problem, one wishes to find a maximal set of vertices that do not share any edge ("maximal" meaning that it is not possible to add any vertex without violating this property). Until very recently, the best known update bound on the complexity to maintain a MIS was a simple $O(\Delta)$ algorithm, where Δ is an upper bound on the degree of vertices in the graph. This bound was first broken by Assadi et al. [2] who gave a $O(m^{3/4})$ algorithm, then by Gupta and Khan [19] improved the update bound to $O(m^{2/3})$. Very recently, using randomization, Assadi et al. [3] presented an amortized fully-dynamic algorithm with an expected $\widetilde{O}(n^{1/2})$-time bound per update.

The MIS problem is closely related to the Dominating Set (DS) problem: given a graph $G = (V, E)$ the DS problems is to find a subset of vertices $\mathcal{D} \subseteq V$ such that every vertex in G is adjacent to $\mathcal{D}$ (or *dominated* by $\mathcal{D}$). Indeed, a MIS is also a Minimal DS: the fact that it is not possible to add a vertex without breaking the independence property implies that every vertex is adjacent to the MIS, so this must be also a DS; at the same time, it is not possible to remove a vertex since that vertex is no longer dominated. Thus, to find a Minimal DS one can simply find a MIS: this gives immediately a deterministic $O(m^{2/3})$ [19] bound and a randomized $\widetilde{O}(n^{1/2})$ [3] one. However, while it is known that is hard to approximate Maximum Independent Set[1] within a factor $n^{1-\epsilon}$ for every $\epsilon > 0$[21], a simple greedy approach achieves a $O(\log n)$-approximation for Minimum DS [11].

In recent years, there has been a lot of work on designing dynamic graph algorithms for maintaining approximate solutions to several problems. A notable example is matching, where for different approximations there exist different algorithms (see e.g., [4, 5, 27, 20, 8, 30]). This raises the natural question on whether there exists a dynamic algorithm capable of maintaining an approximation to Minimum DS, and even better a $O(\log n)$ approximation. In this paper, we answer this question affirmatively by presenting an algorithm that achieves a $O(\log n)$ approximation, with a complexity matching the long standing $O(\Delta)$ bound for MIS. Moreover, if one is interested in finding a DS faster, we present a very simple *deterministic* $O(m^{1/2})$ algorithm to compute a Minimal DS, improving the $O(m^{2/3})$ bound coming from MIS. We believe these are important steps towards understanding the complexity of the problem. Those two results are stated below.

▶ **Theorem 1.** *Starting from a graph with n vertices, a $O(\log n)$ approximation of* Minimum Dominating Set *can be maintained over any sequence of $\Omega(n)$ edge insertions and deletions in $O(\Delta \log n)$ amortized time per update, where Δ is the maximum degree of the graph over the sequence of updates.*

▶ **Theorem 2.** *Starting from a graph with n (fixed) vertices, a* Minimal Dominating Set *can be deterministically maintained over any sequence of edge insertions and deletions in $O(\sqrt{m})$ amortized time per update, where m is an upper bound on the number of edges in the graph.*

We also study the Minimum Connected Dominating Set problem (MCDS), which adds the constraint that the graph induced by the DS $\mathcal{D}$ must be connected. This problem was first introduced by Sampathkumar and Walikar [28] and arises in several applications. The most noteworthy is its use as a *backbone* in routing protocols: it allows to limit the number of packet transmissions, by sending packets only along the backbone rather than throughout the whole network. Du and Wan's book [13] summarizes the knowledge about

[1] It is not possible to find a polynomial-time algorithm that finds a $n^{1-\epsilon}$-approximation to Maximum Independent Set under the assumption NP $\neq$ ZPP

MCDS. A special class of graphs is geometric graphs, where vertices are points in the plane, and two vertices are adjacent if they fall within a certain range (say, their distance is at most 1). This can model wifi transmissions, and the dynamic MCDS had been studied in this setting: a polynomial-time approximation scheme is known [10], and Guibas et al. [17] show how to maintain a constant-factor approximation with polylogarithmic update time. While geometric graphs model problems linked to wifi transmissions, the general graph setting can be also seen as a model for wired networks. However, no work about dynamic MCDS is known in this setting: the static case is well studied, with a greedy algorithm developed by Guha and Keller [16] that achieves an approximation factor $O(\ln \Delta)$. They also show a lower bound matching their complexity, together with their approximation factor. MCDS had also been thoroughly studied in the distributed setting (see e.g. a heuristic to find a Minimal CDS in [9], another one that sends $O(\Delta n)$ messages and has a time complexity at each vertex $O(\Delta^2)$ [31] or a $3 \log n$ approximation that runs in $O(\gamma)$ rounds where γ is the size of the CDS found, with time complexity $O(\gamma\Delta^2 + n)$ and message complexity $O(n\Delta\gamma + m + n \log n)$ [6]). Despite all this work, no results are known in the dynamic graph setting. As another application of our approach, we contribute to filling this gap in the research line of MCDS. In particular, in this paper we show how our algorithm for Minimum DS can be adapted in a non-trivial way to maintain a $O(\log n)$ approximation of the MCDS in general dynamic graphs.

▶ **Theorem 3.** *Starting from a graph with n vertices, a $O(\log n)$ approximation of* Minimum Connected Dominating Set *can be maintained over any sequence of $\Omega(n)$ edge insertions and deletions in $\widetilde{O}(\Delta)$ amortized time per update.*

We further show how to maintain independently a Dominating Set $\mathcal{D}$ and a set of vertices $\mathcal{C}$ such that the induced subgraphs on the vertices $\mathcal{C} \cup \mathcal{D}$ is connected. The set $\mathcal{C}$ has the additional property that $|\mathcal{C}| \leq 2|\mathcal{D}|$, such that $|\mathcal{C} \cup \mathcal{D}| = O(|\mathcal{D}|)$. If $\mathcal{D}$ is a α-approximation of Minimum DS, this gives a $O(\alpha)$ approximation for MCDS.

Further Related Work

It is well known that finding a Minimum DS is NP-hard [15]. It is therefore natural to look for approximation algorithms for this problem. Unfortunately, it is also NP-hard to find a $c \log n$ approximation, for any $0 < c < 1$ [14]. This bound is tight, since there is a simple greedy algorithm matching this bound [11]. Minimum DS had been studied extensively in distributed computing: an algorithm which runs in $O(\log n \log \Delta)$ rounds finds a $O(\log n)$ approximation with high probability [23] and an algorithm with constant number of rounds achieves a non-trivial approximation[25].

The DS problem is closely related to the Set Cover problem: the two problems are equivalents under L-reduction [24]. However, Set Cover was studied in the dynamic setting [18, 1], but with different kinds of updates: instead of edges being inserted or deleted (which would represent new elements in the sets according to the L-reduction), new elements are being added to the cover (which would be new vertices in DS).

Outline. The rest of the paper is organized as follows. First, we present an algorithm for Minimum DS, which will be used later on also for MCDS: we start by a $\widetilde{O}(n)$ algorithm, and then show how to overcome its bottleneck in order to achieve a $\widetilde{O}(\Delta)$ complexity. Finally, we present our $O(\sqrt{m})$ algorithm for Minimal DS.

2 A $O(\log n)$ approximation of Minimum Dominating Set in $O(\Delta \log n)$ time per update

This section aims at proving Theorem 1. Following a reduction from Set Cover, minimum DS is NP-hard to approximate within a factor $\log n$ [14]. Here we present a matching upper bound (up to a constant factor), in the dynamic setting. Our algorithm relies heavily on the clever set cover algorithm by Gupta et al. [18]. While in the static setting Set Cover is equivalent to minimum DS, in the dynamic setting these two problems are different. More precisely, in the dynamic Set Cover problem one is asked to cover a set of points S (called the universe) with a given family of sets F, while the set S is changing dynamically. To draw the parallel with DS, in the latter the set S is the set of vertices of the graph (which does not change) and for every vertex the set of its neighbors is in F. The dynamic part concerns therefore F, and not the universe S.

Gupta et al. present an $O(\log n)$-approximation for dynamic Set Cover problem: in what follows, we show how to adapt their algorithm to the DS case, with an update time of $O(\Delta \log n)$. As in [18], the approach easily adapts to the weighted case. Unfortunately, this cannot be generalized to MCDS, therefore we do not consider this property of the algorithm. The following definitions are partly adapted from [18].

2.1 Preliminaries

For a vertex v, let $N(v)$ be the set of its neighbors, including v. The algorithm maintains a solution S_t at time t such that an element of S_t is a pair composed of

- a dominant vertex v
- a set $\mathrm{Dom}(v) \subseteq N(v)$, which are the vertices that are dominated by v. We call $|\mathrm{Dom}(v)|$ the *cardinality* of the pair.

We call a *dominating pair* an element of S_t. The algorithm requires that multiple copies of a vertex can appear as the dominant vertex of a pair. However, each vertex is exactly in one $\mathrm{Dom}(v)$. The solution to the DS problem is composed of all vertices that appear as dominant vertices of a pair. Since each vertex is in exactly one $\mathrm{Dom}(v)$, each vertex is dominated and therefore the set of dominants is a valid solution to the DS problem.

The dominating pairs are placed into *levels* according to their cardinality: the level l is defined by a range $R_l := [2^{l-10}, 2^l]$, and each pair $(v, \mathrm{Dom}(v))$ is placed at an appropriate level l such that $|\mathrm{Dom}(v)| \in R_l$. In that case, elements of $\mathrm{Dom}(v)$ are said to be dominated at level l; we denote by V_l the set of all vertices dominated at level l. We say that an assignment of levels is valid if it respects the constraint $|\mathrm{Dom}(v)| \in R_l$. This allows us to define the notion of *Stability*:

- *stable solution*: A solution S_t is stable if there is no vertex v and level l such that $|N(v) \cap V_l| > 2^l$; in other words, it is not possible to introduce a new vertex in the solution to dominate some vertices at level l such that the resulting dominating pair could be at level strictly greater that l.

The algorithm will dynamically maintain a stable solution S_t, with a valid assignment of levels. Note that the ranges R_l overlap: this gives some slack to the algorithm, which allows enough flexibility to prevent too many changes while our algorithm maintains a valid solution.

2.2 The algorithm

The main part of the algorithm is the function STABILIZE, which restores the stability at the end of every update. The function is the following (see [18]):

> STABILIZE. As long as a vertex v violates the stability condition at level l, do the following: Add the pair $(v, N(v) \cap V_l)$ to the *lowest* possible level j (i.e., the lowest level such that $|N(v) \cap V_l| \in R_j$); Remove the elements of $N(v) \cap V_l$ from the set of their former covering pair: if it gets empty, remove the pair from the solution. Otherwise, if the cardinality of such a pair goes below 2^{l-10}, put it at the *highest* possible level.

Edge addition. When a new edge (u, v) is added to the graph, one just need to ensure that the solution remains stable, and thus the algorithm runs STABILIZE.

Edge deletion. When an edge (u, v) is removed from the graph, we proceed as follows. If neither u nor v dominates the other endpoint, the solution remains valid and stable, and nothing needs to be done. Otherwise, assume without loss of generality that v dominates u. Then:

- Remove u from $\mathrm{Dom}(v)$
- Add the pair $(u, \mathrm{Dom}_u = \{u\})$ to the solution with level 1
- Run STABILIZE

Correctness. All the nodes of the graph are dominated at every time. Indeed, STABILIZE does not make any node undominated and if a vertex is not dominated after an edge removal, the algorithm simply adds it to the solution. Therefore, the solution S_t maintained by the algorithm is a valid one.

2.3 Analysis

Approximation ratio. We use the following lemma by Gupta et al. [18] to bound the cost of a stable solution.

▶ **Lemma 4** (Lemma 2.1 in [18]). *The number of sets at one level in any stable solution is at most* $2^{10} \cdot OPT$.

Since for every dominating pair $(v, \mathrm{Dom}(v))$ we have that $1 \leq |\mathrm{Dom}(v)| \leq n$, there are only $\log n$ levels that can contain a set. The total cost of a stable solution is therefore $O(\log n \cdot \mathrm{OPT})$.

A token scheme to bound the number of updates. Unfortunately, the analysis of Gupta et al. cannot be applied directly to the case of DS, due to the different nature of the updates. However, we can build upon their analysis, as follows. We first bound the number of vertices that change level, and then explain how to implement a level change so that it costs $O(\Delta)$. We prove the following lemma by using a token argument.

▶ **Lemma 5.** *After k updates of the algorithm, at most $O(k \log n + n \log n)$ elements have changed levels.*

Proof. We use the following token scheme, where each vertex pays one token for each level change. In the beginning, we give $2\log n$ tokens to every vertex. If a vertex is undominated after an edge removal, we give $2\log n$ new tokens to this vertex. Since at most one vertex gets undominated for each edge deletion, the total number of tokens given after k updates is $O(k\log n + n\log n)$. To prove the lemma, we need to show that at any time each vertex has always a positive amount of tokens. We adapt the proof of Gupta et al. to show the following invariant:

▶ **Invariant 1.** *Every vertex at level l has more than $2(\log n - l)$ tokens.*

When a vertex is moved to a higher level, it pays one token for the cost of moving. It also saves one token, and gives it to an "emergency fund" of its former covering pair. Each pair has therefore a fund of tokens that can be used when the pair has to be moved to a lower level.

When the pair $(v, \mathrm{Dom}(v))$ has to be moved from level l to level $l-j$, it means that a lot of vertices have left $\mathrm{Dom}(v)$ and that the tokens they gave to the pair can be used to pay for the operation. Formally, we want to pay one token for every vertex in $\mathrm{Dom}(v)$ for its level change, but we also want to restore the invariant. We need therefore $2j+1$ tokens for each vertex of $\mathrm{Dom}(v)$. Since the pair can be moved to level $l-j$, this means that $|\mathrm{Dom}(v)| < 2^{l-j}$. Since a new pair is moved to the lowest possible level, this pair could not be at level $l-1$, which implies that $|\mathrm{Dom}_{init}(v)| > 2^{l-1}$ where $\mathrm{Dom}_{init}(v)$ is the set $\mathrm{Dom}(v)$ at the time where it was created. Moreover, each of the vertices that left gave one token: the amount of tokens usable is therefore bigger than $2^{l-1} - 2^{l-j}$. Thus we want to prove that $2^{l-1} - 2^{l-j} \geq (2j+1)\cdot|\mathrm{Dom}(v)|$. It is enough to have $2^{l-1} - 2^{l-j} \geq 3\cdot(2j+1)2^{l-j}$, i.e. to have $2^{j-1} - 1 \geq 3(2j+1)$. But since the pair was moved to level $l-j$, it means that $|\mathrm{Dom}(v)| > 2^{l-j-1}$ and $|\mathrm{Dom}(v)| < 2^{l-10}$: putting these two equations together gives $j > 9$, which ensures that $2^{j-1} - 1 \geq 3(2j+1)$ and concludes the proof. ◀

As the following corollary shows, we can bound the number of changes to $\mathcal{D}$ to $O(\log n)$ amortized. This property will be useful in Section 3.

▶ **Corollary 6.** *After k updates of the algorithm, at most $O(k\log n + n\log n)$ vertices can be added to or removed from $\mathcal{D}$.*

Proof. Whenever a vertex is added to or removed from $\mathcal{D}$, its level is changed. Lemma 5 gives the corresponding bound. ◀

We now turn to the implementation of the function Stabilize. As shown in the next lemma, we implemented so that its cost is $O(\Delta)$ for each element that changes level.

▶ **Lemma 7.** *A stable solution can be maintained in $O(\Delta\log n)$ amortized time per update.*

Proof. For all vertices v and all levels l, the algorithm maintains the set $N(v)\cap V_l$ and its cardinality. Every time a vertex changes its level, it has to inform all its neighbors: this can be done in $O(\Delta)$. When an edge (u,v) is added to or removed from the solution, the algorithm updates the sets $N(v)\cap V_{l_u}$ and $N(u)\cap V_{l_v}$, where l_u and l_v are the levels of u and v, respectively.

During a call to Stabilize, the algorithm maintains also a list of vertices that may have to be added to restore the stability: for a vertex v and level l, every time that $N(v)\cap V_l$ changes, if the new cardinality violates the stability, we add v to this list in constant time. The algorithm processes the list vertex by vertex: it checks that the current vertex still needs to be added to the solution, and add it if necessary.

Since we pay $O(\Delta)$ per level change and there are $O(\log n)$ amortized changes, the amortized complexity of each update is $O(\Delta\log n)$. ◀

Since a stable solution gives a $O(\log n)$ approximation to minimum DS, Lemmas 4 and 7 yield the proof of Theorem 1: a $O(\log n)$ approximation of Minimum Dominating Set can be maintained in $O(\Delta \log n)$ amortized time per update.

3 A $O(\log n)$ Approximation for Minimum Connected Dominating Set in $\widetilde{O}(n)$ per update

A possible way to compute a Connected DS is simply to find a DS and add a set of vertices to make it connected. Section 2 gives an algorithm to maintain an approximation of the Minimum DS: we will use it as a black box (and refer to it as the "black box"), and show how to make its solution connected without losing the approximation guarantee. If the original graph is not connected, the algorithm finds a CDS in every connected component: we focus in the following on a single of these components. Let $\mathcal{D}$ be the DS maintained, and $\mathcal{C}$ be a set of vertices such that $\mathcal{C} \cup \mathcal{D}$ is connected and $\mathcal{C}$ is minimal for that property. The minimality of $\mathcal{C}$ will ensure that $|\mathcal{C}| \leq 2|\mathcal{D}|$: since $\mathcal{D}$ is a $O(\log n)$ approximation of MDS, this leads to a $O(\log n)$ approximation for MCDS. Note that the vertices of $\mathcal{C}$ are not used for domination: $\mathcal{C} \cup \mathcal{D}$ is therefore not minimal, but still an approximation of minimum.

Overall, we will apply the following charging scheme to amortize the total running time. The main observation is that although a lot of vertices can be deleted to restore the minimality of $\mathcal{C}$, only a few can be added at every step. We thus give enough potential to a vertex whenever it is added into $\mathcal{C}$ and whenever its neighborhood changes, so that at the time of its removal from $\mathcal{C}$ it has accumulated enough potential for scanning its entire neighborhood. After an edge deletion we might have to restore the connectivity requirement. We do that by adding at most 2 new vertices in $\mathcal{C}$: this is crucial for our amortization argument.

Outline. The set $\mathcal{C}$ may have to be updated for two reasons:

- Restore the connectivity: if an edge gets deleted from the graph, or if the black box removes some vertices from $\mathcal{D}$, it may be necessary to add some vertices to $\mathcal{C}$ in order to restore the connectivity of $\mathcal{C} \cup \mathcal{D}$.
- Restore the minimality of $\mathcal{C}$: when an edge is added to the graph, or when a vertex is added to $\mathcal{C} \cup \mathcal{D}$ (either by the black box or in order to restore the connectivity), some vertices of $\mathcal{C}$ may become useless and therefore need to be removed.

We now address those two points. All our bounds are expressed in term of the total number of changes in $\mathcal{C} \cup \mathcal{D}$: let therefore k be this number of changes. We will show later that, after t updates to the graph, $k = O(t \log n)$.

The first phase of the algorithm is to restore the connectivity. We explain in the following how to decide which vertices should be added to $\mathcal{C}$ for that purpose.

Restore the connectivity after an edge deletion

To monitor the connectivity requirement, we use the following idea. The algorithm maintains a minimum spanning tree (MST) of the graph G where a weight 1 is assigned to the edges between vertices in $\mathcal{C} \cup \mathcal{D}$ (called from now on $\tilde{\mathcal{D}}$), and weight m is assigned to all other edges. These weights ensure that, as long as $\tilde{\mathcal{D}}$ is connected, the MST induces a tree on $\tilde{\mathcal{D}}$. When $G[\tilde{\mathcal{D}}]$ gets disconnected by an update, the MST uses a vertex of $V \setminus \tilde{\mathcal{D}}$ as an internal vertex: in that case, our algorithm adds this vertex to $\mathcal{C}$, to restore the connectivity. We give more details in the next section.

The edge weights are updated as the graph undergoes edge insertions and deletions and vertices enter or leave $\tilde{\mathcal{D}}$. The MST of the weighted version of the graph has the following properties.

- If $\tilde{\mathcal{D}}$ is a connected DS, then the MST has weight $(|\tilde{\mathcal{D}}| - 1) + m \cdot |V \setminus \tilde{\mathcal{D}}|$ (Kruskal's algorithm on this graph would use $|\tilde{\mathcal{D}}| - 1$ edges of weight 1 to construct a spanning tree on $\tilde{\mathcal{D}}$, then $|V \setminus \tilde{\mathcal{D}}|$ edges of weight m to span the entire graph).
- If $\tilde{\mathcal{D}}$ is a DS but $G[\tilde{\mathcal{D}}]$ is not connected, then the weight of the MST has larger value.

The two properties stem from the fact that a MST can be produced by finding a minimum spanning forest on $\tilde{\mathcal{D}}$ and extend it to a MST on V. Kruskal's algorithm ensures that this leads to a MST. In the case where $\tilde{\mathcal{D}}$ is connected, the first step yields a tree of weight $\tilde{\mathcal{D}} - 1$, and since the graph is connected the second step yields a cost $m \cdot |V \setminus \tilde{\mathcal{D}}|$. However, if $\tilde{\mathcal{D}}$ is not connected, the second step adds strictly more that $|V \setminus \tilde{\mathcal{D}}|$ edges, therefore yielding a cost bigger than $m \cdot (1 + |V \setminus \tilde{\mathcal{D}}|)$. This is more than $(|\tilde{\mathcal{D}}| - 1) + m \cdot |V \setminus \tilde{\mathcal{D}}|$, as claimed.

Furthermore, if $G[\tilde{\mathcal{D}}]$ has two connected components C_1, C_2, then the shortest of all paths between vertices $u, v,\ u \in C_1, v \in C_2$ is the minimum number of vertices whose insertion into $\mathcal{C}$ restores the connectivity requirement. Note that the shortest of all such paths must have length at most 2 (otherwise, there must be a vertex not adjacent to any vertex in $\mathcal{D}$, which contradicts the fact that $\mathcal{D}$ is a DS).

After an edge deletion, it may happen that $\tilde{\mathcal{D}}$ becomes disconnected and that the MST includes some internal vertices (at most 2, by the previous discussion) not in $\tilde{\mathcal{D}}$: in that case, we add them to $\mathcal{C}$. This turns out to be enough to ensure the connectivity.

To maintain the MST of the weighted version of the input graph we use the $O(\log^4 n)$ update time fully-dynamic MST algorithm from [22]. Since the weights of the edges incident to the vertices that enter or leave $\tilde{\mathcal{D}}$ are also updated, the algorithm runs in time $\widetilde{O}(\Delta)$ for each change in $\tilde{\mathcal{D}}$, i.e. in time $k \cdot \widetilde{O}(\Delta)$

Restore the connectivity when a vertex is deleted by the black box. When a vertex v is deleted from $\mathcal{D}$ by the black box DS algorithm, we need to be more careful: updating the edge weights and finding the new MST may add a lot of vertices to $\mathcal{C}$ (as many as Δ, one per edge of the MST incident to v). However, if the removal of v disconnects $G[\tilde{\mathcal{D}}]$, it is enough to add v to $\mathcal{C}$ to restore the connectivity. If its removal does not disconnect $G[\tilde{\mathcal{D}}]$, nothing needs to be done. It is possible to know if the graph $G[\tilde{\mathcal{D}}]$ gets disconnected using the properties of the MST, by only looking at the weight of the MST. The complexity of this step is therefore $\widetilde{O}(\Delta)$, the time needed to update the weights of the MST.

Restore the minimality. The second phase of the algorithm is to restore the minimality of $\mathcal{C}$. We explain next how to find the vertices of $\mathcal{C}$ that need to be removed to accomplish this task. This minimality condition is equivalent to the condition that all vertices in $\mathcal{C}$ are *articulation points* in the graph induced by $\mathcal{C} \cup D$. (An articulation point is a vertex such that its removal increases the number of connected components.) This turns out to be useful in order to identify which vertices need to be removed to restore the minimality of $\mathcal{C}$.

To restore the connectivity requirement, new vertices were added into $\mathcal{C}$, and the black box added some vertices to $\mathcal{D}$: this might result in some vertices in $\mathcal{C}$ not being articulation points of $G[\tilde{\mathcal{D}}]$ anymore. As observed before, these are the vertices that need to be removed. We need to identify a maximal set of such vertices that can be removed from $\mathcal{C}$ without violating the connectivity requirement. To do this, the algorithm queries in an arbitrary order one-by-one all the vertices $v \in \mathcal{C}$ to determine whether $G[\tilde{\mathcal{D}} \setminus v]$ is connected. This can be done using a data structure from Holm et al. [22] that requires $\widetilde{O}(1)$ per query. Whenever

the algorithm identifies a vertex such that $G[\tilde{\mathcal{D}} \setminus v]$ is connected, it can safely remove it from $\mathcal{C}$. The complexity of this step is therefore $\widetilde{O}(n)$ to find all articulation points, and an extra $\widetilde{O}(\Delta)$ for each of the vertices we remove from $\mathcal{C}$.

The following three lemmas conclude the proof: the first shows that the algorithm is correct, the second the $\widetilde{O}(n)$ time bound and the third the $O(\log n)$ approximation ratio.

▶ **Lemma 8.** *The algorithm that first restores the connectivity of $\mathcal{C} \cup \mathcal{D}$ and then the minimality of $\mathcal{C}$ is correct: it gives a minimal set $\mathcal{C}$ such that $\mathcal{C} \cup \mathcal{D}$ is connected.*

Proof. After restoring the connectivity requirement the algorithm maintains a spanning tree of $\tilde{\mathcal{D}}$, so $G[\tilde{\mathcal{D}}]$ is indeed connected. In the following steps, before the algorithm removes a vertex v from $\mathcal{C}$, it first verifies that $G[\tilde{\mathcal{D}} \setminus v]$ remains connected, which guarantees that $G[\tilde{\mathcal{D}}]$ is connected at the end of the update procedure. Since the black box ensures that $\mathcal{D}$ is a DS, $\tilde{\mathcal{D}}$ is a DS too: hence at the end, $\tilde{\mathcal{D}}$ satisfies both the domination and the connectivity requirements. It remains to show that $\mathcal{C}$ is minimal, i.e., that all vertices in $\mathcal{C}$ are articulation points in $G[\tilde{\mathcal{D}}]$. Since during the second step the algorithm only removes vertices from $\mathcal{C}$, a vertex that was not an articulation point cannot become one, and therefore the loop to find the articulation points is correct. The set $\mathcal{C}$ is therefore a minimal set such that $\mathcal{C} \cup \mathcal{D}$ is connected. ◀

▶ **Lemma 9.** *The amortized complexity of the algorithm is $\widetilde{O}(n)$ per update.*

Proof. The amortized cost of the black box to compute $\mathcal{D}$ is $\widetilde{O}(\Delta)$. We analyze now the additional cost of maintaining $\tilde{\mathcal{D}}$. As shown in this section, the cost to add or delete a vertex from $\tilde{\mathcal{D}}$ is $\widetilde{O}(\Delta)$. To prove the lemma, we bound the number of changes in $\tilde{\mathcal{D}}$. For that, we count the number of vertices *added* to $\tilde{\mathcal{D}}$: in an amortized sense this bounds the number of changes too. Formally, we pay a budget $deg(v)$ when v is added to $\tilde{\mathcal{D}}$. Following insertions and deletions of edges adjacent to v, we update this budget (with a constant cost), so that when v gets deleted from $\tilde{\mathcal{D}}$ a budget equal to its degree is available to spend.

From Corollary 6, the black box makes at most $\widetilde{O}(1)$ changes to $\mathcal{D}$ per update (in an amortized sense). If it removes a vertex from $\mathcal{D}$, we showed previously that no new vertex is added to $\tilde{\mathcal{D}}$. The number of additions to $\tilde{\mathcal{D}}$ is therefore $\widetilde{O}(1)$. Moreover, in the case of an edge deletion, at most two vertices are added to $\tilde{\mathcal{D}}$ to maintain the connectivity. Since restoring the minimality requires only to delete vertices, the total number of additions into $\tilde{\mathcal{D}}$ is $\widetilde{O}(1)$. As the cost for any of these additions is $\widetilde{O}(\Delta)$, the total cost of this algorithm is upper bounded by the loop to find the articulation points, which is $\widetilde{O}(n)$. ◀

▶ **Lemma 10.** *The algorithm maintains a $O(\log n)$ approximation for MCDS, i.e. $|\mathcal{C} \cup \mathcal{D}| = O(\log n) \cdot OPT$*

Proof. We first prove that $|\mathcal{C}| \leq 2|\mathcal{D}|$, using the minimality of $\mathcal{C}$. Each vertex of $\mathcal{C}$ is there to connect some components of $\mathcal{D}$. Consider the graph (W, F) where vertices W are either connected components of $\mathcal{D}$ or vertices of $\mathcal{C}$, and the set F of edges is constructed as follows. Start with a graph containing one vertex for each connected component of $\mathcal{D}$, and add vertices of $\mathcal{C}$ one by one. When the vertex v is added, identify a node u in $\mathcal{D}$ adjacent to v such that adding the edge (u, v) to F does not create a cycle: add to F an edge between v and the node corresponding to the connected component containing u. It is always possible to find such a vertex u, otherwise v would not be necessary for the connectivity, which would contradict the minimality of $\mathcal{C}$. This process gives a forest such that every node of $\mathcal{C}$ is adjacent to a connected component of $\mathcal{D}$. Since $\mathcal{C} \cup \mathcal{D}$ is connected, it is possible to complete F to make it a tree, adding some other edges. This tree has the two following properties.

1. The leaves are vertices that correspond to connected component of $\mathcal{D}$: indeed, if a vertex of $\mathcal{C}$ was a leaf in this tree, it could be removed without losing the connecting of $\mathcal{C} \cup \mathcal{D}$, which would contradict the minimality of $\mathcal{C}$.
2. Any vertex of $\mathcal{C}$ is adjacent to a connected component of $\mathcal{D}$, by construction of the forest.

These properties ensure that for every subtree rooted at a vertex of $\mathcal{C}$, there is a $\mathcal{D}$ vertex at distance at most 2 from the root: otherwise, the vertices at distance 1 from it would be from $\mathcal{C}$ and adjacent only to $\mathcal{C}$ vertices. Moreover, since a $\mathcal{C}$ vertex is not a leaf, it has necessarily some descendant and the reasoning applies. Therefore, by rooting the tree at an arbitrary vertex of $\mathcal{C}$, we can charge every $\mathcal{C}$ vertex to a $\mathcal{D}$ descendant at distance at most 2. As a $\mathcal{D}$ vertex can be charged only by an ancestor at most two levels above it, it is charged at most twice. This ensures that $|\mathcal{C}| \leq 2|\mathcal{D}|$.

Moreover, since $\mathcal{D}$ is a $O(\log n)$ approximation of MDS, $|\mathcal{D}| = O(\log n) \cdot \mathrm{OPT}$. Putting things together, we have $|\mathcal{C} \cup \mathcal{D}| = |\mathcal{C}| + |\mathcal{D}| = O(\log n) \cdot \mathrm{OPT}$. ◀

Combining Lemmas 8, 9 and 10 proves our claim: there is a $\widetilde{O}(n)$ algorithm to maintain a $O(\log n)$ approximation of the Minimum Connected Dominating Set. The main bottleneck of this approach is the time spent by the algorithm in the second phase to query all vertices in $\mathcal{C}$ in order to identify the vertices that are no longer articulation points. In the next section we present an algorithm that overcomes this limitation and is able to identify the necessary vertices more efficiently.

4 A more intricate $\widetilde{O}(\Delta)$ algorithm to restore the minimality of $\mathcal{C}$

In this section we present a more sophisticated algorithm for implementing the phase that guarantees the minimality of the maintained connected dominating set. This gives a proof of Theorem 3. We focus on a single edge update: indeed, when a vertex is added to (or removed from) $\tilde{\mathcal{D}}$, one can simply add (or remove) all its edges one by one. As in the analysis of the complexity in Lemma 9, the amortized number of changes in $\tilde{\mathcal{D}}$ is $\widetilde{O}(1)$. We aim now at proving that the time required for handling a single change is $\widetilde{O}(\Delta)$: for that, we treat edge insertions and deletions to $\tilde{\mathcal{D}}$ one by one, and prove that any edge update can be done in $\widetilde{O}(1)$, which would prove the claimed bound. Our algorithm maintains another spanning forest F of $G[\tilde{\mathcal{D}}]$ (unweighted) using the algorithm from [22].

▶ **Lemma 11.** *The vertices of $\mathcal{C}$ that are not articulation points after the insertion of the edge (v, w) all lie on the tree path $v...w$ of F. Moreover, the removal of any of these vertices results in the other vertices being articulation points again.*

Proof. Let G_b be the graph before the insertion of (v, w), and G_a be the one after. Let u be a vertex that is an articulation point in $G_b[\tilde{\mathcal{D}}]$ but not in $G_a[\tilde{\mathcal{D}}]$. Suppose by contradiction that u is not on the tree path $v...w$: that means that v and w are connected in $G_b[\tilde{\mathcal{D}}] \setminus \{u\}$. Since u is an articulation point in $G_b[\tilde{\mathcal{D}}]$, v is not connected to some vertex x in $G_b[\tilde{\mathcal{D}}] \setminus \{u\}$. But as v and w are connected in $G_b[\tilde{\mathcal{D}}] \setminus \{u\}$, adding the edge (v, w) does not connect v and x and therefore u is still an articulation point after the insertion of the edge. Therefore, all the articulation points that can be removed are in the cycle $v...w, v$. Since they are not articulation points in $G_a[\tilde{\mathcal{D}}]$, they separate $G_b[\tilde{\mathcal{D}}]$ in only two components: one with v, the other with w. Therefore, $v...w, v$ is the only cycle containing v and w, and removing any vertex from it make the articulation points of $G_b[\tilde{\mathcal{D}}]$ be articulations point in $G_a[\tilde{\mathcal{D}}]$, because they disconnect v and w again. ◀

Lemma 11 allows us to focus on the following problem: find a vertex in $\mathcal{C}$ that is no longer an articulation point in $G[\tilde{\mathcal{D}}]$ after the insertion of the edge (v, w). To achieve this, the algorithm maintains for each vertex $v \in \mathcal{C}$ the number $nc(v)$ of connected component of $G[\mathcal{D} \setminus v]$. For $v \notin \mathcal{C}$ we set for convenience $nc(v)$ to be the number of connected component in $G[\mathcal{D} \setminus v]$ plus n. This information can be used as follows: when an edge (v, w) is added, if for one vertex $u \in \mathcal{C}$ it holds $nc(u) = 1$ then u is removed from $\mathcal{C}$ (because it is no longer an articulation point). To identify such a vertex, the algorithm queries for the minimal value along the path $v...w$ in T: if the minimum value is 1, the corresponding vertex is removed from $\mathcal{C}$. This removal makes all the other vertices of the set $\mathcal{C}$ articulation points again: by Lemma 11, the cycle created by the insertion of (v, w) is broken by the deletion of u from $G[\tilde{\mathcal{D}}]$.

Notice that we are only interested in the $nc(v)$ values of the vertices in $\mathcal{C}$, as $nc(v) > n$ for $v \notin \mathcal{C}$. Since we compute a minimum and the values relevant are smaller than n, this is equivalent to ignoring v. The advantage of this offset is that when v becomes part of $\mathcal{C}$, it is sufficient to decrease its value by n to make it consistent. We now show how to keep this value up to date after adding or removing an edge.

Maintaining the $nc(v)$ values in a top-tree. For this purpose, we use the biconnectivity data structure from [22] (called *top-tree*) on the subgraph $G[\tilde{\mathcal{D}}]$. To avoid cumbersome notation, we pretend that we execute the algorithm on G, although the underlying graph on which we execute the algorithm is $G[\tilde{\mathcal{D}}]$. We also assume that the number of vertices remains n throughout the execution, which is simply implemented by removing from G all incident edges from the vertices with no incident edges in $G[\tilde{\mathcal{D}}]$.

We now briefly describe the approach of [22]. The algorithm maintains a spanning forest F of G and assigns a level $\ell(e)$ to each edge e of the graph. Let G_i be the graph composed of F and all edges of level at least i. The levels are attributed such that the following invariant is maintained:

▶ **Invariant 2.** *The maximal number of vertices in a biconnected component of G_i is $\lceil n/2^i \rceil$.*

Therefore the algorithm needs only to consider $\lceil \log_2 n \rceil$ levels. Whenever an edge (v, w) is deleted, one needs to find which vertices in the path $v...w$ in F are still biconnected. We use the following notion to describe the algorithm.

▶ **Definition 12.** *A vertex u is* covered *by a nontree edge (x, y) if it is contained in a tree cycle induced by (x, y). We say that a path $v...w$ is covered at level i if every of its node is in a tree cycle induced by an edge at level greater than i.*

Mark that all the vertices that are covered by a given edge are in the same biconnected component.

When a non-tree edge (v, w) is removed, it may affect the 2-edge connected components along the tree-path $v...w$ in T. To find which vertices are affected, the following algorithm is used in [22]. It first marks the vertices in $v...w$ as no longer covered at level $\ell(v, w)$. Then, it iterates over edges (x, y) that could cover $v...w$, i.e., the ones such that the intersection between $x...y$ and $v...w$ is not empty, and marks the vertices in this intersection as covered. This step is explained in the following function, which is called for all level i from $\ell(v, w)$ down to 0. $meet(v, w, x)$ is the intersection of the tree paths $v...w$, $v...x$ and $x...w$.

Recover(v, w, i). Set $u := v$, and iterate over the vertices of $v...w$ towards w. For each value of u, consider each nontree edge (q, r) with $meet(q, v, w) = u$ and such that $u...q$ is covered at level i. If it is possible without breaking Invariant 2, increase the level of (q, r)

to $i+1$ and mark the edges of $q...r$ covered at level $i+1$. Otherwise, mark them covered at level i and stop. If the phase stopped, start a second symmetric phase with $u = w$ and iterating on $w...v$ towards v.

As shown in [22], this is correct and runs in $O(\log^4)$ amortized time.

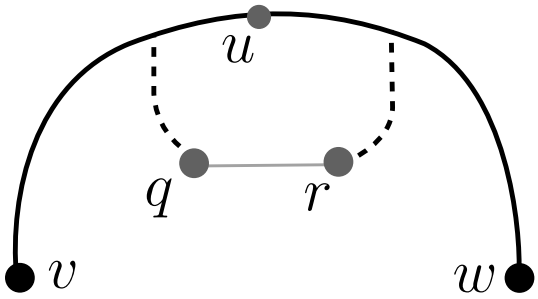


Figure 1 The edge (q, r) covers some node u on the path $v...w$.

In our case, we are interested in the vertices u whose value $nc(u)$ changes. They are exactly those that are still marked as not covered at the end of the process. Indeed, if an edge (q, r) covers a vertex u (see Figure 1), then v and w are still connected in $G[D \setminus u]$, hence the connected component of $G[D \setminus u]$ do not change. However, if u is not covered by any edge, then v and w gets disconnected in $G[D \setminus u]$, thus $nc(u)$ must be updated.

We maintain the $nc(\cdot)$ values in a top-tree, as follows. We call a *segment* a subpath of $v...w$. The idea is to maintain the non-covered segments and decrease the nc values along these at the end of the process. The top-trees allow us to alter the value of a segment of a path in $O(\text{polylog}n)$ time.

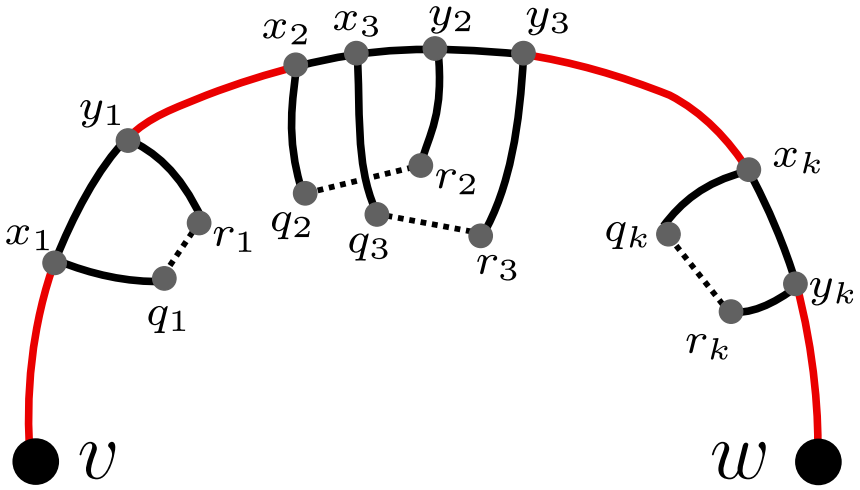


Figure 2 The black segments are covered by edges (q_i, r_i). The red segments are uncovered.

Computing the list of uncovered segments. To find the uncovered segments (in red on Figure 2), we sort the covered ones and take the complementary. Let $(q_1, r_1), ..., (q_k, r_k)$ be the nontree edges considered in the execution of Recover, and let $x_i = LCA(v, q_i)$ and $y_i = LCA(v, r_i)$ (where $LCA(u, v)$ is the lowest common ancestor of u and v in the tree). The covered segments are exactly the (x_i, y_i). Using lowest common ancestor queries, it is

possible to sort those segments according to the position of x_i along the path $v...w$. Given the segments in order, it is then possible to determine the uncovered segments in linear time: they correspond to the complementary of those segments. Answering a lowest common ancestor query on a dynamic tree can be done in $O(\log n)$ (see [29]), hence it is possible to sort the covered segments in time $O(k \log^2 n)$ and to find the uncovered segments with the same complexity.

Since k is the number of edges that move to a higher level during a call to Recover, and the maximum level is $\log n$, the total complexity of computing the uncovered segments is at most $\log^3 n$ per edges. Hence the overall complexity is $O(\log^4 n)$, which is the cost of the function Recover.

Adding an edge. To add an edge, two things are required: first decrease some nc value, and then query if a vertex has a nc value 1. We have to decrease the nc value of a vertex y if and only if its predecessor and its successor along the tree path $v...w$ were not connected in $\mathcal{D} \setminus \{y\}$ before the insertion of (v, w). This turns out to be equivalent to saying that y is not covered: thus, the algorithm needs to compute the list of segments along $v...w$ that were uncovered before the insertion of (v, w). It then must decrease the nc values along these segments, because they become connected. This is analogous to the case of an edge deletion: the latter can be used the following way. First add the edge (v, w) (and make updates to the data structure according to [22]), then delete it using the algorithm from the previous section, with the only difference that, instead of increasing the nc values along the uncovered segments, the algorithm decrease them.

It is then easy to find the minimum nc value along the path $v...w$, using the top-tree. If this value is 1, we can remove the corresponding vertex from $\mathcal{C}$. To remove it, we remove its incident edges one by one, each time updating the nc values of the remaining vertices.

The results of this section are summarized in the following lemma.

▶ **Lemma 13.** *After these updates, $\mathcal{C}$ is minimal. Moreover, the algorithm runs in amortized time $\widetilde{O}(1)$ for a single edge update.*

A direct corollary of this lemma and Lemma 9 is Theorem 3.

▶ **Corollary 14** (Theorem 3)**.** *The whole algorithm to maintain the Connected DS is correct and runs in time $\widetilde{O}(\Delta)$*

Proof. The correctness follows from Lemma 13 and from the correctness of the $\widetilde{O}(n)$ algorithm. As for the running time, the only difference from Lemma 9 is the search for articulation points: this takes $\widetilde{O}(1)$ for each edge added or removed from $\tilde{\mathcal{D}}$, and consequently $\widetilde{O}(\Delta)$ for each node added to or removed from $\tilde{\mathcal{D}}$. This yields that the algorithm takes $\widetilde{O}(\Delta)$ amortized time per update. ◀

5 A $O(\min(\Delta, \sqrt{m}))$ amortized algorithm for Minimal Dominating Set

This section presents a faster algorithm if one is only interested in finding a *Minimal* DS. This is a DS in which it is not possible to remove a vertex, but it can be arbitrarily big. For instance, in a star, the Minimum DS is only one vertex (the center), but its complementary is another minimal DS and has size $n - 1$. This result highlights the difference between MIS and Minimal DS: the best known *deterministic* complexity for MIS is $O(m^{2/3})$, whereas we present here a $O(\sqrt{m})$ algorithm for Minimal DS.

Key idea. When one needs to add a new vertex to the dominating set in order to dominate a vertex v, he can choose a vertex with degree $O(\sqrt{m})$, either v or one of its neighbors (a similar idea appears in Neiman et al. [27]). We present an algorithm with complexity proportional to the degree of the vertex added to the DS: this will give a $O(\min(\Delta, \sqrt{m}))$ algorithm. To analyze the complexity, we follow an argument similar to the one for CDS. At most one vertex is added to the DS at every step, even though several can be removed. Therefore we can pay for the (future) deletion of a vertex at the time it enters the DS.

For a vertex v, $N(v)$ is the set of its neighbors, including v. Let $\mathcal{D}$ be the dominating set maintained by the algorithm. If $v \in \mathcal{D}$ and $u \in N(v)$, we say that v *dominates* u.

For each vertex v, the algorithm keeps this sets up-to-date:

- let $N_{\mathcal{D}}(v)$ be the set of neighbors of v that are in the dominating set $\mathcal{D}$, i.e., $N_{\mathcal{D}}(v) = \mathcal{D} \cap N(v)$
- if $v \in \mathcal{D}$, let OnlyBy(v) be the set of neighbors of v that are dominated only by v, i.e., OnlyBy(v) $= \{u \in N(v) \mid |N_{\mathcal{D}}(u)| = 1\}$

Note that $N_{\mathcal{D}}(v)$ and OnlyBy(v) are useful to check, throughout any sequence of updates, whether a vertex v must be added to or removed from the current dominating set. In particular, if $N_{\mathcal{D}}(v) = \emptyset$ then v is not dominated by any other vertex, and thus it must be included in the dominating set. On the other hand, if OnlyBy(v) $= \emptyset$, all the neighbors of v (v included) are already dominated by some other vertex, and thus v could be removed from the dominating set.

5.1 The algorithm

We now show how to maintain a minimal dominating set $\mathcal{D}$ and the sets $N_{\mathcal{D}}(v)$ and OnlyBy(v), for each vertex v, under arbitrary sequences of edge insertions and deletions. We first describe two basic primitives, which will be used by our insertion and deletion algorithms: adding a vertex to and deleting a vertex from a dominating set $\mathcal{D}$.

Adding a vertex v to $\mathcal{D}$. Following some edge insertion or deletion, it may be necessary to add a vertex v to the current dominating set $\mathcal{D}$. In this case, we scan all its neighbors u and add v to the sets $N_{\mathcal{D}}(u)$. If before the update $N_{\mathcal{D}}(u)$ consisted of a single vertex, say w, we also have to remove u from the set OnlyBy(w), since now u is dominated by both v and w. If OnlyBy(w) becomes empty after this update, we remove w from $\mathcal{D}$ since it is no longer necessary in the dominating set.

Removing a vertex v from $\mathcal{D}$. When a vertex v is removed from the dominating set, we have to remove v from all the sets $N_{\mathcal{D}}(u)$ such that $u \in N(v)$. If after this update $N_{\mathcal{D}}(u)$ consists of a single vertex, say w, we add u to OnlyBy(w).

Edge insertion. Let (u, v) be an edge to be inserted in the graph. We distinguish three cases depending on whether u and v are in the dominating set $\mathcal{D}$ before the insertion. If neither of them is in the dominating set (i.e., $u \notin \mathcal{D}$ and $v \notin \mathcal{D}$), then nothing needs to be done. If both are in the dominating set (i.e., $u \in \mathcal{D}$ and $v \in \mathcal{D}$), then we start by adding v to the set $N_{\mathcal{D}}(u)$. If u was only necessary to dominate itself, we remove u from $\mathcal{D}$. Otherwise, we add u to $N_{\mathcal{D}}(v)$ and perform the same check on v.

If only one of them is in the dominating set (say, $u \notin \mathcal{D}$ and $v \in \mathcal{D}$), we have to add v to the set $N_{\mathcal{D}}(u)$. As in the case of adding a vertex to $\mathcal{D}$, this may cause the removal of another vertex from the dominating set. This can happen only if before the insertion, $N_{\mathcal{D}}(u) = \{w\}$

for some vertex w and $\text{OnlyBy}(w) = \{u\}$: in other terms, u was dominated only by w, and w was in the dominating set only to dominate u. Since after the addition of the edge (u, v) u is also dominated by v, w can be removed from the dominating set.

Edge deletion. Let (u, v) be the edge being deleted from the graph. We distinguish again the same three cases as before. If $u \notin \mathcal{D}$ and $v \notin \mathcal{D}$, nothing needs to be done. If both $u \in \mathcal{D}$ and $v \in \mathcal{D}$, we just have to remove u (resp. v) from the sets $N_{\mathcal{D}}(u)$ and $\text{OnlyBy}(u)$ (resp. $N_{\mathcal{D}}(v)$ and $\text{OnlyBy}(v)$).

If only one of them is in the dominating set, say $u \notin \mathcal{D}$ and $v \in \mathcal{D}$, then we have to remove v from $N_{\mathcal{D}}(u)$. Now, there are two different subcases:

- If $N_{\mathcal{D}}(u) \neq \{v\}$ before the deletion, then nothing needs to be done.
- Otherwise, we have to remove u from $\text{OnlyBy}(v)$: if $\text{OnlyBy}(v) = \emptyset$ after this operation, then we can safely remove v from $\mathcal{D}$. The algorithm must find a new vertex to dominate u: we simply add u to the dominating set.

5.2 Running time

Adding or removing a vertex v from the dominating set can be done in time $O(deg(v))$, where $deg(v)$ is the degree of v in the current graph. While several vertices can be removed from $\mathcal{D}$ at every step, only one can be added (following an edge deletion): the amortized complexity of the algorithm is therefore $O(\Delta)$, where Δ is an upper bound on the degree of the nodes.

Nevertheless, it is possible to chose the vertex to be added to the dominating set more carefully. When the algorithm must find a new vertex to dominate vertex u, it does the following:

- If $deg(u) \leq 2\sqrt{m} + 1$, the algorithm simply adds u to $\mathcal{D}$.
- Otherwise, $deg(u) > 2\sqrt{m} + 1$. The algorithms finds a vertex $w \in N(u)$ with $deg(w) \leq \sqrt{m}$ and adds w to $\mathcal{D}$. Note that such a vertex w can be found by simply scanning only $2\sqrt{m} + 1$ neighbors of u, since (by averaging) at least one of them must have degree smaller than $\sqrt{m}$.

In both cases, the insertion takes time $O(\min(\Delta, \sqrt{m}))$.

When a vertex v is deleted from the dominating set, its degree can be potentially larger than $2\sqrt{m}$. However, when v was added to the dominating set its degree must have been $O(\sqrt{m})$: this implies that many edges were added to v, and we can amortize the work over those edges. More precisely, when a vertex v enters the dominating set, we put a budget $deg(v)$ on it. Every time an edge incident to v is added to the graph, we increase by one this budget, so that when v has to be removed from $\mathcal{D}$, v has a budget larger than $deg(v)$ that can be used for the operation.

References

1 Raghavendra Addanki and Barna Saha. Fully Dynamic Set Cover–Improved and Simple. *arXiv preprint*, 2018. arXiv:1804.03197.

2 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, 2018.

3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. *Fully Dynamic Maximal Independent Set with Sublinear in n Update Time*, pages 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116.

4 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.

5 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 692–711. Society for Industrial and Applied Mathematics, 2016.

6 Sivakumar R Bevan Das and V Bharghavan. Routing in ad-hoc networks using a virtual backbone. In *Proceedings of the 6th International Conference on Computer Communications and Networks (IC3N'97)*, pages 1–20, 1997.

7 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. Society for Industrial and Applied Mathematics, 2018.

8 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM Journal on Computing*, 47(3):859–887, 2018.

9 Sergiy Butenko, Xiuzhen Cheng, Carlos A Oliveira, and Panos M Pardalos. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. In *Recent developments in cooperative control and optimization*, pages 61–73. Springer, 2004.

10 Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks: An International Journal*, 42(4):202–208, 2003.

11 V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. URL: http://www.jstor.org/stable/3689577.

12 Camil Demetrescu and Giuseppe F. Italiano. A New Approach to Dynamic All Pairs Shortest Paths. *J. ACM*, 51(6):968–992, 2004.

13 Ding-Zhu Du and Peng-Jun Wan. *Connected dominating set: theory and applications*, volume 77. Springer Science & Business Media, 2012.

14 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

17 Leonidas Guibas, Nikola Milosavljević, and Arik Motskin. Connected dominating sets on dynamic geometric graphs. *Computational Geometry*, 46(2):160–172, 2013.

18 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 537–550. ACM, 2017.

19 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for Maximal Independent Set and other problems. *arXiv preprint*, 2018. arXiv:1804.01823.

20 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 548–557. IEEE, 2013.

21 Johan Håstad. Clique is hard to approximate withinn 1- ε. *Acta Mathematica*, 182(1):105–142, 1999.

22 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic Deterministic Fully-dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-edge, and Biconnectivity. *J. ACM*, 48(4):723–760, 2001.

23 Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

24 Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.

25 Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

26 D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, October 2017.

27 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

28 E Sampathkumar and HB Walikar. The connected domination number of a graph. *J. Math. Phys*, 1979.

29 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of computer and system sciences*, 26(3):362–391, 1983.

30 S. Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, October 2016. doi:10.1109/FOCS.2016.43.

31 Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14. ACM, 1999.

On Kernelization for Edge Dominating Set under Structural Parameters

Eva-Maria C. Hols
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
hols@informatik.hu-berlin.de

Stefan Kratsch
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Abstract

In the NP-hard EDGE DOMINATING SET problem (EDS) we are given a graph $G = (V, E)$ and an integer k, and need to determine whether there is a set $F \subseteq E$ of at most k edges that are incident with all (other) edges of G. It is known that this problem is fixed-parameter tractable and admits a polynomial kernelization when parameterized by k. A caveat for this parameter is that it needs to be large, i.e., at least equal to half the size of a maximum matching of G, for instances not to be trivially negative. Motivated by this, we study the existence of polynomial kernelizations for EDS when parameterized by *structural* parameters that may be much smaller than k.

Unfortunately, at first glance this looks rather hopeless: Even when parameterized by the deletion distance to a disjoint union of paths P_3 of length two there is no polynomial kernelization (under standard assumptions), ruling out polynomial kernelizations for many smaller parameters like the feedback vertex set size. In contrast, somewhat surprisingly, there is a polynomial kernelization for deletion distance to a disjoint union of paths P_5 of length *four*. As our main result, we fully classify for all finite sets $\mathcal{H}$ of graphs, whether a kernel size polynomial in $|X|$ is possible when given X such that each connected component of $G - X$ is isomorphic to a graph in $\mathcal{H}$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Edge dominating set, kernelization, structural parameters

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.36

Related Version A full version is available at [16], `https://arxiv.org/abs/1901.03582`.

Funding *Eva-Maria C. Hols*: Supported by DFG Emmy Noether-grant (KR 4286/1).

1 Introduction

In the EDGE DOMINATING SET problem (EDS) we are given a graph $G = (V, E)$ and an integer k, and need to determine whether there is a set $F \subseteq E$ of at most k edges that are incident with all (other) edges of G. It is known that this is equivalent to the existence of a maximal matching of size at most k. The EDGE DOMINATING SET problem is NP-hard but admits a simple 2-approximation by taking any maximal matching of G. It can be solved in time $\mathcal{O}^*(2.2351^k)$[1] [18], making it fixed-parameter tractable for parameter k. Additionally, for EDS any given instance (G, k) can be efficiently reduced to an equivalent one (G', k') with only $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^3)$ edges [33] (this is called a *kernelization*).

The drawback of choosing the solution size k as the parameter is that k is large on many types of easy instances. This has been addressed for many other problems by turning to so called *structural parameters* that are independent of the solution size. Two lines of research in this direction have yielded polynomial kernelizations for several other NP-hard problems.

[1] $\mathcal{O}^*$-notation hides factors that are polynomial in the input size.

One possibility is to choose the parameter as the size of a set X such that $G - X$ belongs to some class $\mathcal{C}$ where the problem in question can be efficiently solved; such sets X are called *modulators*. The other possibility is to parameterize above some lower bound for the solution, i.e., the parameter is the difference between the solution size k and the lower bound.

The VERTEX COVER problem, where, given a graph G and an integer k, we are asked whether there are k vertices that are incident with all edges, has been successfully studied under different structural parameters. It had been observed that VERTEX COVER is FPT parameterized by the size of a modulator to a class $\mathcal{C}$ when one can solve vertex cover on graphs that belong to $\mathcal{C}$ in polynomial time; e.g. if $\mathcal{C}$ is the graph class of forests or, more generally, of bipartite or Kőnig graphs. Furthermore, there also exist kernelizations for VERTEX COVER parameterized by modulators to some graph classes $\mathcal{C}$. The first of a number of such results is due to Jansen and Bodlaender [19] who gave a kernelization with $\mathcal{O}(\ell^3)$ vertices where ℓ is the size of a (minimum) feedback vertex set of the input graph. Clearly, the solution size k cannot be bounded in terms of ℓ alone because forests already have arbitrarily large minimum vertex covers. This result has been generalized, e.g., for parameterization by the size of an odd cycle transversal [24].

There are also parameterized algorithms for VERTEX COVER above lower bounds that address the specific complaint about the seemingly unnecessarily large parameter value k in many graph classes. It was first shown that VERTEX COVER parameterized by $\ell = k - MM$ where MM stands for the size of a maximum matching is FPT [27]. In other words, the parameter value ℓ is the difference between k and the obvious lower bound. This has been improved to work also for parameterization by $\ell = k - LP$ where LP stands for the minimum *fractional vertex cover* (as determined by the LP relaxation) [5, 25] and, recently, even for parameter $\ell = k - (2LP - MM)$ [13]. All of these above lower bound parameterizations of VERTEX COVER also have randomized polynomial kernelizations [24, 23].

Motivated by the number of positive results for VERTEX COVER parameterized by structural parameters we would like to know whether some of these results carry over to the related but somewhat more involved EDGE DOMINATING SET problem.

Our results. For kernelization subject to the size of a modulator to some tractable class $\mathcal{C}$ there is bad news: Even if $\mathcal{C}$ contains only the disjoint unions of paths of length two (consisting of three vertices each) we show that there is no polynomial kernelization for parameterization by $|X|$ with $G - X \in \mathcal{C}$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ (and the polynomial hierarchy collapses). The same is true when $\mathcal{C}$ contains at least all disjoint unions of triangles. Thus, for the usual program of studying modulators to well-known *hereditary* graph classes $\mathcal{C}$ there is essentially nothing left to do because the only permissible connected components would have one or two vertices.[2] That said, as the next result shows, this perspective would ignore an interesting landscape of positive and negative results that can be obtained by permitting certain forms of connected components in $G - X$ but not necessarily all induced subgraphs thereof, i.e., by dropping the requirement that $\mathcal{C}$ needs to be hereditary (closed under induced subgraphs).

Indeed, there is, e.g., a polynomial kernelization for parameter $|X|$ when all connected components of $G - X$ are paths of length *four*. This indicates that the structure even of constant-sized components permitted in $G - X$ determines in a nontrivial way whether or not there is a polynomial kernelization. Note the contrast with VERTEX COVER where a modulator to component size d admits a kernelization with $\mathcal{O}(k^d)$ vertices for each fixed d. Naturally, we are interested in finding out exactly which cases admit polynomial kernelizations.

[2] This very modest case actually admits a polynomial kernelization.

This brings us to our main result. For $\mathcal{H}$ a set of graphs, say that G is an $\mathcal{H}$-component graph if each connected component of G is isomorphic to some graph in $\mathcal{H}$. We fully classify the existence of polynomial kernelizations for parameterization by the size of a modulator to the class of $\mathcal{H}$-component graphs for all finite sets $\mathcal{H}$. To clarify, the input consists of (G, k, X) such that $G - X$ is an $\mathcal{H}$-component graph and the task is to determine whether G has an edge dominating set of size at most k; the parameter is $|X|$. Note that these problems are fixed-parameter tractable for all finite sets $\mathcal{H}$ because G has treewidth at most $|X| + \mathcal{O}(1)$.

▶ **Theorem 1.** *For every finite set $\mathcal{H}$ of graphs, the* EDGE DOMINATING SET *problem parameterized by the size of a given modulator X to the class of $\mathcal{H}$-component graphs falls into one of the following two cases:*

1. *It has a kernelization with $\mathcal{O}(|X|^d)$ vertices, $\mathcal{O}(|X|^{d+1})$ edges, and size $\mathcal{O}(|X|^{d+1} \log |X|)$. Moreover, unless* NP $\subseteq$ coNP/poly*, there is no kernelization to size $\mathcal{O}(|X|^{d-\varepsilon})$ for any $\varepsilon > 0$. Here $d = d(\mathcal{H})$ is a constant depending only on the set $\mathcal{H}$.*
2. *It has no polynomial kernelization unless* NP $\subseteq$ coNP/poly*.*

To obtain the classification one needs to understand how connected components of $G - X$ that are isomorphic to some graph $H \in \mathcal{H}$ can interact with a solution for G, and to derive properties of H that can be leveraged for kernelizations or lower bounds for kernelization. Crucially, edge dominating sets for G may contain edges between X and components of $G - X$. From the perspective of such a component (isomorphic to H) this is equivalent to first covering edges incident with some vertex set $B \subseteq V(H)$ (the endpoints of chosen edges to X) and then covering the remaining edges by a minimum edge dominating set for $H - B$. Depending on the size of a minimum edge dominating set of $H - B$ and further properties of H, such a set B may be used to rule out any polynomial kernelizations or to give a lower bound of $\mathcal{O}(|X|^{d-\varepsilon})$ for the kernel size, where $d = |B|$. Conversely, absence of such sets or an upper bound for their size can be leveraged for kernelizations. Some sets B may make others redundant, further complicating both upper and lower bounds.

For a given finite set $\mathcal{H}$ of graphs, the lower bound obtained from the classification is simply the strongest one over all $H \in \mathcal{H}$. If this does not already rule out a polynomial kernelization then, for each $H \in \mathcal{H}$, we can reduce the number of components isomorphic to H to $\mathcal{O}(|X|^{d(H)})$ where $d(H)$ depends only on H. Moreover, we also have the almost matching lower bound of $\mathcal{O}(|X|^{d(H)-\varepsilon})$, assuming NP $\not\subseteq$ coNP/poly. The value $d(\mathcal{H})$ is the maximum over all $d(H)$ for $H \in \mathcal{H}$ that yield such a polynomial lower bound; it can be computed in time depending only on $\mathcal{H}$, i.e., in constant time for each fixed $\mathcal{H}$.

Regarding parameterization above lower bounds, we prove that it is NP-hard to determine whether a graph G has an edge dominating set of size equal to the lower bound of half the size of a maximum matching. This rules out any positive results for parameter $\ell = k - \frac{1}{2}MM$.

Related work. The parameterized complexity of EDGE DOMINATING SET has been studied in a number of papers [9, 10, 31, 32, 33, 34, 8, 18]. Structural parameters were studied, e.g., by Escoffier et al. [8] who obtained an $\mathcal{O}^*(1.821^\ell)$ time algorithm where ℓ is the vertex cover size of the input graph, and by Kobler and Rotics [22] who gave a polynomial-time algorithm for graphs of bounded clique-width. It is easy to see that EDS is fixed-parameter tractable with respect to the treewidth of the input graph. Prieto [26] was the first to find a kernelization to $\mathcal{O}(k^2)$ vertices for the standard parameterization by k; this was improved to $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^3)$ edges by Xiao et al. [33] and further tweaked by Hagerup [15]. Our work appears to be the first to study the existence of polynomial kernelizations for EDS subject to structural parameters, though some lower bounds, e.g., for parameter treewidth are obvious.

Classically, EDGE DOMINATING SET remains NP-hard on planar cubic graphs, bipartite graphs with maximum degree three [36]. This implies NP-hardness already for $|X| = 0$ when considering parameterization by a modulator to any graph class containing this special case. EDGE DOMINATING SET has also been studied from the perspective of approximation [12, 4, 3, 29, 8], enumeration [20, 14, 21], and exact exponential-time algorithms [28, 32, 30, 35].

Organization. We begin with some preliminaries in Section 2. Section 3 provides some intuition for the main result by proving the lower bound for EDGE DOMINATING SET parameterized by the size of a modulator to a P_3-component graph as well as the polynomial kernelization for parameterization by the size of a modulator to a P_5-component graph. Section 4 gives a detailed statement of the main result including the required definitions to determine which result applies for any given set $\mathcal{H}$. Due to space restrictions, the proof of the main result and the hardness proof for parameter $\ell = k - \frac{1}{2}MM$ are deferred to the full version of this work. We conclude in Section 5.

2 Preliminaries

We use standard graph notation as given by Diestel [7]. In particular, for a graph $G = (V, E)$ we let $N(v) = \{u \in V \mid \{u, v\} \in E\}$ and $N[v] = N(v) \cup \{v\}$; similarly, $N[X] = \bigcup_{x \in X} N[x]$ and $N(X) = N[X] \setminus X$. We let $E(X, Y) = \{\{x, y\} \mid x \in X, y \in Y\}$ and we let $\delta(v) = \{\{u, v\} \mid u \in V, \{u, v\} \in E\}$. By $G[X]$ we denote the induced subgraph of G on vertex set X and by $G - X$ the induced subgraph on vertex set $V \setminus X$; we let $G - v = G - \{v\}$. We denote the size of a minimum edge dominating set of a graph G by $\text{EDS}(G)$.

Let $\mathcal{H}$ be a set of graphs. We say that a graph G is an *$\mathcal{H}$-component graph* if each connected component of G is isomorphic to some graph in $\mathcal{H}$. Clearly, disconnected graphs in $\mathcal{H}$ do not affect which graphs G are $\mathcal{H}$-component graphs and, thus, our proofs need only consider the connected graphs $H \in \mathcal{H}$. We write H-component graph rather than $\{H\}$-component graph for single (connected) graphs H.

Let $[n]$ denote the set $\{1, 2, \ldots, n\}$.

Parameterized complexity. A *parameterized problem* $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$ where Σ is any finite set. The second component k of instances (x, k) is called the *parameter*. A parameterized problem $\mathcal{Q}$ is *fixed-parameter tractable* if there is an algorithm that correctly solves all instances (x, k) in time $f(k)|x|^c$ where f is a computable function and c is a constant independent of k. A *kernelization* for $\mathcal{Q}$ is an efficient algorithm that, given an instance (x, k), takes time polynomial in $|x| + k$ and returns an instance (x', k') of size at most $f(k)$ such that $(x, k) \in \mathcal{Q}$ if and only if $(x', k') \in \mathcal{Q}$ where f is a computable function. The function f is also called the *size* of the kernelization and a kernelization is polynomial (resp. linear) if $f(k)$ is polynomially (resp. linearly) bounded in k.

We use the notion of a cross-composition [2], which is a convenient front-end for the seminal kernel lower bound framework of Bodlaender et al. [1] and Fortnow and Santhanam [11]. A relation $\mathcal{R} \subseteq \Sigma^* \times \Sigma^*$ is a *polynomial equivalence relation* if equivalence of two strings $x, y \in \Sigma^*$ can be tested in time polynomial in $|x| + |y|$ and if $\mathcal{R}$ partitions any finite set $S \subseteq \Sigma^*$ into a number of classes that is polynomially bounded in the largest element of S.

▶ **Definition 2** ((OR-)cross-composition [2]). *Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on Σ^*, and let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. An* (OR-)cross-composition *of L into $\mathcal{Q}$ (with respect to $\mathcal{R}$) is an algorithm that, given t instances $x_1, \ldots, x_t \in$*

Σ^* *of* L *belonging to the same equivalence class of* $\mathcal{R}$*, takes time polynomial in* $\sum_{i=1}^{t} |x_i|$ *and outputs an instance* $(y, k) \in \Sigma^* \times \mathbb{N}$ *such that the following hold:*

- *"PB": The parameter value* k *is polynomially bounded in* $\max_{i=1}^{t} |x_i| + \log t$*.*
- *"OR": The instance* (y, k) *is yes for* $\mathcal{Q}$ *if and only if* at least one *instance* x_i *is yes for* L*.*

An (OR-)cross-composition of L into $\mathcal{Q}$ of cost $f(t)$ *instead satisfies "OR" and "CB":*

- *"CB": The parameter value* k *is bounded by* $\mathcal{O}(f(t) \cdot (\max_{i=1}^{t} |x_i|)^c)$*, where* c *is some constant independent of* t*.*

If L is NP-hard then both forms of cross-compositions are known to imply lower bounds for kernelizations for $\mathcal{Q}$. Theorem 4 additionally builds on Dell and van Melkebeek [6].

▶ **Theorem 3** ([2, Corollary 3.6.])**.** *If an* NP*-hard language* L *has a cross-composition to* $\mathcal{Q}$ *then* $\mathcal{Q}$ *admits no polynomial kernelization or polynomial compression unless* NP $\subseteq$ coNP/poly*.*

▶ **Theorem 4** ([2, Theorem 3.8.])**.** *Let* $d, \varepsilon > 0$*. If an* NP*-hard language* L *has a cross-composition into* $\mathcal{Q}$ *of cost* $f(t) = t^{1/d+o(1)}$*, where* t *is the number of instances, then* $\mathcal{Q}$ *has no polynomial kernelization or polynomial compression of size* $\mathcal{O}(k^{d-\varepsilon})$ *unless* NP $\subseteq$ coNP/poly*.*

All our composition-based proofs use for L the NP-hard MULTICOLORED CLIQUE problem. Therein we are given a graph $G = (V, E)$, an integer k, and a partition of V into k sets $V_1, \ldots, V_k$ of equal size; we need to determine whether there is a clique of size k in G that contains exactly one vertex from each set V_i. Such a set X is called a *multicolored* k*-clique*.

3 EDS parameterized by the size of a modulator to a P_3- resp. P_5-component graph

In this section we study the difference of EDGE DOMINATING SET parameterized by the size of a modulator to a P_3-component graph and EDGE DOMINATING SET parameterized by the size of a modulator to a P_5-component graph, which are both more restrictive than parameterization by size of a feedback vertex set (modulator to a forest). Note that the latter is FPT, because the treewidth is at most the size of the feedback vertex set plus one and EDGE DOMINATING SET parameterized by the treewidth is FPT. Hence, EDGE DOMINATING SET parameterized by the above modulators is FPT too.

First, we show that EDGE DOMINATING SET parameterized by the size of a modulator to a P_3-component graph has no polynomial kernelization unless NP $\subseteq$ coNP/poly. This rules out polynomial kernelizations for a large number of interesting parameters like feedback vertex set size or size of a modulator to a linear forest. Somewhat surprisingly, we then show that when parameterized by the modulator to a P_5-component graph we do get a polynomial kernelization.

3.1 Lower bound for EDS parameterized by the size of a modulator to a P_3-component graph

We give a kernelization lower bound for EDGE DOMINATING SET parameterized by the size of a modulator X, such that deleting X results in a disjoint union of P_3's. To prove this we give a cross-composition from MULTICOLORED CLIQUE.

▶ **Theorem 5.** EDGE DOMINATING SET *parameterized by the size of a modulator to a* P_3*-component graph (and thus also parameterized by the size of a modulator to a linear forest) does not admit a polynomial kernelization unless* NP $\subseteq$ coNP/poly*.*

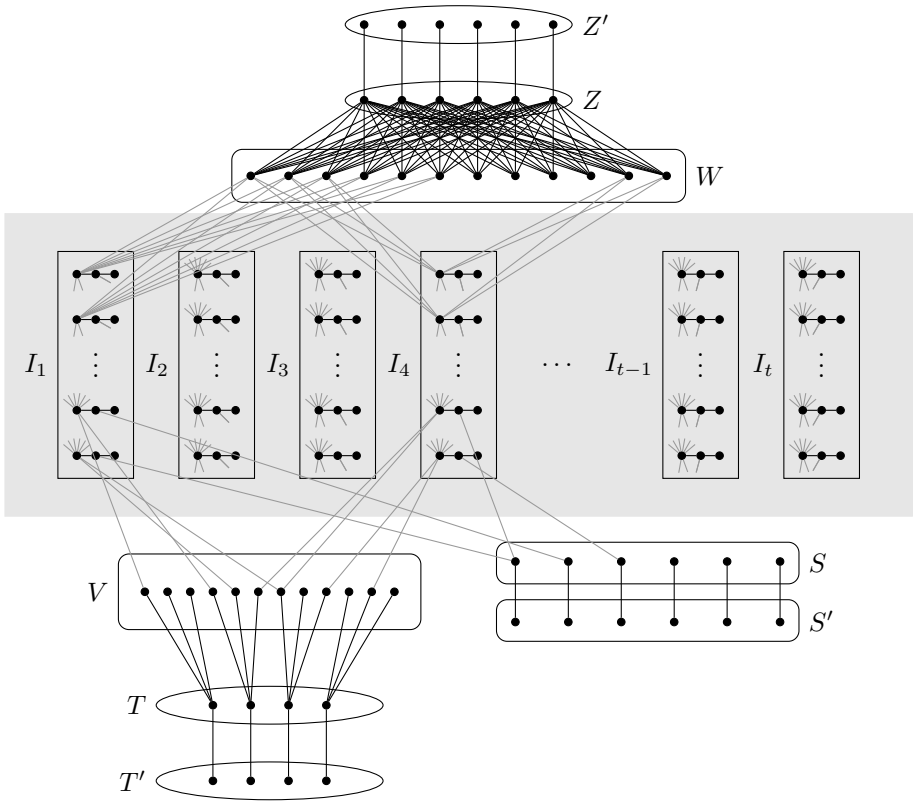


Figure 1 Construction of the graph G' with $k = 4$, where $X' = W \cup Z \cup Z' \cup V \cup T \cup T' \cup S \cup S'$.

Proof. To prove the theorem we give a cross-composition from the NP-hard MULTICOLORED CLIQUE problem to EDGE DOMINATING SET parameterized by the size of a modulator to a P_3-component graph. Input instances are of the form (G_i, k_i) where G_i comes with a partition of the vertex set into k color classes. (Since the color classes are of equal size it holds that $k \leq |V(G_i)|$.) For the polynomial equivalence relation $\mathcal{R}$ we take the relation that puts two instances (G_1, k_1), (G_2, k_2) of MULTICOLORED CLIQUE in the same equivalence class if $k_1 = k_2$ and $|V(G_1)| = |V(G_2)|$. It is easy to check that $\mathcal{R}$ is a polynomial equivalence relation. (Instances with size at most N have at most N vertices. Thus, we get at most N^2 classes for instances of size at most N.)

Let a sequence of instances $I_i = (G_i, k)_{i=1}^t$ of MULTICOLORED CLIQUE be given that are equivalent under $\mathcal{R}$. We identify the color classes of the input graphs so that all graphs have the same vertex set V and the same color classes $V_1, V_2, \ldots, V_k$. Let $n := |V_i|$ be the number of vertices of each color class; thus, each instance has $|V| = n \cdot k$ vertices. We assume w.l.o.g. that every instance has at least one edge in $E(V_p, V_q)$ for all $1 \leq p < q \leq k$; otherwise, this instance would be a trivial no instance and we can delete it. Furthermore, we can assume w.l.o.g. that $t = 2^s$ for an integer s, since we may copy some instances if needed (while at most doubling the number of instances and increasing $\log t$ by less than one).

Now, we construct an instance (G', k', X') of EDGE DOMINATING SET parameterized by the size of a modulator to a P_3-component graph, where the size of X' is polynomially bounded in $n + k + s$ (see Figure 1 for an illustration). We add a set V consisting of $k \cdot n$ vertices to graph G' which represents the vertices of the t instances. The set V is partitioned into the k color classes $V_1, V_2, \ldots, V_k$. To choose which vertices are contained in a clique of

size k, we add a set $T = \{t_1, t_2, \ldots, t_k\}$ and a set $T' = \{t'_1, t'_2, \ldots, t'_k\}$, each of size k, to G'. We make $t_j \in T$, with $j \in [k]$, adjacent to all vertices in V_j and to vertex $t'_j \in T'$. Next, we add two sets Z, Z', each of size s, and a set W of size $2s$ to G' and add edges to G' such that each vertex in Z has exactly one private neighbor in Z' and is adjacent to all vertices in W. The set W contains $\binom{2s}{s} \geq 2^s$ different subsets of size s. For each instance (G_i, k), with $i \in [t]$, we pick a different subset of size s of W and denote it by $W(i)$. For all $1 \leq p < q \leq k$ we add a vertex $s_{p,q}$ and a vertex $s'_{p,q}$ to G'; these will correspond to the edge sets $E(V_p, V_q)$. Let $S = \{s_{p,q} \mid 1 \leq p < q \leq k\}$ and $S' = \{s'_{p,q} \mid 1 \leq p < q \leq k\}$. We make vertex $s_{p,q}$ adjacent to vertex $s'_{p,q}$ for all $1 \leq p < q \leq k$. For each graph G_i, for $i \in [t]$, we add $|E(G_i)|$ paths of length two to the graph G'; every P_3 represents exactly one edge of the graph G_i. Let $P_i^e = u_{i,1}^e u_i^e u_{i,2}^e$ denote the path of instance $i \in [t]$ that represents edge $e \in E(G_i)$. Finally, we make vertices in P_i^e, with $i \in [t]$ and $e \in E(G_i)$, adjacent to vertices in the sets W, V, and S as follows: We make vertex $u_{i,1}^e$ of path P_i^e, with $i \in [t]$, which represents edge $e = \{x, y\} \in E(G_i)$ adjacent to the vertices x, y in V and to all vertices in the set $W(i) \subseteq W$. Additionally, we make vertex u_i^e adjacent to vertex $s_{p,q}$ where $1 \leq p < q \leq k$ such that $e \in E(V_p, V_q)$.

The set X' is defined to contain all vertices that do not participate in the paths P_i^e, i.e., $X' = W \cup Z \cup Z' \cup V \cup T \cup T' \cup S \cup S'$. Clearly, $G - X'$ is a P_3-component graph and $|X'| = 4s + k \cdot n + 2k + 2 \cdot \binom{k}{2}$. Let $k' = k + s + \sum_{i=1}^{t} |E(G_i)|$. Note that the size of k' can depend linearly on the number of instances, because our parameter is the size of X', which is polynomially bounded in $n + s$, as $k \leq n$. We return the instance (G', k', X'); clearly, this instance can be generated in polynomial time.

Now, we have to show that (G', k', X') is a YES-instance of EDS if and only if there exists an $i^* \in [t]$ such that (G_{i^*}, k) is a YES-instance of MULTICOLORED CLIQUE.

($\Rightarrow$:) Assume first that (G', k', X') is yes for EDS and that there exists an edge dominating set F of size at most k' in G'. We can always pick F such that it fulfills the following properties (most hold for all solutions of size at most k'):

1. The vertex sets S, T, and Z must be subsets of $V(F)$: E.g., for each edge $\{z, z'\}$ with $z \in Z$ and $z' \in Z'$ the set $V(F)$ must contain z or z'; if it contains z' then $\{z, z'\} \in F$ as it is the only edge incident with z'; either way we get $z \in V(F)$. The same applies for S and S', and for T and T'.
2. Because $S, T, Z \subseteq V(F)$ but $S \cup T \cup Z$ is an independent set, the set F must contain at least $|S|$ edges incident with S, $|T|$ edges incident with T, and $|Z|$ edges incident with Z. By straightforward replacement arguments we may assume that F contains exactly the following edges incident with $S \cup T \cup Z$: $|T|$ edges between T and V, $|Z|$ edges between Z and W, and $|S|$ edges between S and middle vertices u_i^e of P_3's in $G' - X'$. Furthermore, we can assume that these edges are a matching, because no color class is empty, no edge set $E(V_p, V_q)$ is empty, and Z is adjacent to all vertices in W.
3. For each $P_i^e = u_{i,1}^e u_i^e u_{i,2}^e$, which represents the edge e of instance (G_i, k), at least vertex u_i^e must be an endpoint of an edge in F: Indeed, to cover the edge $\{u_i^e, u_{i,2}^e\}$ one of its two vertices must be in $V(F)$. Similar to Property 1 above, if $u_{i,2}^e \in V(F)$ then F must contain its sole incident edge $\{u_i^e, u_{i,2}^e\}$ and, hence, $u_i^e \in V(F)$.
4. An edge in F cannot have its endpoints in two different P_3's of $G' - X'$ because no such edges exist.

Let $F_T = F \cap E(T, V)$, let $F_Z = F \cap E(Z, W)$, let $F_S = F \cap E(S, \{u_i^e \mid i \in [t], e \in E(G_i)\})$, and let $F_R = F \setminus (F_T \cup F_Z \cup F_S)$. Hence, due to Properties 1 and 2, we have

$$|F_R| \leq k' - |F_T| - |F_Z| - |F_S| \leq \sum_{i=1}^{t} |E(G_i)| - \binom{k}{2}.$$

By Property 3, all vertices u_i^e are endpoints of edges in F. Among $F_T \cup F_Z \cup F_S$ this can only be true for the $|S| = \binom{k}{2}$ edges in F_S. Since there are exactly $\sum_{i=1}^{t} |E(G_i)|$ vertices u_i^e, which is (greater or) equal to $|F_R| + |F_S|$, and there are no edges connecting different such vertices, each edge in $F_R \cup F_S$ is incident with a private vertex u_i^e. This also implies that all edges in F_R have no endpoints in $V \cup W$ as those sets are not adjacent to any vertex u_i^e. Thus, in W exactly the $|Z| = s$ endpoints of F_Z are endpoints of F. Similarly, in V exactly the $|T| = k$ endpoints of F_T are endpoints of F; let $X \subseteq V$ denote this set of k vertices. Observe that by construction of G' the set X contains exactly one vertex from each color class, because $t_j \in T$, for $j \in [k]$, is only adjacent to vertices of V_j.

Now, consider any path $P_i^e = u_{i,1}^e u_i^e u_{i,2}^e$ where u_i^e is an endpoint of an edge $f \in F_S$. Clearly, the other endpoint of f lies in S, and, by the above accounting, no other edge of F is incident with $u_{i,1}^e$ or $u_{i,2}^e$. In particular, this implies that all neighbors of $u_{i,1}^e$ in W and V must be endpoints of edges in F. If $e = \{x, y\}$ then these neighbors of $u_{i,1}^e$ are the set $W(i) \subseteq W$ and the vertices $x, y \in V$, and, by construction of G', the edge $\{x, y\}$ must exist in G_i. Thus, $W(i) \cup \{x, y\} \subseteq V(F)$ which implies that $x, y \in X$.

Repeating this argument for all $|S| = \binom{k}{2}$ paths of this type, we can conclude the following: (1) All paths correspond to the same instance $i^* \in [t]$ because we require $W(i) \subseteq V(F)$, but exactly $|Z| = |W(i^*)| = s$ such vertices are in $V(F)$. (Different values of i would require different sets $W(i)$, exceeding size s.) (2) There are $\binom{k}{2}$ edges of G_{i^*} represented by the paths and all their endpoints must be in $X = V \cap V(F)$. Since $|X| = k$, the edges must form a clique of size k on vertex set X in G_{i^*}. We already observed above that X contains exactly one vertex per color class, hence, instance (G_{i^*}, k) is yes, as claimed.

($\Leftarrow$:) For the other direction, assume that for some $i^* \in [t]$ the MULTICOLORED CLIQUE instance (G_{i^*}, k) is a YES-instance. Let $X = \{x_1, x_2, \ldots, x_k\} \subseteq V$ be a multicolored clique of size k in G_{i^*} with $x_j \in V_j$ for $j \in [k]$, let E' be the set of edges of the clique X, and let $e_{p,q} = \{x_p, x_q\}$ for $1 \leq p < q \leq k$. We construct an edge dominating set F of G' of size at most k' as follows: First we add the k edges $\{t_j, x_j\}$ for $j \in [k]$ between T and $X \subseteq V$; thus, $T \cup X \subseteq V(F)$. We then add a maximum matching (of size s) between $W(i^*) \subseteq W$ and Z to the set F. This matching saturates $W(i^*)$ and Z because $|Z| = |W(i^*)| = s$; thus, $W(i^*) \cup Z \subseteq V(F)$. Next, we add the edges $\{u_{i^*}^{e_{p,q}}, s_{p,q}\}$ for all edges $e_{p,q} \in E'$, with $1 \leq p < q \leq k$, to the set F; hence $S \subseteq V(F)$. Finally, for all other paths P_i^e, with $i \in [t]$, $e \in E(G_i)$, and $i \neq i^*$ or $e \notin E'$, we add the edge $\{u_{i,1}^e, u_i^e\}$ to F. (We have thus selected exactly one edge incident with each path of $G' - X'$.) By construction, it holds that $|F| = k + s + \sum_{i=1}^{t} |E(G_i)| = k'$.

It remains to show that F is indeed an edge dominating set of G'. To prove this, it suffices to show that $V(G') - V(F)$ is an independent set in G'. We already know that $S \cup T \cup W(i^*) \cup X \cup Z \subseteq V(F)$. Moreover, $V(F)$ contains the middle vertex u_i^e for all P_3's in $G' - X'$ and it contains $u_{i,1}^e$ for all P_3's that do not correspond to an edge of the clique X (i.e., with $i \neq i^*$ or with $i = i^*$ but $e \neq e_{p,q}$ for any $1 \leq p < q \leq k$). The sets S', T', and Z' are independent sets whose neighborhoods S, T, and Z are subsets of $V(F)$. Similarly, all vertices $u_{i,2}^e$ have their single neighbor u_i^e in $V(F)$. Thus, only vertices in $W \setminus W(i^*)$ and $V \setminus X$ could possibly be adjacent to vertices $u_{i^*,1}^{e_{p,q}}$, which correspond to the edges of $G_{i^*}[X]$, in $G' - V(F)$, but this can be easily refuted: Indeed, each $u_{i^*,1}^{e_{p,q}}$ is adjacent only to x_p and x_q in V, which are both in $X \subseteq V(F)$, and to the vertices in $W(i^*)$ in W, but $W(i^*) \subseteq V(F)$ as well. Thus $V(G') - V(F)$ is an independent set in G' and hence F is an edge dominating set for G' of size at most k'. Thus, (G', k', X') is yes, which completes the cross-composition.

By Theorem 3 the cross-composition from MULTICOLORED CLIQUE implies the claimed lower bound for kernelization. ◀

We proved that EDGE DOMINATING SET parameterized by the size of a modulator to a P_3-component graph has no polynomial kernelization unless NP $\subseteq$ coNP/poly. A similar proof establishes the same lower bound for modulators to K_3-component graphs. As mentioned in the introduction this rules out polynomial kernels using modulators to essentially all interesting hereditary graph classes.[3]

3.2 Polynomial kernelization for EDS parameterized by the size of a modulator to a P_5-component graph

To illustrate why other, non-hereditary, sets $\mathcal{H}$ may well allow polynomial kernelizations for parameterization by the size of a modulator X to an $\mathcal{H}$-component graph, we sketch a simple kernelization for the case of $\mathcal{H} = \{P_5\}$, i.e., when components of $G - X$ are isomorphic to the path of length four. This does not use the full generality of the kernelization obtained in Section 4 because P_5 does not have any (later called) uncovered vertices or (later called) strongly beneficial sets (which are the main source of complication).

For the kernelization we need the following theorem which is due to Hopcroft and Karp [17]. The second claim of the theorem is not standard (but well known).

▶ **Theorem 6** ([17]). *Let G be an undirected bipartite graph with partition R and S, on n vertices and m edges. Then we can find a maximum matching of G in time $\mathcal{O}(m\sqrt{n})$. Furthermore, in time $\mathcal{O}(m\sqrt{n})$ we can find either a maximum matching that saturates R or a set $Y \subseteq R$ such that $|N_G(Y)| < |Y|$ and such that there exists a maximum matching M in $G - N_G[Y]$ that saturates $R \setminus Y$.*

▶ **Theorem 7.** EDGE DOMINATING SET *parameterized by the size of a given modulator X to a P_5-component graph admits a kernelization with $\mathcal{O}(|X|)$ vertices.*

Proof. Let (G, k, X) be an instance of EDGE DOMINATING SET parameterized by the size of a modulator to a P_5-component graph, and let $\mathcal{C}$ be the set of connected components of $G - X$. We construct a bipartite graph G_B where one part is the set X, the other part consists of one vertex s_P for every connected component P in $\mathcal{C}$, and where there is an edge between $x \in X$ and s_P with $P = w_1w_2w_3w_4w_5 \in \mathcal{C}$ if and only if x is adjacent to a vertex of P that is not the middle vertex w_3. Now, we apply Theorem 6 to obtain either a maximum matching in G_B that saturates X or a set $Y \subseteq X$ such that $|N_{G_B}(Y)| < |Y|$ and such that there exists a maximum matching in $G_B - N_{G_B}[Y]$ that saturates $X \setminus Y$. If there exists a maximum matching in G_B that saturates X then let $X_1 = X$ and $X_2 = \emptyset$. Otherwise, if there exists a set Y with the above properties then let $X_1 = X \setminus Y$ and $X_2 = Y$. Observe that X_2 also contains the vertices in X that are only adjacent to middle vertices of components in $\mathcal{C}$, and the vertices in X that are not adjacent to any component in $\mathcal{C}$. Let M be a maximum matching in $G_B - N_{G_B}[X_2]$ that saturates X_1. The partition $X_1 \dot\cup X_2$ of X fulfills the following properties:

- Let $\mathcal{C}_2$ be the set of connected components P in $\mathcal{C}$ where s_P is a vertex in $N_{G_B}(X_2)$, i.e., $\mathcal{C}_2 = \{P = w_1w_2w_3w_4w_5 \in \mathcal{C} \mid N_G(\{w_1, w_2, w_4, w_5\}) \cap X_2 \neq \emptyset\}$. It holds either that $\mathcal{C}_2$ is the empty set (when $X_2 = \emptyset$) or that it contains less than $|X_2|$ connected components of $\mathcal{C}$, i.e., $|\mathcal{C}_2| < |X_2|$ (when $Y = X_2 \neq \emptyset$).

[3] It certainly does completely settle the question for modulators to $\mathcal{H}$-component graphs for all *hereditary* classes $\mathcal{H}$. If $\mathcal{H}$ contains any connected graph with at least three vertices then we get a lower bound; else all connected components have one or two vertices and there is a polynomial kernelization.

- For every vertex $x \in X_1$, let $P_x = w_1^x w_2^x w_3^x w_4^x w_5^x$ be the connected component in $\mathcal{C}_1 := \mathcal{C} \setminus \mathcal{C}_2$ that is paired to x by M, i.e., $\{x, s_{P_x}\} \in M$. It holds that there exists a vertex $w^x \in \{w_1^x, w_2^x, w_4^x, w_5^x\}$ such that $\{w^x, x\} \in E(G)$ (definition of G_B). Note that $\mathcal{C}_1$ also contains all connected components that are not adjacent to any vertex in X or where only the middle vertex of a path in $\mathcal{C}$ is adjacent to a vertex in X.

Using the above partition, one can show that there exists an optimum solution S that contains for each path P_x with $x \in X_1$ the locally optimal solution $\{\{x, w^x\}, \{w_3^x, w_2^x\}\}$ resp. $\{\{x, w^x\}, \{w_3^x, w_4^x\}\}$ depending on whether $w^x \in \{w_4^x, w_5^x\}$ or $w^x \in \{w_1^x, w_2^x\}$. More generally, for every vertex w of a path $P \in \mathcal{C}$, except the middle vertex, and every vertex $x \in X$ that is adjacent to w there exists a local optimum solution to P that uses edge $\{w, x\}$ and has the middle vertex of P as an endpoint of the second solution edge. This is the crucial difference to a path $P' = v_1 v_2 v_3$ of length two. Here, the only locally optimal solution that dominates P' and contains an edge between P' and X is $\{\{v_2, x\}\}$ with $x \in X$, but this local solution does not contain the vertices v_1 and v_3. We used this in our lower bound construction to control which P_3's may be used to "buy" vertices in X.

▶ **Reduction Rule 1.** Delete X_1 from G, i.e., let $G' = G - X_1$, $X' = X \setminus X_1 = X_2$, and $k' = k$.

▷ Claim 8. Reduction Rule 1 is safe.

Proof. Let F be an edge dominating set of size at most k in G. We construct an edge dominating set F' of size at most $k' = k$ in G' by deleting every edge $e = \{x, y\} \in F$ if both endpoints of e are contained in X_1, or if exactly one endpoint is contained in X_1 and the other endpoint is isolated in G'; and by replacing every edge $e = \{x, y\} \in F$ with $x \in X_1$ and $y \notin X_1$ by exactly one edge in $\delta_{G'}(y)$ if $\delta_{G'}(y) \neq \emptyset$. It holds that F' has size at most $k = k'$ because we either delete edges in F or replace them one for one by a new edge. Since every vertex in $V(G') \cap V(F)$ is either contained in $V(F')$ or isolated in G' it holds that F' is an edge dominating set in G'.

For the other direction, let F' be an edge dominating set of size at most k' in G'. Consider the path $P_x = w_1^x w_2^x w_3^x w_4^x w_5^x$ for some vertex $x \in X_1$. It holds that the only vertex in P_x that can be adjacent to a vertex in $X' = X \setminus X_1 = X_2$ is vertex w_3^x; otherwise P_x would be a component in $\mathcal{C}_2$ and not in $\mathcal{C}_1$ (by definition of $\mathcal{C}_1$ and $\mathcal{C}_2$). Furthermore, the edge dominating set F' must dominate the two non-adjacent edges $\{w_1^x, w_2^x\}$ and $\{w_4^x, w_5^x\}$. Since w_1^x, w_2^x, w_4^x, and w_5^x are only adjacent to vertices in P_x the set F' must contain one of the two edges $e_{1,2}^x = \{w_1^x, w_2^x\}$, $e_{2,3}^x = \{w_2^x, w_3^x\}$ and one of the two edges $e_{3,4}^x = \{w_3^x, w_4^x\}$, $e_{4,5}^x = \{w_4^x, w_5^x\}$. To obtain an edge dominating set of size at most k in G we replace for each vertex $x \in X_1$ these edges with the local optimum solution $\{\{x, w^x\}, \{w_3^x, w_2^x\}\}$ resp. $\{\{x, w^x\}, \{w_3^x, w_4^x\}\}$ depending whether $w^x \in \{w_4^x, w_5^x\}$ or $w^x \in \{w_1^x, w_2^x\}$. It holds that $|F| \leq |F'|$ because for every vertex $x \in X_1$ we replace the at least two edges in $F' \cap \{e_{1,2}^x, e_{2,3}^x, e_{3,4}^x, e_{4,5}^x\}$ by the two edges of the locally optimal solution $\{\{x, w^x\}, \{w_3^x, w_2^x\}\}$ resp. $\{\{x, w^x\}, \{w_3^x, w_4^x\}\}$.

It remains to show that F is indeed an edge dominating set in G. The set $V(F)$ contains all vertices in $V(F')$, except some vertices in the connected components P_x with $x \in X_1$ where we change the edge dominating set F'. Furthermore, $V(F)$ contains all vertices in X_1 because for every vertex $x \in X_1$ the edge $\{w^x, x\}$ is contained in F. Thus, the only edges that are possibly not dominated by F have one endpoint in a path P_x with $x \in X_1$. Since w_3^x is contained in $V(F)$ (by construction), since every edge in P_x is dominated by F (by construction), and since the vertices in $\{w_1^x, w_2^x, w_4^x, w_5^x\}$ are only adjacent to vertices in $P_x \cup X_1$, it follows that F is an edge dominating set in G. ◁

After applying Reduction Rule 1 it holds that for each path $P = w_1w_2w_3w_4w_5 \in \mathcal{C}_1$ only the vertex w_3 can be adjacent to a vertex in X, and we can assume that every (optimum) solution contains the edges $\{w_2, w_3\}$ and $\{w_3, w_4\}$. Additionally, one can show that there exists an optimum solution that does not contain any edge between $\mathcal{C}_1$ and X because we can replace any such edge $e = \{x, v\}$ with $v \in V(\mathcal{C}_1)$ by the edge $\{x, u\}$ with $u \in N_G(x) \setminus V(\mathcal{C}_1)$ (or delete this edge when $N_G(x) \setminus V(\mathcal{C}_1) = \emptyset$). This allows us to delete $\mathcal{C}_1$ from G.

▶ **Reduction Rule 2.** Delete all connected components in $\mathcal{C}_1$ and decrease k by the size of a minimum edge dominating set in $\mathcal{C}_1$, i.e., let $G' = G - \mathcal{C}_1$, $X' = X$, and $k' = k - \text{EDS}(\mathcal{C}_1)$.

▷ Claim 9. Reduction Rule 2 is safe.

Proof. First, we will show that there exists an edge dominating set F of size at most k in G such that no edge in F has one endpoint in a connected component of $\mathcal{C}_1$ and the other endpoint in X. Let F be an edge dominating set of size at most k in G with $F \cap E(\mathcal{C}_1, X)$ minimal, and let $P = w_1w_2w_3w_4w_5$ be a path in $\mathcal{C}_1$. We can assume, w.l.o.g., that F contains the edges $\{w_2, w_3\}$ and $\{w_3, w_4\}$ because F must dominate the non-adjacent edges $\{w_1, w_2\}$, $\{w_4, w_5\}$, and the vertices w_1, w_2, w_4, w_5 are only adjacent to vertices in P; otherwise, P is contained in $\mathcal{C}_2$ and not $\mathcal{C}_1$. Now, assume for contradiction that there exists an edge $e = \{x, y\} \in F \cap E(\mathcal{C}_1, C)$ with $x \in X$ and $y \in P$ where $P = w_1w_2w_3w_4w_5$ is a path in $\mathcal{C}_1$. It holds that $y = w_3$ because w_3 is the only vertex in P that is adjacent to a vertex in X. If every vertex $u \in N_G(x)$ is contained in $V(F)$ then let $\widetilde{F} = F \setminus \{e\}$. Otherwise, let $\widetilde{F} = F \setminus \{e\} \cup \{\{x, u\}\}$, where $u \in N_G(x) \setminus V(F)$. It holds that $\widetilde{F}$ is an edge dominating set in G because $y = w_3$ is still a vertex in $V(\widetilde{F})$ which implies $V(F) \subseteq V(\widetilde{F})$. Furthermore, u is not contained in a connected component of $\mathcal{C}_1$ because for every path $P = w_1w_2w_3w_4w_5$ in $\mathcal{C}_1$ the vertex w_3 is contained in $V(F)$ and no other vertex is adjacent to a vertex in X. Now, the set $\widetilde{F}$ is an edge dominating set of size at most k in G with $\widetilde{F} \cap E(\mathcal{C}_1, X) \subsetneq F \cap E(\mathcal{C}_1, X)$ which contradicts the minimality of $F \cap E(\mathcal{C}_1, X)$ and proves that there exists an edge dominating set F of size at most k in G with $F \cap E(\mathcal{C}_1, X) = \emptyset$. This implies that $F' = F \setminus E(\mathcal{C}_1)$ is an edge dominating set of size at most k' in G' when F is a solution to (G, k, X) with $F \cap E(\mathcal{C}_1, X) = \emptyset$.

For the other direction, let F' be an edge dominating set of size at most k' in G'. To obtain an edge dominating set F of size at most k in G we add for every path $P = w_1w_2w_3w_4w_5$ in $\mathcal{C}_1$ the two edges $\{w_2, w_3\}$ and $\{w_3, w_4\}$, which are a minimum edge dominating set of P, to F'. It follows that F has size $|F'| + \text{EDS}(\mathcal{C}_1) \leq k$. The set F dominates all edges in $G - X$ as well as all edges between $\mathcal{C}_2$ and X because $F' \subseteq F$, and because F contains an edge dominating set of $\mathcal{C}_1$. Additionally, F dominates all edges between $\mathcal{C}_1$ and X because F dominates all middle vertices of the paths in $\mathcal{C}_1$ which are the only vertices in $\mathcal{C}_1$ that are adjacent to X. Hence, F is an edge dominating set of size at most k in G. ◁

Let (G', k', X') be the reduced instance. It holds that the set of connected components in $G' - X'$ is $\mathcal{C}_2$ because we delete all other connected components during Reduction Rule 2. Since $|\mathcal{C}_2| \leq |X_2| = |X'|$ it follows that G' has at most $5 \cdot |\mathcal{C}_2| + |X'| \leq 6|X'|$ vertices. It remains to show that we can perform the reduction in polynomial time. We apply each Reduction Rule at most once. Furthermore, we can apply the Reduction Rules in polynomial time because we can compute the partition of X as well as the sets $\mathcal{C}_1$ and $\mathcal{C}_2$ in polynomial time, and because we can delete sets of vertices from G and X in polynomial time. ◀

While this is not the full story about the classification in the following section, it hopefully shows the spirit of how upper and lower bounds for kernelization can arise. Solution edges between components of $G - X$ and X play a crucial role and they affect the solutions for components in nontrivial ways, e.g., apart from control opportunities, it depends on how much budget is needed for $H - B$ when edges between B and X are in the solution.

4 EDS parameterized by the size of a modulator to an $\mathcal{H}$-component graph

In this section, we develop a complete classification of EDGE DOMINATING SET parameterized by the size of a modulator to an $\mathcal{H}$-component graph regarding existence of polynomial kernelizations for all finite sets $\mathcal{H}$. This is motivated by the observed difference between modulating to P_3-component graphs (no polynomial kernelization unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$) vs. modulating to P_5-component graphs (polynomial kernelization). To this end, we will study which properties graphs $H \in \mathcal{H}$ must have, such that EDGE DOMINATING SET parameterized by the size of a modulator to an $\mathcal{H}$-component graph has resp. does not have a polynomial kernel. To recall, the input of our problem is a tuple (G, k, X) where $G - X$ is an $\mathcal{H}$-component graph and we ask whether G has an edge dominating set of size at most k; the parameter is $|X|$.

In contrast to VERTEX COVER, where we can delete a vertex in the modulator if we know that this vertex must be in a solution of certain size, this is not the case for EDGE DOMINATING SET because we do not necessarily know which incident edge should be chosen. Of course, we can check for a vertex x in the modulator X how not having this vertex as an endpoint of a solution edge influences the size of a minimum edge dominating set of $G - X$. But, even if we find out that a vertex x in the modulator X must be an endpoint of a solution edge, we do not know if the other endpoint of the solution edge incident with x is in X or in a connected component of $G - X$. If there would be a connected component C in $G - X$ with the property that there exists a vertex $v \in N(x) \cap V(C)$ with $\text{EDS}(C) = \text{EDS}(C - v) + 1$, then it could be possible to have x as an endpoint of a solution edge without paying more than the cost of a minimum edge dominating set in C. Thus, instead of finding vertices in the modulator that must be endpoint of a solution edge, we want to find vertices in the modulator that can be endpoints of a solution edge without spending more budget than the size of a minimum edge dominating set in $G - X$. Similarly, getting edges to r vertices in X while increasing the cost in C by less than r is of interest (cost equal to r can always be had). The following definition classifies relevant vertices and vertex sets in a graph H, which may occur as a component of $G - X$.

▶ **Definition 10.** *Let $H = (V, E)$ be a connected graph.*

- *We call a vertex $v \in V$* extendable *if $\text{EDS}(H - v) + 1 = \text{EDS}(H)$. We denote the set of extendable vertices of H by $Q(H)$.* (Intuitively, these vertices allow a local solution for an H-component in $G - X$ that includes an edge $\{v, x\}$ with $x \in X$ and $v \in V(H)$.)
- *We call a set $Y \subseteq Q(H)$* free *if for all vertices $v \in Y$ and for all minimum edge dominating sets F in H there exists a minimum edge dominating set F' in $H - v$ of size $|F| - 1$ and with $V(F) \setminus Y \subseteq V(F')$. By $W(H)$ we denote the unique maximum free set of H. We call a vertex $w \in W(H)$* free.[4] (Intuitively, vertices in Y can be used for solution edges between components and X, while covering the same vertices of $H - Y$ as any local optimum solution; thus, they cannot be used for lower bounds like for P_3-components.)
- *We call a vertex $v \in V$* uncovered *if no minimum edge dominating set F of H contains an edge incident with v, i.e. $v \notin V(F)$. We denote the set of uncovered vertices by $U(H)$.* (Intuitively, H-components with any $v \in U(H)$ adjacent to $x \in X$ are easy to handle because $x \notin V(F)$ would imply that the local cost for H increases above $\text{EDS}(H)$.)

[4] We show in the full version that $W(H)$ is unique.

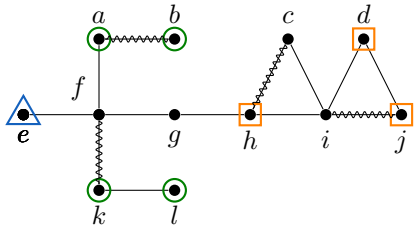


Figure 2 Example of an H-component with $\text{EDS}(H) = 4$. The wavy edges are a possible minimum edge dominating set of H.

- *For any* $Y \subseteq V$ *define* $\text{cost}(Y) := |Y| + \text{EDS}(H - Y) - \text{EDS}(H)$.
  (Intuitively, $\text{cost}(Y)$ is equal to the additional budget that is needed for an H-component of $G - X$ when exactly the vertices in Y have solution edges to X. Note that $\text{cost}(\{v\}) = 0$ for all extendable vertices v.)
- *We call a set* $B \subseteq V \setminus W(H)$ beneficial *if for all* $\widetilde{B} \subsetneq B$ *we have* $|B| - \text{cost}(B) > |\widetilde{B}| - \text{cost}(\widetilde{B})$ *or, equivalently,* $\text{EDS}(H - B) < \text{EDS}(H - \widetilde{B})$. *Note that this must also hold for* $\widetilde{B} = \emptyset$ *which implies that for all beneficial sets we have* $|B| - \text{cost}(B) > 0$ *or, equivalently,* $\text{EDS}(H - B) < \text{EDS}(H)$.
  (Intuitively, the solution may include $|B|$ edges between B and some $X' \subseteq X$ while increasing the cost for the H-component by exactly $\text{cost}(B)$; this saves $|B| - \text{cost}(B) > 0$ over taking any $|B|$ edges incident with X'. The condition for all $\widetilde{B} \subsetneq B$ ensures that the savings of getting $|B|$ edges at cost $\text{cost}(B)$ is greater than for any proper subset.)
- *We call a beneficial set* B strongly beneficial *if* $\text{cost}(B) < \sum_{i=1}^{h} \text{cost}(B_i)$ *holds for all covers* $B_1, B_2, \ldots, B_h \subsetneq B$ *of* B. (Intuitively, for a strongly beneficial set B we cannot get the same number of edges to X by using sets B_i in several different H-components.)

▶ **Example 11** (Illustration of Definition 10)**.** Figure 2 shows a connected graph H. The size of an edge dominating set in H is at least four because a solution has to dominate the four pairwise non-adjacent edges $\{a, b\}, \{k, l\}, \{j, d\}$ and $\{g, h\}$. Thus, $\text{EDS}(H) = 4$ because the wavy edges are an edge dominating set of H.

The vertices $\{a, b, k, l\}$, marked with a green cycle, as well as the vertices $\{d, h, j\}$, marked with an orange rectangle, are extendable. But only the green marked vertices $\{a, b, k, l\}$ are free: Let F be any minimum edge dominating set in H. The set F must contain exactly one of the two edges $e_1 = \{a, b\}$ and $e_2 = \{a, f\}$, and exactly one of the two edges $e_3 = \{k, l\}$ and $e_4 = \{k, f\}$. Now, $F' = F \setminus \{e_1, e_2, e_3, e_4\} \cup \{f, k\}$ is an edge dominating set in $H - a$ and $H - b$ of size $|F| - 1$, and $F' = F \setminus \{e_1, e_2, e_3, e_4\} \cup \{a, f\}$ is an edge dominating set in $H - k$ and $H - l$ of size $|F| - 1$ which implies that the vertices $\{a, b, k, l\}$ are free. The vertices $\{d, h, j\}$ are not free because no minimum edge dominating set F' in $H - d$, resp. $H - h$, resp. $H - j$ has vertex c, which is not extendable, as an endpoint of a solution edge, but the graph H has a minimum edge dominating set that has c as an endpoint, namely the one containing the wavy edges $\{a, b\}, \{h, c\}, \{d, j\}$. The vertex e, marked with a blue triangle, is uncovered.

The set $\{c, g\}$ is strongly beneficial, whereas the set $\{c, g, i, j\}$ is only beneficial, but not strongly beneficial: The set $\{c, g\}$ is beneficial because $\text{EDS}(H - \{c, g\}) = 3$ and $\text{EDS}(H - c) = \text{EDS}(H - g) = \text{EDS}(H) = 4$, and strongly beneficial because the only possible non-trivial cover of $\{c, g\}$ is $\{c\}, \{g\}$ and $\text{cost}(\{c, g\}) = 1 < 2 = \text{cost}(\{c\}) + \text{cost}(\{g\})$. The set $\{c, g, i, j\}$ is beneficial because $\text{EDS}(H - \{c, g, i, j\}) = 2$ and $\text{EDS}(H - B) \geq 3$ for all $B \subsetneq \{c, g, i, j\}$. But $\{c, g, i, j\}$ is not strongly beneficial because $\text{cost}(\{c, g, i, j\}) = 2 = 1 + 1 + 0 = \text{cost}(\{c, g\}) + \text{cost}(\{i\}) + \text{cost}(\{j\})$. Observe that the set $\{c, g, i\}$ is not beneficial even though

$\text{EDS}(H - \{c, g, i\}) = 3 < 4 = \text{EDS}(H)$, because $\{c, g\} \subsetneq \{c, g, i\}$ and $\text{EDS}(H - \{c, g, i\}) = 3 = \text{EDS}(H - \{c, g\})$.

We are now able to give a more detailed version of Theorem 1, which specifies for each finite set $\mathcal{H}$ of connected graphs the kernelization complexity of EDGE DOMINATING SET parameterized by the size of a modulator to $\mathcal{H}$-component graphs.

▶ **Theorem 12.** *Let $\mathcal{H}$ be any finite set of connected graphs. The* EDGE DOMINATING SET *problem parameterized by the size of a modulator to $\mathcal{H}$-component graphs behaves as follows:*

1. *If $\mathcal{H}$ contains any graph H fulfilling one of the following items then there is no polynomial kernelization unless* NP ⊆ coNP/poly*:*
   - **a.** *There is an extendable vertex in H that is not free, i.e., $Q(H) \setminus W(H) \neq \emptyset$.*
   - **b.** *There is a strongly beneficial set B in H that contains an uncovered vertex, i.e., $B \cap U(H) \neq \emptyset$.*
   - **c.** *There is a vertex in H that is neither uncovered, free, nor neighbor of a free vertex, i.e., $V(H) \setminus (N[W(H)] \cup U(H)) \neq \emptyset$.*
   - **d.** *There is a strongly beneficial set $B \subseteq N(W(H))$ in H such that no minimum edge dominating set F_B of $H - B$ covers all vertices of $N(W(H)) \setminus B$.*
2. *Else, if $\mathcal{H}$ contains at least one graph that has a strongly beneficial set, then there is a kernelization to $\mathcal{O}(|X|^d)$ vertices, $\mathcal{O}(|X|^{d+1})$ edges, and size $\mathcal{O}(|X|^{d+1} \log |X|)$, and there is no kernelization to size $\mathcal{O}(|X|^{d-\varepsilon})$, for any $\varepsilon > 0$, unless* NP ⊆ coNP/poly *where d is the size of the largest strongly beneficial set in any $H \in \mathcal{H}$.*
3. *Else, there is a kernelization to $\mathcal{O}(|X|^2)$ vertices, $\mathcal{O}(|X|^3)$ edges, and size $\mathcal{O}(|X|^3 \log |X|)$, and there is no kernelization to size $\mathcal{O}(|X|^{2-\varepsilon})$, for any $\varepsilon > 0$, unless* NP ⊆ coNP/poly*.*

Observe, Theorem 1 directly follows from Theorem 12 because disconnected graphs in $\mathcal{H}$ do not affect the resulting class of $\mathcal{H}$-component graphs, i.e., given *any* finite set $\mathcal{H}$ of graphs we can take the subset $\mathcal{H}'$ of connected graphs in $\mathcal{H}$ and apply Theorem 12 to $\mathcal{H}'$. As an example for applying the theorem, for $\mathcal{H} = \{P_3\}$ we get Item 1a, for $\mathcal{H} = \{P_4\}$ we get Item 1b, for $\mathcal{H} = \{K_3\}$ and $\mathcal{H} = \{K_5\}$ we get Item 1c, and for $\mathcal{H} = \{P_2\} = \{K_2\}$, $\mathcal{H} = \{K_4\}$, $\mathcal{H} = \{P_5\}$, as well as $\mathcal{H} = \{E = \Xi\}$ we get Item 3.

▶ Remark. We showed that EDGE DOMINATING SET parameterized by the size of a given modulator X to a P_5-component graph admits a kernelization with $\mathcal{O}(|X|)$ vertices (see Theorem 7). The reason why we the kernelization procedure of Item 3 only reduces to $\mathcal{O}(|X|^2)$ vertices instead of $\mathcal{O}(|X|)$ vertices is that H-components can have uncovered vertices. This leads to a different marking argument similar to the case for EDGE DOMINATING SET parameterized by solution size. Note that EDS parameterized by solution size is covered by Item 3.

Proof outline for Theorem 12. We begin by establishing a number of useful properties of the terms introduced in Definition 10, e.g., that each graph H containing a beneficial set B also contains a strongly beneficial set $B' \subseteq B$ (Proposition 13 (11)).

The kernelization lower bound of Item 1 is proved by generalizing the lower bound obtained for P_3-component graphs in Theorem 5. We define so-called *control pairs* by abstracting properties of P_3-components used in the proof and show that there is no polynomial kernelization when any graph $H \in \mathcal{H}$ has a control pair (Theorem 15). For a connected graph $H = (V, E)$ the pair (C, B) is called a *control pair*, if $B \subseteq V$ is strongly beneficial, if $C \subseteq V \setminus (Q(H) \cup B)$ and no vertex $c \in C$ is extendable in $H - B$, if there exists a minimum edge dominating set F in H such that $C \subseteq V(F)$, and if for all minimum edge dominating

sets F_B in $H - B$ it holds that $C \not\subseteq V(F_B)$. Observe that for $H = P_3 = v_1v_2v_3$ the set B is the vertex v_2 and the set C is the vertex v_1 (or v_2). Afterwards, we show that graphs H fulfilling Items 1a, 1b, 1c, or 1d have control pairs (Lemmas 17, 18, 19, and 20).

In Item 1d, and in the items below, we (may) use that no graph in $\mathcal{H}$ fulfills Items 1a, 1b, or 1c. Accordingly, each graph $H \in \mathcal{H}$ has $V(H) = N[W(H)] \cup U(H)$, i.e., each vertex of H is uncovered, free, or neighbor of a free vertex. Moreover, every extendable vertex is also free, i.e., $Q(H) = W(H)$, and strongly beneficial sets contain no (uncovered) vertices of $U(H)$. This implies that all strongly beneficial sets are subsets of $N(W(H))$, the neighborhood of the free vertices, as neither uncovered nor free vertices can be contained and no further vertices except those in $N(W(H))$ exist in H (in this case).

For Item 2 we have that no graph in $\mathcal{H}$ fulfills any of the Items 1a through 1d and that at least one graph in $\mathcal{H}$ has a strongly beneficial set. Thus, in addition to the above restrictions on $H \in \mathcal{H}$, we know that for each strongly beneficial set B, which here must be a subset of $N(W(H))$, there is a minimum edge dominating set F_B of $H - B$ that covers all vertices in $N(W(H)) \setminus B$. We give a general kernelization procedure that reduces the number of components in $G - X$ to $\mathcal{O}(|X|^d)$ where d is the size of the largest strongly beneficial set among graphs $H \in \mathcal{H}$ (Lemma 27). We then rule out kernelizations of size $\mathcal{O}(|X|^{d-\varepsilon})$ using only H-components, where H is any graph in $\mathcal{H}$ that exhibits the largest size d of strongly beneficial sets (Lemma 31). Note that in the present item d is always at least two because having a strongly beneficial set B of size one would mean that $v \in B$ is an extendable vertex that is not free (because beneficial sets are disjoint from the set $W(H)$ of free vertices), which is handled by Item 1a.

Finally, for Item 3, it remains to consider the case that no graph $H \in \mathcal{H}$ fulfills any of the Items 1a through 1d and that no graph in H has a strongly beneficial set. It follows that no graph in H has any beneficial sets (Proposition 13 (11)) and, as before, we have $V(H) = N[W(H)] \cup U(H)$. We obtain a kernelization to $\mathcal{O}(|X|^2)$ vertices, $\mathcal{O}(|X|^3)$ edges, and size $\mathcal{O}(|X|^3 \log |X|)$ (Lemma 23). The lower bound ruling out kernelizations of size $\mathcal{O}(|X|^{2-\varepsilon})$ for any $\varepsilon > 0$, and in fact for any set $\mathcal{H}$, follows easily by a simple reduction from VERTEX COVER for which a lower bound ruling out size $\mathcal{O}(n^{2-\varepsilon})$ is known [6] (Lemma 35). ◀

5 Conclusion

As our main result, we have given a complete classification for EDGE DOMINATING SET parameterized by the size of a modulator to $\mathcal{H}$-component graphs for all finite sets $\mathcal{H}$. An obvious follow-up question is to extend this result to infinite sets $\mathcal{H}$. Our lower bounds of course continue to work in this setting, and the upper bounds still permit us to reduce the number of connected components (under the same conditions as before, e.g., that relevant beneficial sets have bounded size). However, for infinite $\mathcal{H}$, polynomial kernels also require us to shrink connected components of $G - X$, and to derive general rules for this. Moreover, even determining beneficial sets etc. for graphs $H \in \mathcal{H}$ could no longer be dismissed as being constant time. It is conceivable that such a classification is doable whenever graphs in $\mathcal{H}$ have bounded treewidth, as this simplifies the required additional steps. Since most known tractable graph classes for EDGE DOMINATING SET have bounded treewidth (and tractability for $G - X$ is required, or else NP-hardness for $|X| = 0$ rules out kernels and fixed-parameter tractability), this seems like a reasonable goal. Apart from this, it would be nice to close the gap between size $\mathcal{O}(|X|^{d+1} \log |X|)$ and the lower bound of $\mathcal{O}(|X|^{d-\varepsilon})$.

References

1 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.

2 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization Lower Bounds by Cross-Composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

3 Jean Cardinal, Stefan Langerman, and Eythan Levy. Improved approximation bounds for edge dominating set in dense graphs. *Theor. Comput. Sci.*, 410(8-10):949–957, 2009. doi:10.1016/j.tcs.2008.12.036.

4 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of edge dominating set problems. *J. Comb. Optim.*, 11(3):279–290, 2006. doi:10.1007/s10878-006-7908-0.

5 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *TOCT*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.

6 Holger Dell and Dieter van Melkebeek. Satisfiability Allows No Nontrivial Sparsification unless the Polynomial-Time Hierarchy Collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 Bruno Escoffier, Jérôme Monnot, Vangelis Th. Paschos, and Mingyu Xiao. New Results on Polynomial Inapproximabilityand Fixed Parameter Approximability of Edge Dominating Set. *Theory Comput. Syst.*, 56(2):330–346, 2015. doi:10.1007/s00224-014-9549-5.

9 Henning Fernau. Edge dominating set: Efficient Enumeration-Based Exact Algorithms. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2006. doi:10.1007/11847250_13.

10 Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On Two Techniques of Combining Branching and Treewidth. *Algorithmica*, 54(2):181–207, 2009. doi:10.1007/s00453-007-9133-3.

11 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

12 Toshihiro Fujito and Hiroshi Nagamochi. A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Applied Mathematics*, 118(3):199–207, 2002. doi:10.1016/S0166-218X(00)00383-8.

13 Shivam Garg and Geevarghese Philip. Raising The Bar For Vertex Cover: Fixed-parameter Tractability Above a Higher Guarantee. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1152–1166. SIAM, 2016. doi:10.1137/1.9781611974331.ch80.

14 Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An Incremental Polynomial Time Algorithm to Enumerate All Minimal Edge Dominating Sets. *Algorithmica*, 72(3):836–859, 2015. doi:10.1007/s00453-014-9875-7.

15 Torben Hagerup. Kernels for Edge Dominating Set: Simpler or Smaller. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 491–502. Springer, 2012. doi:10.1007/978-3-642-32589-2_44.

16 Eva-Maria C. Hols and Stefan Kratsch. On Kernelization for Edge Dominating Set under Structural Parameters. *CoRR*, abs/1901.03582, 2019. arXiv:1901.03582.

17 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

18 Ken Iwaide and Hiroshi Nagamochi. An Improved Algorithm for Parameterized Edge Dominating Set Problem. *J. Graph Algorithms Appl.*, 20(1):23–58, 2016. doi:10.7155/jgaa.00383.

19 Bart M. P. Jansen and Hans L. Bodlaender. Vertex Cover Kernelization Revisited - Upper and Lower Bounds for a Refined Parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

20 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the Neighbourhood Helly of Some Graph Classes and Applications to the Enumeration of Minimal Dominating Sets. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, volume 7676 of *Lecture Notes in Computer Science*, pages 289–298. Springer, 2012. doi:10.1007/978-3-642-35261-4_32.

21 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. Polynomial Delay Algorithm for Listing Minimal Edge Dominating Sets in Graphs. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 2015. doi:10.1007/978-3-319-21840-3_37.

22 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3):197–221, 2003. doi:10.1016/S0166-218X(02)00198-1.

23 Stefan Kratsch. A Randomized Polynomial Kernelization for Vertex Cover with a Smaller Parameter. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.59.

24 Stefan Kratsch and Magnus Wahlström. Representative Sets and Irrelevant Vertices: New Tools for Kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

25 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster Parameterized Algorithms Using Linear Programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

26 Elena Prieto. *Systematic kernelization in FPT algorithm design*. PhD thesis, The University of Newcastle, Australia, 2005.

27 Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Paths, Flowers and Vertex Cover. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 382–393. Springer, 2011. doi:10.1007/978-3-642-23719-5_33.

28 Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. Efficient Exact Algorithms through Enumerating Maximal Independent Sets and Other Techniques. *Theory Comput. Syst.*, 41(3):563–587, 2007. doi:10.1007/s00224-007-1334-2.

29 Richard Schmied and Claus Viehmann. Approximating edge dominating set in dense graphs. *Theor. Comput. Sci.*, 414(1):92–99, 2012. doi:10.1016/j.tcs.2011.10.001.

30 Johan M. M. van Rooij and Hans L. Bodlaender. Exact Algorithms for Edge Domination. *Algorithmica*, 64(4):535–563, 2012. doi:10.1007/s00453-011-9546-x.

31 Jianxin Wang, Beiwei Chen, Qilong Feng, and Jianer Chen. An Efficient Fixed-Parameter Enumeration Algorithm for Weighted Edge Dominating Set. In Xiaotie Deng, John E. Hopcroft, and Jinyun Xue, editors, *Frontiers in Algorithmics, Third International Workshop, FAW 2009, Hefei, China, June 20-23, 2009. Proceedings*, volume 5598 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2009. doi:10.1007/978-3-642-02270-8_25.

32 Mingyu Xiao. Exact and Parameterized Algorithms for Edge Dominating Set in 3-Degree Graphs. In Weili Wu and Ovidiu Daescu, editors, *Combinatorial Optimization and Applications - 4th International Conference, COCOA 2010, Kailua-Kona, HI, USA, December 18-20, 2010, Proceedings, Part II*, volume 6509 of *Lecture Notes in Computer Science*, pages 387–400. Springer, 2010. doi:10.1007/978-3-642-17461-2_31.

33 Mingyu Xiao, Ton Kloks, and Sheung-Hung Poon. New parameterized algorithms for the edge dominating set problem. *Theor. Comput. Sci.*, 511:147–158, 2013. doi:10.1016/j.tcs.2012.06.022.

34 Mingyu Xiao and Hiroshi Nagamochi. Parameterized edge dominating set in graphs with degree bounded by 3. *Theor. Comput. Sci.*, 508:2–15, 2013. doi:10.1016/j.tcs.2012.08.015.

35 Mingyu Xiao and Hiroshi Nagamochi. A refined exact algorithm for Edge Dominating Set. *Theor. Comput. Sci.*, 560:207–216, 2014. doi:10.1016/j.tcs.2014.07.019.

36 Mihalis Yannakakis and Fanica Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

Compressed Decision Problems in Hyperbolic Groups

Derek Holt
University of Warwick, UK
D.F.Holt@warwick.ac.uk

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Saul Schleimer
University of Warwick, UK
s.schleimer@warwick.ac.uk

Abstract

We prove that the compressed word problem and the compressed simultaneous conjugacy problem are solvable in polynomial time in hyperbolic groups. In such problems, group elements are input as words defined by straight-line programs defined over a finite generating set for the group. We prove also that, for any infinite hyperbolic group G, the compressed knapsack problem in G is NP-complete.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases hyperbolic groups, algorithms for compressed words, circuit evaluation problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.37

Related Version A full version can be found `https://arxiv.org/abs/1808.06886`.

Funding *Markus Lohrey*: has been supported by the DFG research project LO 748/12-1.

1 Introduction

Compression techniques in group theory have attracted attention in recent years [9, 10, 30, 36, 37]. Often, algorithms for classical group theoretic problems, such as the word problem or the conjugacy problem, face the problem that huge intermediate words arise during the computation. In some situations, these words are highly compressible; one can then attempt to compute on succinct representatives instead of on the words themselves.

Straight-line programs (SLPs) are a widely-used compression technique for words. An SLP can be seen as a context-free grammar $\mathcal{G}$ that produces a single word denoted $\mathsf{val}(\mathcal{G})$; see Section 4 for a precise definition. The *size* of $\mathcal{G}$ can be defined as the sum of the lengths of the right-hand sides of the productions of $\mathcal{G}$. In fact, the length of $\mathsf{val}(\mathcal{G})$ can be exponential in the size of $\mathcal{G}$, showing that non-trivial compression is possible for SLPs. There are numerous papers in computer science that study the complexity of decision problems for words that are succinctly represented by SLPs; see [29] for a survey. Applications of SLPs in group theory can be traced back to Babai and Szemeredi's reachability theorem for finite groups [2].

In this paper we deal with the so-called compressed word problem for a finitely generated group G, which we define as follows. Suppose that Σ is a finite generating set for G. We assume that Σ is *symmetric*: if a lies in Σ then so does a^{-1}. The *word problem* for G asks whether a given word $w \in \Sigma^*$ represents the group identity of G. This is one of the

fundamental decision problems in group theory as set out by Dehn [8] in 1911. The *compressed word problem* for G is the same problem except that the input word w is represented by an SLP. We also call such inputs *compressed words*.

Clearly, the compressed word problem for a group G is decidable if and only if the word problem for G is decidable. It also seems obvious that the computational complexity of the compressed word problem for G should be more difficult than the word problem itself. This is indeed the case if $\mathsf{P} \neq \mathsf{NP}$; see the discussion on the next page. It is also interesting to note that the compressed word problem for a group G is exactly the *circuit evaluation problem* for G: here the input is a *circuit* (a directed acyclic graph whose nodes are called gates) where input gates are labelled by generators of G and where internal gates compute the product of their inputs. We then ask if a distinguished output gate evaluates to the identity of G. For finite groups, the complexity of the circuit evaluation problem (and hence, the compressed word problem) was clarified in [4]: if G is a finite solvable group, then the compressed word problem for G belongs to the parallel complexity class $\mathsf{DET} \subseteq \mathsf{NC}^2$. Futher, if G is a finite non-solvable group, then the compressed word problem for G is P-complete. This dichotomy naturally motivates the investigation of compressed word problems for general finitely generated groups.

The compressed word problem also has applications for the ordinary (uncompressed) word problem. From techniques very similar to those used in the proof of [40, Theorem 5.2] we may deduce the following: the word problem for a finitely generated subgroup of the automorphism group $\mathsf{Aut}(G)$ is polynomial time reducible to the compressed word problem for G. Similar reductions exist for certain group extensions; see [40, Theorem 4.1] and [30, Theorem 4.8 and 4.9]. This makes groups for which the compressed word problem can be solved in polynomial time interesting. Indeed the class of these groups is quite rich. Let F_n be the free group on n generators. The first result for infinite groups was obtained in [28], where the second author showed that the compressed word problem for F_n is P-complete. This result was used by the third author to show that the word problem for $\mathsf{Aut}(F_n)$ can be solved in polynomial time [40, Theorem 5.2]. This solved an open problem posed by Baumslag [3, Problem (C1)]. Two other important classes of groups in which the compressed word problem can be solved in polynomial time have been found, as follows.

- Virtually special groups; that is, finite extensions of finitely generated subgroups of right-angled Artin groups. Right-angled Artin groups are also known as graph groups or partially commutative groups. Recent work related to three-dimensional topology has shown that the class of virtually special groups is very rich. It contains all Coxeter groups [19], one-relator groups with torsion [43], fully residually free groups [43] (for fully residually free groups, Macdonald [33] independently obtained a polynomial time solution for the compressed word problem), and fundamental groups of hyperbolic 3-manifolds [1].
- Finitely generated nilpotent groups [30]. Here, the compressed word problem even belongs to the parallel complexity class DET [26].

Moreover, for finitely generated linear groups the compressed word problem belongs to the complexity class coRP [30, Theorem 4.15], which implies that there is an efficient randomized polynomial time algorithm that may err with a small probability on negative input instances. On the negative side, it is known that the compressed word problem for every restricted wreath product $G \wr \mathbb{Z}$ with G finitely generated non-abelian is coNP-hard [30, Theorem 4.21]. If G is also finite, then the word problem for $G \wr \mathbb{Z}$ can be easily solved in polynomial time, see also [42]. Assuming $\mathsf{P} \neq \mathsf{NP}$ this gives examples of groups in which the compressed word problem is harder than the word problem. Another interesting result that relates

the compressed word problem to the area of algebraic complexity theory was shown in [30, Theorem 4.16]: The compressed word problem for the linear group $\mathsf{SL}_3(\mathbb{Z})$ is equivalent (up to polynomial time reductions) to polynomial identity testing (that is, the problem whether a circuit over a polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$ evaluates to the zero polyomial).

In this paper, we prove that the compressed word problem can be solved in polynomial time in every hyperbolic group.[1] Hyperbolic groups have a Cayley graph that satisfies a certain hyperbolicity condition, see Section 3 for a precise definition. Hyperbolic groups are of fundamental importance in geometric group theory. In a certain probabilistic sense, almost all finitely presented groups are hyperbolic [16, 38]. Also from a computational viewpoint, hyperbolic groups have nice properties: it is known that the word problem and the conjugacy problem can be solved in linear time [12, 21]. They also have a nice shortlex automatic structure [11]. We show in Theorem 15 that, from a given SLP $\mathcal{G}$ over the generators of a hyperbolic group G, one can compute in polynomial time an SLP for the shortlex normal form of the word $\mathsf{val}(\mathcal{G})$ (this is the length lexicographically smallest word that represents the same group element as $\mathsf{val}(\mathcal{G})$). Since the shortlex normal form for a word w is the empty word if and only if $w =_G 1$ (here, and in the rest of the paper, we write $u =_G v$ if the words u and v represent the same element of the group G), we obtain the following corollary:

▶ **Corollary 1.** *The compressed word problem for a hyperbolic group can be solved in polynomial time.*

A relatively easy consequence of Corollary 1 is that for every hyperbolic group one can compute in polynomial time the order of the group element that is represented by a given SLP (Corollary 16). In Section 6.2 we consider the *compressed conjugacy problem*: the input consists of SLP-compressed words u, v and it is asked whether there exists a word x with $x^{-1}ux =_G v$. We prove that the compressed conjugacy problem for a hyperbolic group can be solved in polynomial time. For this, we show that the algorithm from [12], which solves the conjugacy problem for a hyperbolic group in linear time, can be implemented in polynomial time for SLP-compressed input words. Based on this algorithm, we then generalise our result on compressed conjugacy to the *compressed simultaneous conjugacy problem*, where the input consists of two finite lists $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ of SLP-compressed words over the generators of the group G, and it is asked whether there exists a word x with $x^{-1}u_ix =_G v_i$ for $1 \leq i \leq n$. This problem was shown to be solvable in polynomial time for finitely generated nilpotent groups in [34]. In the uncompressed setting, the simultaneous conjugacy problem was shown to be solvable in linear time for hyperbolic groups in [5]. Again, we show that the algorithm in [5] can be implemented in polynomial time for SLP-compressed input words, which yields the following.

▶ **Theorem 2.** *Let G be a hyperbolic group. Then the compressed simultaneous conjugacy problem for G can be solved in polynomial time. Moreover, if the two input lists are conjugate, then we can compute an SLP for a conjugating element in polynomial time.*

The (ordinary) simultaneous conjugacy problem has also been studied for classes of groups other than hyperbolic groups; see for example [25] and the references therein. The SLP-compressed version is important because the word problem for finitely generated subgroups of the outer automorphism group $\mathsf{Out}(G)$ of G can be reduced to the compressed simultaneous conjugacy problem for G [20, Proposition 10]. Note that in [7] it is shown that for a hyperbolic group G, $\mathsf{Aut}(G)$ and hence $\mathsf{Out}(G)$ are finitely generated. Hence, we get the following corollary from our main results.

[1] This result was announced in [30, Theorem 4.12] without proof.

▶ **Corollary 3.** *For every hyperbolic group G, the word problems for* $\mathsf{Aut}(G)$ *and* $\mathsf{Out}(G)$ *can be solved in polynomial time.*

As a byproduct of our algorithm for the compressed simultaneous conjugacy problem we also show that for every hyperbolic group one can compute in polynomial time from a given finite set S of SLP-compressed group elements a finite generating set for the centraliser of S, where every element of this generating set is represented by an SLP. We call this computation problem the *compressed centraliser problem*.

▶ **Theorem 4.** *Let G be a hyperbolic group. Then the compressed centraliser problem for G can be solved in polynomial time.*

Finally, we consider the compressed knapsack problem for a hyperbolic group. In the (ordinary) knapsack problem for a finitely generated group G the input is a list of words $u_1, \ldots, u_n, v$ over the generators of G, and it is asked whether there exist natural numbers $n_1, \ldots, n_k$ such that $v =_G u_1^{n_1} \cdots u_k^{n_k}$. This problem has been studied in [13, 14, 27, 32, 35] for various classes of groups. In [35] it was shown that the knapsack problem for a hyperbolic group can be solved in polynomial time. Recently, this complexity bound was improved to LogCFL (the closure of the class of context-free languages under logspace-reductions) [31]. Moreover, for every non-elementary hyperbolic group (meaning that the group contains a free non-abelian subgroup), knapsack is LogCFL-complete, whereas for elementary hyperbolic groups knapsack belongs to NL (nondeterministic logspace) [31]. In the compressed knapsack problem, the words $u_1, \ldots, u_n, v$ are represented by SLPs. For the special case $G = \mathbb{Z}$ this problem is a variant of the classical knapsack problem for binary encoded integers, which is known to be NP-complete. This makes it interesting to look for groups where the compressed knapsack problem belongs to NP. In [32] it was shown that compressed knapsack for every virtually special group belongs to NP. Here, we prove:

▶ **Theorem 5.** *If G is an infinite hyperbolic group then the compressed knapsack problem for G is* NP*-complete.*

Full proofs can be found in the arXiv version [22].

2 General notations

Zero is included in the set of natural numbers: that is, $\mathbb{N} = \{0, 1, 2, \ldots\}$. Let Σ be a finite alphabet of symbols. The set of all finite words over Σ is denoted with Σ^*. We use $\varepsilon \in \Sigma^*$ to denote the empty word. Suppose that $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$ with $a_i \in \Sigma$. The *length* of w is $|w| = n$. For $0 \leq i \leq n-1$ we define $w[i] = a_i$. For $0 \leq i \leq j \leq n$ we define $w[i:j] = a_i \cdots a_{j-1}$. We use $w[:j]$ to mean $w[0:j]$, the prefix of length j, and we also use $w[i:]$ to mean $w[i:n]$, the suffix of length $n-i$. Note that $w[i:i] = \varepsilon$ and $w = w[:i]w[i:]$ for all $0 \leq i \leq n$. We say that $u \in \Sigma^*$ is a *factor* of $w \in \Sigma^*$ if there exist $x, y \in \Sigma^*$ with $w = xuy$.

Fix a strict linear order $<$ on the alphabet Σ. We extend $<$ to the length-lexicographic order $<_{\mathrm{llex}}$ on Σ^*: for words $u, v \in \Sigma^*$ we have $u <_{\mathrm{llex}} v$ if and only if (i) $|u| < |v|$ or (ii) $|u| = |v|$ and there exist words $x, y, z \in \Sigma^*$ and symbols $a, b \in \Sigma$ such that $a < b$, $u = xay$, and $v = xbz$. Note that $<_{\mathrm{llex}}$ is a well-order on Σ^*. Hence, every non-empty subset $L \subseteq \Sigma^*$ contains a unique smallest element with respect to $<_{\mathrm{llex}}$.

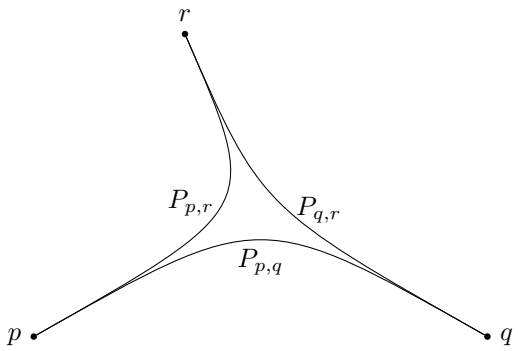


Figure 1 The shape of a geodesic triangle in a hyperbolic group.

3 Hyperbolic groups

Let G be a finitely generated group equipped with a finite, symmetric, generating set Σ. The Cayley graph of G with respect to Σ is the directed edge-labelled graph $\Gamma = \Gamma(G)$ with node set G and all edges of the form (g, ga) for $g \in G$ and $a \in \Sigma$. The edge (g, ga) is labelled with the generator a. Note that for every a-labelled edge (g, h), the reversed edge (h, g) is labelled with a^{-1}. We view Γ as a geodesic metric space (the precise definition of a geodesic metric space is not needed in this paper), where every edge (g, ga) is identified with a unit-length interval. The distance between two nodes $p, q \in \Gamma$ is denoted by $d_\Gamma(p, q)$. For $g \in G$ let $|g| := d_\Gamma(1, g)$; so $|g|$ is the length of a shortest word in Σ^* that represents g. For $r \geq 0$, let $\mathcal{B}_r(1) = \{g \in G \colon d_\Gamma(1, g) \leq r\}$.

Given a word $w \in \Sigma^*$, one obtains a unique path $\mathcal{P}[w]$ that starts at 1 and is labelled by the word w. This path ends in the group element represented by w. More generally, for $g \in G$ we denote by $g \cdot \mathcal{P}[w]$ the path that starts at g and is labelled by w. We will mostly consider paths of the form $g \cdot \mathcal{P}[w]$. One views $P := g \cdot \mathcal{P}[w]$ as a continuous mapping $P : [0, n] \to \Gamma$ from the real interval $[0, n]$ to Γ, where $n = |w|$. We say that a path $P : [0, n] \to \Gamma$ is path from $P(0)$ to $P(n)$. A path $P : [0, n] \to \Gamma$ is *geodesic* if $d_\Gamma(P(0), P(n)) = n$. A word $w \in \Sigma^*$ is *geodesic* if the path $\mathcal{P}[w]$ is geodesic, which means that there is no shorter word representing the same group element from G. A word $w \in \Sigma^*$ is *shortlex reduced* if it is the length-lexicographically least word that represents the same group element as w. For this, we have to fix an arbitrary linear order on Σ. Note that if $u = xy$ is shortlex reduced then x and y are shortlex reduced too. For a word $u \in \Sigma^*$ we denote by $\mathsf{shlex}(u)$ the unique shortlex reduced word that represents the same group element as u. Whenever appropriate, we identify elements of $\mathcal{B}_r(1)$ with geodesic words over Σ of length at most r.

A geodesic triangle consists of three points $p, q, r \in \Gamma$ and geodesic paths $P_{p,q}$, $P_{p,r}$, $P_{q,r}$ (the three sides of the triangle), where $P_{x,y}$ is a path from x to y. We call a geodesic triangle δ-*slim* for $\delta \geq 0$, if every point p on one of the three sides has distance at most δ from a point p' belonging to one of the two sides that are opposite to p. The group G is called δ-hyperbolic if every geodesic triangle is δ-slim. Finally, G is hyperbolic, if it is δ-hyperbolic for some $\delta \geq 0$. Figure 1 shows the shape of a geodesic triangle in a hyperbolic group. The property of being hyperbolic is independent of the chosen finite generating set Σ, but the constant δ depends in general on the chosen finite generating set. Finitely generated free groups are for instance 0-hyperbolic, if the generating set is a free basis. The word problem for every hyperbolic group can be decided in real time [21]. Moreover, one can compute $\mathsf{shlex}(w)$ from a given word w in linear time; see [12] where the result is attributed to Shapiro.

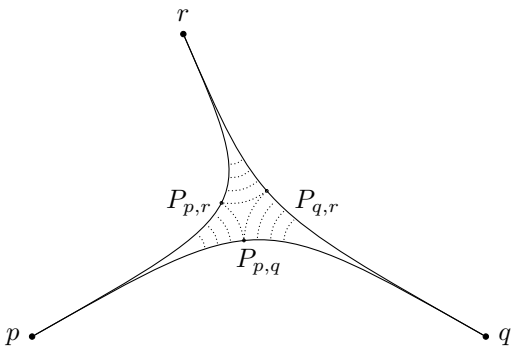


Figure 2 A δ-thin triangle in a hyperbolic group. Dotted lines represent geodesic paths of length at most δ.

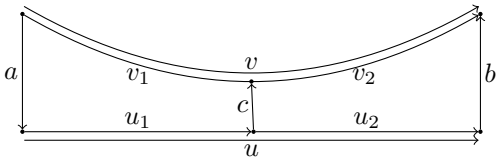


Figure 3 Splitting a geodesic rectangle according to Lemma 7.

We will need an equivalent definition of hyperbolicity in terms of so-called thin triangles. Again, consider three points $p, q, r \in \Gamma$ and let $P_{x,y}$ for $x, y \in \{p, q, r\}$ be a geodesic path from x to y, where $P_{y,x}$ is the path $P_{x,y}$ traversed in the reversed direction. Moreover, let $d_{x,y} = d_\Gamma(x, y)$ be the length of $P_{x,y}$. The three lengths $d_{p,q}$, $d_{p,r}$ and $d_{q,r}$ fulfil the triangle inequality. From this one can deduce real numbers $s_p, s_q, s_r \geq 0$ such that $s_x + s_y = d_{x,y}$ for all $x, y \in \{p, q, r\}$ with $x \neq y$. The geodesic triangle determined by the three sides $P_{p,q}$, $P_{p,r}$, $P_{q,r}$ is called *δ-thin* for $\delta \geq 0$, if for all x, y, z with $x \in \{p, q, r\}$ and $\{y, z\} = \{p, q, r\} \setminus \{x\}$ we have $d_\Gamma(P_{x,y}(t), P_{x,z}(t)) \leq \delta$ for all $t \in [0, s_x]$; see Figure 2. It is well known (see for example [23, Theorem 6.1.3]) that in a δ-hyperbolic group every geodesic triangle is δ'-thin for some constant $\delta' \geq \delta$.

Let us fix a δ-hyperbolic group G with the finite symmetric generating set Σ for the rest of the section, and let Γ be the corresponding geodesic metric space. By choosing δ large enough, we can assume that all geodesic triangles in Γ are δ-thin. We need a few well-known results about hyperbolic groups.

▶ **Lemma 6** (c.f. [11, Theorem 3.4.5]). *The set* $\{\mathsf{shlex}(u) \colon u \in \Sigma^*\}$ *is a regular language.*

The proofs of the following two simple lemmas can be found in [22].

▶ **Lemma 7.** *Let $a, b, u, v \in \Sigma^*$ be geodesic words such that $v =_G aub$ and consider a factorisation $v = v_1 v_2$ with $|v_1| \geq |a| + 2\delta$ and $|v_2| \geq |b| + 2\delta$. Then there exists a factorisation $u = u_1 u_2$ and a geodesic word c with $|c| \leq 2\delta$ such that $v_1 =_G au_1 c$ and $v_2 =_G c^{-1} u_2 b$.*

The situation in Lemma 7 is shown in Figure 3.

▶ **Lemma 8.** *For $i \in \{0, 1, 2\}$ let $P_i : [0, n_i] \to \Gamma$ be geodesic paths such that $P_0(0) = P_1(0)$, $P_0(n_0) = P_2(0)$ and $P_1(n_1) = P_2(n_2)$ (so P_0, P_1, P_2 form a geodesic triangle). Let $j \leq \min\{n_0, n_1\}$ be any integer such that $d_\Gamma(P_0(j), P_1(j)) > \delta$. Then there exist integers $i_0, i_1 \in [0, n_2]$ with $i_0 \leq i_1$, $d_\Gamma(P_0(j), P_2(i_0)) \leq \delta$, and $d_\Gamma(P_1(j), P_2(i_1)) \leq \delta$.*

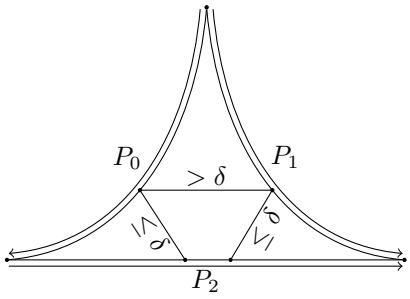


Figure 4 The situation from Lemma 8.

Lemma 8 follows easily from the fact that the geodesic triangle with sides P_0, P_0, and P_1 is thin. The situation is shown in Figure 4.

4 Compressed words and the compressed word problem

A *straight-line program* (SLP for short) over the alphabet Σ is a triple $\mathcal{G} = (V, \rho, S)$, where V is a finite set of variables such that $V \cap \Sigma = \emptyset$, $S \in V$ is the start variable, and $\rho : V \to (V \cup \Sigma)^*$ is a mapping such that the relation $\{(B, A) \in V \times V : B \text{ occurs in } \rho(A)\}$ is acyclic. For the reader familiar with context-free grammars, it might be helpful to view the SLP $\mathcal{G} = (V, \rho, S)$ as the context-free grammar (V, Σ, P, S), where P contains all productions $A \to \rho(A)$ for $A \in V$. The definition of an SLP implies that this context-free grammar derives exactly one terminal word, which will be denoted by $\mathsf{val}(\mathcal{G})$. One can define this string inductively as follows. First, for every $A \in V$ we define $\mathsf{val}_{\mathcal{G}}(A)$. Assume that $\rho(A) = w_0 A_1 w_1 \cdots A_k w_k$ with $k \geq 0$, $w_i \in \Sigma^*$ and $A_i \in V$. Then we define $\mathsf{val}_{\mathcal{G}}(A) = w_0 \mathsf{val}_{\mathcal{G}}(A_1) w_1 \cdots \mathsf{val}_{\mathcal{G}}(A_k) w_k$. Finally, we define $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(S)$.

The word $\rho(A)$ is also called the *right-hand side* of A. We define the size of the SLP $\mathcal{G} = (V, \rho, S)$ as the total length of all right-hand sides: $|\mathcal{G}| = \sum_{A \in V} |\rho(A)|$. SLPs offer a succinct representation of words that contain many repeated substrings. For instance, the word $(ab)^{2^n}$ can be produced by the SLP $\mathcal{G} = (\{A_0, \ldots, A_n\}, \rho, A_n)$ with $\rho(A_0) = ab$ and $\rho(A_{i+1}) = A_i A_i$ for $0 \leq i \leq n-1$.

Quite often, it is convenient to assume that all right-hand sides are of the form $a \in \Sigma$ or BC with $B, C \in V$. This corresponds to the well-known Chomsky normal form for context-free grammars. There is a simple linear time algorithm that transforms an SLP $\mathcal{G}$ with $\mathsf{val}(\mathcal{G}) \neq \varepsilon$ into an SLP $\mathcal{G}'$ in Chomsky normal form with $\mathsf{val}(\mathcal{G}) = \mathsf{val}(\mathcal{G}')$, see for example [30, Proposition 3.8]. We use the fact that the following algorithmic tasks for SLPs can be solved in polynomial time; see also [30, Proposition 3.9].

- Given an SLP $\mathcal{G}$, compute the length $|\mathsf{val}(\mathcal{G})|$.
- Given an SLP $\mathcal{G}$ and an integer $0 \leq i < |\mathsf{val}(\mathcal{G})|$, compute the symbol $\mathsf{val}(\mathcal{G})[i]$.
- Given an SLP $\mathcal{G}$ and integers $0 \leq i \leq j \leq |\mathsf{val}(\mathcal{G})|$, compute an SLP for $\mathsf{val}(\mathcal{G})[i : j]$.

Also the following two propositions are well-known:

▶ **Proposition 9** (c.f. [6, Lemma 2]). *For a given SLP $\mathcal{G}$ and $n \in \mathbb{N}$, we can compute an SLP $\mathcal{G}_n$ with $\mathsf{val}(\mathcal{G}_n) = \mathsf{val}(\mathcal{G})^n$ in time $O(|\mathcal{G}| + \log n)$.*

▶ **Proposition 10** (c.f. [30, Theorem 3.11]). *Given a deterministic finite state automaton M over the alphabet Σ and an SLP $\mathcal{G}$ over the alphabet Σ, we can determine in polynomial time whether $\mathsf{val}(\mathcal{G})$ is in the language $L(M)$ of M.*

Finally, we need the following fundamental result:

▶ **Theorem 11** (c.f. [39]). *Given two SLPs $\mathcal{G}$ and $\mathcal{H}$, one can check in polynomial time whether* $\mathsf{val}(\mathcal{G}) = \mathsf{val}(\mathcal{H})$.

The *compressed word problem* for a finitely generated group G with the finite symmetric generating set Σ is the following decision problem:
Input: an SLP $\mathcal{G}$ over the alphabet Σ.
Question: does $\mathsf{val}(\mathcal{G})$ represent the group identity of G?
It is an easy observation that the computational complexity of the compressed word problem for G does not depend on the chosen generating set Σ in the sense that if Σ' is another finite symmetric generating set for G, then the compressed word problem for G with respect to Σ is logspace reducible to the compressed word problem for G with respect to Σ' [30, Lemma 4.2]. Therefore we do not have to specify the generating set.

5 The compressed word problem for hyperbolic groups

Fix a δ-hyperbolic group G with the finite symmetric generating set Σ, where $\delta > 0$ is chosen in such a way that all geodesic triangles are δ-thin. We can moreover assume that δ is an integer (later, we want to cut off from a word its prefix and suffix of length a certain multiple of δ). Let us set $\zeta := 2\delta \geq 1$ for the following.

We need an extension of SLPs by two operators: so-called tether operators and cut operators. A TCSLP (T stands for "tethered", C stands for "cut") over the alphabet Σ is a tuple $\mathcal{G} = (V, \rho, S)$, where V is a finite set of variables such that $V \cap \Sigma = \emptyset$, $S \in V$ is the start variable, and ρ is a mapping with domain V such that for every $A \in V$, $\rho(A)$ (the right-hand side of A) is of one of the following forms:
(1) a word $w \in (V \cup \Sigma)^*$
(2) an expression $B[: i]$ or $B[i :]$ with $B \in V$ and $i \in \mathbb{N}$ ($[: i]$ and $[i :]$ are called cut operators),
(3) an expression $B\langle a, b\rangle$ with $B \in V$ and $a, b \in \mathcal{B}_\zeta(1)$ ($\langle a, b\rangle$ is called a tether operator).
Moreover, we require that the relation $\{(B, A) \in V \times V : B \text{ occurs in } \rho(A)\}$ is acyclic. The reflexive and transitive closure of this relation is denoted with $\leq_{\mathcal{G}}$. We evaluate variables of type (1) as for SLPs. If $\rho(A) = B[: i]$ or $\rho(A) = B[i :]$ then we define $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(B)[: i]$ or $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(B)[i :]$, respectively. Finally, if $\rho(A) = B\langle a, b\rangle$ we define $\mathsf{val}_{\mathcal{G}}(A) := \mathsf{shlex}(a\, \mathsf{val}_{\mathcal{G}}(B)\, b)$. The reader might ask what happens if $i > |\mathsf{val}_{\mathcal{G}}(B)|$ in case $\rho(A) = B[: i]$ or $\rho(A) = B[i :]$. This will not occur in the TCSLPs constructed in this paper.

For convenience we will also allow more complex right-hand sides $\rho(A)$ such as for instance $(B[: i]\langle a, b\rangle)(C[j :]\langle c, d\rangle)$. We define the *size* of such a right-hand side as the total number of occurrences of symbols from $\Sigma \cup V$ in the right-hand side. The size of $\mathcal{G}$ is obtained by taking the sum over all variables. Note that the numbers $i \in \mathbb{N}$ in cut operators do not contribute to the size of a TCSLP. This does not cause any problems. If one encodes these numbers in binary notation and adds the lengths of these encodings to the size of an TCSLP, then this would only increase the size by a constant factor.

If right-hand sides of type (2) do not occur, we speak of a TSLP and if right-hand sides of type (3) do not occur, we speak of a CSLP. CSLPs are also known as *composition systems* and have been studied before, see for example [18]. In [28], CSLPs are used in the polynomial time algorithm for the compressed word problem of a free group.

We say that the TCSLP $\mathcal{G}$ is *shortlex*, if for every variable A, the word $\mathsf{val}_{\mathcal{G}}(A)$ is shortlex reduced. Note that right-hand sides of type (1) may lead to words that are not

shortlex reduced, since the concatenation of two shortlex reduced words is not necessarily shortlex reduced. The goal of this section is to compute from a given shortlex TCSLP $\mathcal{G}$ in polynomial time an SLP $\mathcal{G}'$ such that $\mathsf{val}(\mathcal{G}) =_G \mathsf{val}(\mathcal{G}')$. In a first step we will present such a transformation only for TSLPs. In a second step, we will transform a shortlex TCSLP into an equivalent TSLP. This second step is inspired by the polynomial time transformation of a CSLP into an equivalent SLP from [18].

For a variable A, we define the height, $\mathsf{height}(A)$ for short, inductively by

$$\mathsf{height}(A) = \max\{\mathsf{height}(B) + 1 \colon B \in V \text{ occurs in } \rho(A)\},$$

where $\max \emptyset = 0$. Let $\mathsf{height}(\mathcal{G}) = \mathsf{height}(S)$; it is the length of a longest chain in the partial order $\leq_{\mathcal{G}}$ that ends in S.

▶ **Lemma 12.** *Given a TSLP $\mathcal{G}$ over the alphabet Σ, we can check in polynomial time whether $\mathcal{G}$ is shortlex. Moreover, if $\mathcal{G}$ is shortlex, then we can compute in polynomial time an SLP $\mathcal{G}'$ over G such that $\mathsf{val}(\mathcal{G}) =_G \mathsf{val}(\mathcal{G}')$.*

Proof. Let $\mathcal{G} = (V, \rho, S)$. In the same way as for SLPs, we can assume that all right-hand sides from $(V \cup \Sigma)^*$ are of the form $a \in \Sigma$ or BC with $B, C \in V$ (variables with right-hand side ε can be eliminated). Let $\mu = \mathsf{height}(\mathcal{G})$. We will transform $\mathcal{G}$ into the desired SLP $\mathcal{G}'$. This will be done by a bottom-up process; that is we consider the variables in $\mathcal{G}$ in order of increasing height. If $\mathcal{G}$ is not shortlex, we will detect this during the transformation.

For a variable A, we define the tether-height, $\mathsf{theight}(A)$ for short, inductively as follows:

- if $\rho(A) = a$, then $\mathsf{theight}(A) = 0$,
- if $\rho(A) = BC$, then $\mathsf{theight}(A) = \max\{\mathsf{theight}(B), \mathsf{theight}(C)\}$, and
- if $\rho(A) = B\langle s, t\rangle$ then $\mathsf{theight}(A) = \mathsf{theight}(B) + 1$.

By removing unused variables, we can assume that S has maximal height and maximal tether height among all variables. For a nonterminal A we define $\eta_A := \mathsf{theight}(S) - \mathsf{theight}(A) + 1 > 0$.

Consider a nonterminal A. Since we are processing the variables in order of increasing height, we can assume that for all $B <_{\mathcal{G}} A$ the word $\mathsf{val}_{\mathcal{G}}(B)$ is shortlex reduced. Let $w := \mathsf{val}_{\mathcal{G}}(A)$. If $|w| \leq 16\zeta\eta_A + 2\zeta$ then we will explicitly compute the word w in the process of defining the SLP $\mathcal{G}'$. Otherwise, we will compute explicitly words ℓ_A, r_A such that $w = \ell_A w' r_A$ for some word w' of length at least 2ζ. The words ℓ_A and r_A will satisfy the length constraints $8\zeta\eta_A \leq |\ell_A|, |r_A| \leq 8\zeta\eta_A + 2\zeta\mathsf{height}(A)$. Moreover, in the latter case, the SLP $\mathcal{G}'$ will contain variables $A'_{a,b}$ for all $a, b \in \mathcal{B}_\zeta(1)$ such that $\mathsf{val}_{\mathcal{G}'}(A'_{a,b}) = \mathsf{shlex}(aw'b)$. The $A'_{a,b}$, together with a start variable S', which will be added at the end of the process, will be the only variables that we include in the SLP $\mathcal{G}'$. All of the words that we compute and store, such as the ℓ_A and r_A, are to enable us to carry out the necessary computations, and are not stored as part of $\mathcal{G}'$.

We now make a case distinction on the form of the right-hand side $\rho(A)$. We only consider the most difficult case that $\rho(A) = B\langle a, b\rangle$ for $a, b \in \mathcal{B}_\zeta(1)$.

Let $u := \mathsf{val}_{\mathcal{G}}(B)$ and $v := \mathsf{val}_{\mathcal{G}}(A) = \mathsf{shlex}(aub)$. The word u is shortlex reduced by assumption, and v is shortlex reduced by definition. Let $\eta = \eta_B$. We have $\eta_A = \eta - 1 \geq 1$. If $|u| \leq 16\zeta\eta + 2\zeta$ then we have explicitly computed the word u. We explicitly compute the word $v = \mathsf{shlex}(aub)$, and then distinguish the cases $|v| \leq 16\zeta\eta + 2\zeta$ and $|v| > 16\zeta\eta + 2\zeta$. In the first case, there is nothing to do. If $|v| > 16\zeta\eta + 2\zeta$, we factorise v as $v = \ell_A v' r_A$ with $|\ell_A| = |r_A| = 8\zeta\eta$, and thus $|v'| \geq 2\zeta$. We can compute for all $a, b \in \mathcal{B}_\zeta(1)$ the word $\mathsf{shlex}(av'b)$ and set $\rho'(A'_{a,b}) = \mathsf{shlex}(av'b)$.

Now assume that $|u| > 16\zeta\eta + 2\zeta$. We have computed words ℓ_B, r_B such that $8\zeta\eta \leq |\ell_B|, |r_B| \leq 8\zeta\eta + 2\zeta\mathsf{height}(B)$ and $u = \ell_B u' r_B$ for a word u' of length at least 2ζ. Moreover, we have already defined variables $B'_{c,d}$ for all $c, d \in \mathcal{B}_\zeta(1)$, which produce $\mathsf{shlex}(cu'd)$.

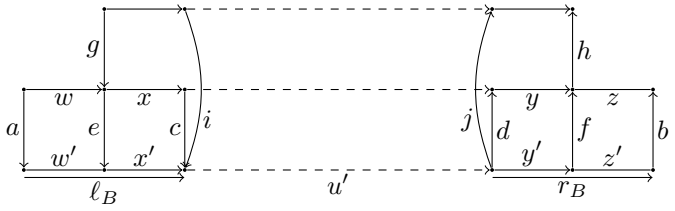


Figure 5 The situation from the proof of Lemma 12. Dotted lines represent words that are given by SLPs.

Using Proposition 10 we check in polynomial time for all $c, d \in \mathcal{B}_\zeta(1)$ whether the word

$$\mathsf{shlex}(a\ell_B c^{-1})\mathsf{val}_{\mathcal{G}'}(B'_{c,d})\mathsf{shlex}(d^{-1}r_B b) = \mathsf{shlex}(a\ell_B c^{-1})\mathsf{shlex}(cu'd)\mathsf{shlex}(d^{-1}r_B b)$$

is shortlex reduced, in which case it is $\mathsf{shlex}(a\ell_B u' r_B b) = \mathsf{shlex}(aub) = v$; see Figure 5. By Lemma 7, there must exist such $c, d \in \mathcal{B}_\zeta(1)$. Let $s = \mathsf{shlex}(a\ell_B c^{-1})$ and $t = \mathsf{shlex}(d^{-1}r_B b)$. By the triangle inequality, these words have length at least $8\zeta\eta - 2\zeta$. Hence we can factorise these words as $s = wx$ and $t = yz$ with $|w| = |z| = 8\zeta(\eta - 1) = 8\zeta\eta_A \geq 8\zeta$. The words x and y have length at least 6ζ. We set $\ell_A := w$ and $r_A := z$. These words satisfy the required bounds on their lengths. Note that $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{shlex}(aub) = \ell_A x\,\mathsf{shlex}(cu'd)yr_A$ and $|x\,\mathsf{shlex}(cu'd)y| \geq 12\zeta \geq 2\zeta$.

It remains to define the right-hand sides of the variables $A'_{g,h}$ for all $g, h \in \mathcal{B}_\zeta(1)$. Let us fix $g, h \in \mathcal{B}_\zeta(1)$. The lower bounds on the lengths of w, x, y, z allow us to apply Lemma 7 to the geodesic rectangles with sides a, ℓ_B, c, wx and d, r_B, b, yz, respectively (all of these words have been computed explicitly). We can compute in polynomial time $e, f \in \mathcal{B}_\zeta(1)$ and factorisations $\ell_B = w'x'$, $r_B = y'z'$ as shown in Figure 5. By the triangle inequality, the words x' and y' must have length at least 4ζ. Now consider the geodesic rectangle with sides $x'u'y'$, $\mathsf{shlex}(ge)$, $\mathsf{shlex}(fh)$, and $\mathsf{shlex}(gex'u'y'fh)$. Since $|x'|, |y'| \geq 4\zeta$ and $|\mathsf{shlex}(ge)|, |\mathsf{shlex}(fh)| \leq 2\zeta$, we can apply Lemma 7 again: There must exist $i, j \in \mathcal{B}_\zeta(1)$ such that the word

$$\mathsf{shlex}(gex'i^{-1})\,\mathsf{val}_{\mathcal{G}'}(B'_{i,j})\,\mathsf{shlex}(j^{-1}y'fh) = \mathsf{shlex}(gex'i^{-1})\,\mathsf{shlex}(iu'j)\,\mathsf{shlex}(j^{-1}y'fh)$$

is shortlex reduced, in which case the above word is $\mathsf{shlex}(gex'u'y'fh)$. Using Proposition 10, we can compute such $i, j \in \mathcal{B}_\zeta(1)$ in polynomial time. We finally define the right-hand side of $A'_{g,h}$ as $\rho'(A'_{g,h}) = \mathsf{shlex}(gex'i^{-1})\,B'_{i,j}\,\mathsf{shlex}(j^{-1}y'fh)$.

This concludes the definition of the right-hand sides for the variables $A'_{a,b}$. We complete the definition of $\mathcal{G}'$ by adding a start variable S' to $\mathcal{G}'$ and setting $\rho'(S') = \ell_S S'_{1,1} r_S$. ◀

The next lemma generalises Lemma 12 to TCSLPs.

▶ **Lemma 13.** *Given a TCSLP $\mathcal{G}$ over the alphabet Σ, we can check in polynomial time whether $\mathcal{G}$ is shortlex. Moreover, if $\mathcal{G}$ is shortlex, then we can compute in polynomial time an SLP $\mathcal{G}'$ over G such that $\mathsf{val}(\mathcal{G}) =_G \mathsf{val}(\mathcal{G}')$.*

Proof sketch. The idea of the proof is taken from [18], where it is shown that a CSLP can be transformed in polynomial time into an equivalent SLP. Let $\mathcal{G} = (V, \rho, S)$ be the input TCSLP. We can assume that all right-hand sides from $(V \cup \Sigma)^*$ are of the form $a \in \Sigma$ or BC with $B, C \in V$. By Lemma 12 it suffices to transform $\mathcal{G}$ into an equivalent TSLP. Let $\mu = \mathsf{height}(\mathcal{G})$. Consider a variable A such that $\rho(A) = B[:i]$; the case that $\rho(A) = B[i:]$ can be dealt with analogously. We can assume that $i \leq |\mathsf{val}_{\mathcal{G}}(B)|$ (this will be true for the TCSLP constructed in the proof of Theorem 15 below). By considering the variables in order

of increasing height, we can moreover assume that no cut operator occurs in the right-hand side of any variable $C <_{\mathcal{G}} A$. We then push the operator in $\rho(A)$ towards smaller (with respect to $<_{\mathcal{G}}$) variables. Thereby we add at most μ new variables to the TCSLP. Moreover the height of the TCSLP after the cut elimination is still bounded by μ. Hence, the final TSLP has at most $\mu \cdot |V|$ variables. In addition, every right-hand side of the final TSLP will have length at most $2\zeta + 1$, so its size will be polynomially bounded. ◀

▶ **Lemma 14.** *Given shortlex TCSLPs $\mathcal{G}_0$ and $\mathcal{G}_1$ such that $\mathsf{val}(\mathcal{G}_i)$ represents the group element g_i, we can check in polynomial time, whether $d_\Gamma(g_0, g_1) \leq \delta$. Moreover, if this is true, we can compute $a \in \mathcal{B}_\delta(1)$ such that $g_0 a =_G g_1$.*

Proof. For all $a \in \mathcal{B}_\delta(1)$ we compute, by adding one new variable to $\mathcal{G}_0$, a shortlex TCSLP $\mathcal{G}_{0,a}$ for $\mathsf{shlex}(\mathsf{val}(\mathcal{G}_0)a)$. Using Lemma 13 and Theorem 11 we can check in polynomial time whether $\mathsf{val}(\mathcal{G}_{0,a}) = \mathsf{val}(\mathcal{G}_1)$, which is equivalent to $g_0 a =_G g_1$. ◀

Finally, we can prove the main technical result of this section:

▶ **Theorem 15.** *From a given SLP $\mathcal{G}$ over the alphabet Σ we can compute in polynomial time an SLP $\mathcal{G}'$ for $\mathsf{shlex}(\mathsf{val}(\mathcal{G}))$.*

Proof. By Lemma 13 it suffices to compute in polynomial time a shortlex TCSLP $\mathcal{G}'$ for $\mathsf{shlex}(\mathsf{val}(\mathcal{G}))$. For this, we process $\mathcal{G}$ bottom-up; that is, we consider the variables in order of increasing height. Assume that $\mathcal{G} = (V, \rho, S)$ and that $\mathcal{G}$ is in Chomsky normal form. The TCSLP $\mathcal{G}'$ will contain all variables from V plus some auxiliary variables. Let us write $\mathcal{G}' = (V', \rho', S)$. For every variable $A \in V$ we will have $\mathsf{val}_{\mathcal{G}'}(A) = \mathsf{shlex}(\mathsf{val}_{\mathcal{G}}(A))$. Consider a variable $A \in V$ and assume that, for all variables $B <_{\mathcal{G}} A$, we have already defined $\rho'(B)$ in such a way that $\mathsf{val}_{\mathcal{G}'}(B) = \mathsf{shlex}(\mathsf{val}_{\mathcal{G}}(B))$.

If $\rho(A) = a \in \Sigma$ then we set $\rho'(A) := \mathsf{shlex}(a)$. Now assume that $\rho(A) = BC$. Thus we have already defined TCSLPs for the words $u := \mathsf{shlex}(\mathsf{val}_{\mathcal{G}}(B))$ and $v := \mathsf{shlex}(\mathsf{val}_{\mathcal{G}}(C))$. Moreover, by Lemma 13 we can transform these TCSLPs into SLPs. Using these SLPs, we can compute the lengths $m = |u|$ and $n = |v|$. If $m = 0$ or $n = 0$, then we set $\rho'(A) := C$ or $\rho'(A) := B$, respectively. So let us assume that m and n are both non-zero. Moreover, we only consider the case that $m \leq n$; the other case is symmetric. From the SLP for u we can compute an SLP for u^{-1}. Consider the geodesic paths $P_0 := \mathcal{P}[u^{-1}]$ and $P_1 := \mathcal{P}[v]$. Using Lemma 14 we can check whether $d_\Gamma(P_0(m), P_1(m)) \leq \delta$.

Case 1. $d_\Gamma(P_0(m), P_1(m)) \leq \delta$. In this case, we can compute by Lemma 14 a word a of length at most δ such that $a =_G uv[:m]$. The situation is shown in Figure 6 on the left. We set $\rho'(A) := C[m:]\langle a, 1\rangle$.

Case 2. $d_\Gamma(P_0(m), P_1(m)) > \delta$. Using binary search, we compute an integer $i \in [0, m-1]$ such that $d_\Gamma(P_0(i), P_1(i)) \leq \delta$ and $d_\Gamma(P_0(i+1), P_1(i+1)) > \delta$. For this we store an interval $[p, q] \subseteq [0, m]$ such that $p < q$, $d_\Gamma(P_0(p), P_1(p)) \leq \delta$ and $d_\Gamma(P_0(q), P_1(q)) > \delta$. Initially, we set $p = 0$ and $q = m$, and we stop if $q = p + 1$. In each iteration, we compute $r = \lceil (p+q)/2 \rceil$ and check, using Lemma 14, whether $d_\Gamma(P_0(r), P_1(r)) \leq \delta$ or $d_\Gamma(P_0(r), P_1(r)) > \delta$. In the first case we set $p := r$ and do not change q, and in the second case we set $q := r$ and do not change p. Hence, in each iteration the size of the interval $[p, q]$ is roughly halved. Therefore, the binary search stops after $\mathcal{O}(\log(m))$ iterations, which is polynomial in the input length. In addition to the position i, we can also compute $a \in \mathcal{B}_\delta(1)$ that labels a path from $P_0(i)$ to $P_1(i)$.

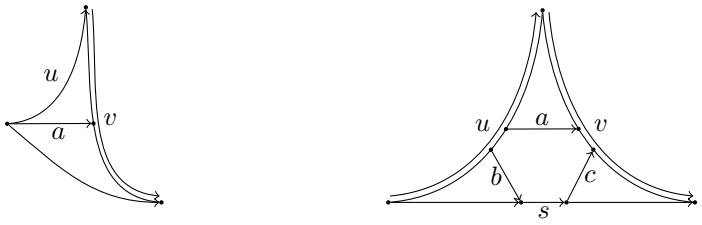


Figure 6 Case 1 (left) and 2 (right) from the proof of Theorem 15.

Let P_2 be the unique geodesic path from $P_0(m)$ to $P_1(n)$ that is labelled with a shortlex reduced word. Note that this path is labelled with $\mathsf{shlex}(uv)$. By Lemma 8 there exist $i_0 \leq i_1$ such that $d_\Gamma(P_0(i+1), P_2(i_0)) \leq \delta$ and $d_\Gamma(P_1(i+1), P_2(i_1)) \leq \delta$. We therefore iterate over all $b, c \in \mathcal{B}_\delta(1)$, compute the word $s := \mathsf{shlex}(b^{-1}u[m-i-1]av[i]c^{-1})$ explicitly, and check whether the word

$$\mathsf{shlex}(u[:m-i-1]b)\, s\, \mathsf{shlex}(cv[i+1:]) \tag{1}$$

is shortlex reduced too, in which case it is $\mathsf{shlex}(uv)$. This can be done using Proposition 10 and using the fact that SLPs for u and v are available. From these SLPs we can compute TCSLPs for $\mathsf{shlex}(u[:m-i-1]b)$ and $\mathsf{shlex}(cv[i+1:])$, which can be transformed into SLPs using Lemma 13. It is guaranteed by Lemma 8 that we will find $b, c \in \mathcal{B}_\delta(1)$ such that the word in (1) is shortlex reduced. For these b, c we finally set $\rho'(A) := (B[:m-i-1]\langle 1, b\rangle)\ s\ (C[i+1:]\langle c, 1\rangle)$. This concludes the proof of the theorem. ◀

A word $w \in \Sigma^*$ represents the group identity if and only if $\mathsf{shlex}(w) = \varepsilon$. Hence, Corollary 1 from the introduction follows directly from Theorem 15.

6 Further compressed decision problems

6.1 Computing the order of a compressed group element

An easy consequence of Corollary 1 is the following result:

▶ **Corollary 16.** *Let G be a hyperbolic group G with the finite symmetric generating set Σ. From a given SLP $\mathcal{G}$ over the alphabet Σ one can compute in polynomial time the order (an element from $\mathbb{N} \cup \{\infty\}$) of the group element represented by $\mathsf{val}(\mathcal{G})$.*

Proof. Let $\mathcal{G}$ be an SLP over the alphabet Σ. It is known that every hyperbolic group has a finite number of conjugacy classes of finite subgroups, and hence that there is a bound on the order of its finite subgroups [23, Theorem 6.8.4]. So there exists a constant $c = c(G)$ such that the order of every element $g \in G$ belongs to $\{1, \ldots, c, \infty\}$. Hence, in order to compute the order of (the group element represented by) $\mathsf{val}(\mathcal{G})$, it suffices to check whether $\mathsf{val}(\mathcal{G})^k =_G 1$ for any $1 \leq k \leq c$. By Corollary 1, this can be done in polynomial time. ◀

6.2 Compressed conjugacy and centralisers

Let G be a finitely generated group G with a fixed finite symmetric generating set for G. For group elements $g, h \in G$ we use the standard abbreviation $g^h = h^{-1}gh$, which is extended to lists $\mathcal{L} = (g_1, \ldots, g_k)$ with $g_i \in G$ by $\mathcal{L}^h = (g_1^h, \ldots, g_k^h)$. We extend these definitions to words over Σ in the obvious way. The *compressed conjugacy problem* for G is the following problem:

Input: SLPs $\mathcal{G}$ and $\mathcal{H}$ over the alphabet Σ.
Question: Do $\mathcal{G}$ and $\mathcal{H}$ represent conjugate elements in G? That is, does there exist $g \in G$ with $\mathsf{val}(\mathcal{G})^g =_G \mathsf{val}(\mathcal{H})$?

More generally, we can define the *compressed simultaneous conjugacy problem* for G:
Input: Finite lists $\mathcal{L}_\mathcal{G} := (\mathcal{G}_1, \ldots, \mathcal{G}_k)$ and $\mathcal{L}_\mathcal{H} := (\mathcal{H}_1, \ldots, \mathcal{H}_k)$ of SLPs over the alphabet Σ.
Question: Do $\mathcal{L}_\mathcal{G}$ and $\mathcal{L}_\mathcal{H}$ represent conjugate lists of elements in G? That is, does there exist $g \in G$ with $\mathsf{val}(\mathcal{G}_i)^g =_G \mathsf{val}(\mathcal{H}_i)$ for all $1 \leq i \leq k$?
In the case when the answer to either of these questions is positive, we might also want to compute an SLP for an element $g \in G$ that conjugates $\mathsf{val}(\mathcal{G})$ to $\mathsf{val}(\mathcal{H})$ or $\mathcal{L}_\mathcal{G}$ to $\mathcal{L}_\mathcal{H}$.

The *compressed centraliser problem* for G is the following computation problem:
Input: A finite list $(\mathcal{G}_1, \ldots, \mathcal{G}_k)$ of SLPs over G.
Output: A finite list of SLPs $(\mathcal{H}_1, \ldots, \mathcal{H}_l)$ such that $\{\mathsf{val}(\mathcal{H}_1), \ldots, \mathsf{val}(\mathcal{H}_l)\}$ is a generating set for the centraliser of the group elements represented by $\mathsf{val}(\mathcal{G}_1), \ldots, \mathsf{val}(\mathcal{G}_k)$.
The proofs of Theorems 2 and 4 from the introduction can be found in the full version [22]. A linear-time algorithm for solving the conjugacy problem of a hyperbolic group G is described in [12, Section 3]. This was generalised in [5] to a linear-time algorithm for the (uncompressed) simultaneous conjugacy problem and the centraliser problem. We show in [22] that essentially the same algorithms (modulo applications of Theorem 15) can be used to solve the compressed (simultaneous) conjugacy problem for G and the compressed centraliser problem in polynomial time.

6.3 Compressed knapsack

Let G be a finitely generated group with the finite symmetric generating set Σ. A *knapsack expression* over G is a rational expression of the form $E = v_0 u_1^* v_1 u_2^* v_2 \cdots u_k^* v_k$ with $k \geq 0$ and $u_i, v_i \in \Sigma^*$. A solution for E is a tuple $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ of natural numbers such that $v_0 u_1^{n_1} v_1 u_2^{n_2} v_2 \cdots u_k^{n_k} v_k =_G 1$. The *length* of E is defined as $|E| = |v_0| + \sum_{i=1}^k |u_i| + |v_i|$.
The knapsack problem for G is the following decision problem:
Input: A knapsack expression E over G.
Question: Does E has a solution?
In [35] it was shown that the knapsack problem for a hyperbolic group can be solved in polynomial time. A crucial step in the proof for this fact is the following result, which is of independent interest:

▶ **Theorem 17** (c.f. [35]). *For every hyperbolic group G there exists a polynomial $p(x)$ such that the following holds: if a knapsack expression $E = v_0 u_1^* v_1 u_2^* v_2 \cdots u_k^* v_k$ over G has a solution then it has a solution $(n_1, \ldots, n_k) \in \mathbb{N}^k$ such that $n_i \leq p(|E|)$ for all $1 \leq i \leq k$.*

Let us now consider the *compressed knapsack problem* for G. It is defined in the same way as the knapsack problem, except that the words $u_i, v_i \in \Sigma^*$ are given by SLPs. The compressed knapsack problem for $\mathbb{Z}$ is NP-complete [17, Proposition 4.1.1]. In fact, this problem corresponds to a variant of the classical knapsack problem for binary encoded integers (for an integer z, it is easy to construct in polynomial time from the binary encoding of z an SLP over the symmetric generating set $\{a, a^{-1}\}$ of $\mathbb{Z}$ which evaluates to a^z or to $(a^{-1})^{-z}$). Using this fact, Corollary 1 and Theorem 17, we can easily deduce Theorem 5 from the introduction, as follows.

Proof of Theorem 5. Consider a knapsack expression $E = v_0 u_1^* v_1 u_2^* v_2 \cdots u_k^* v_k$ over G, where the u_i and v_i are given by SLPs $\mathcal{G}_i$ and $\mathcal{H}_i$, respectively. We then have $|u_i| \leq 3^{|\mathcal{G}_i|/3}$ and $|v_i| \leq 3^{|\mathcal{H}_i|/3}$; see the proof of Lemma 1 in [6] (these bounds on the lengths of the u_i and v_i do

not assume that the $\mathcal{G}_i$ and $\mathcal{H}_i$ are in Chomsky normal form). Let $N := |\mathcal{H}_0| + \sum_{i=1}^{k}(|\mathcal{G}_i| + |\mathcal{H}_i|)$ be the input length. By Theorem 17, there exists a polynomial $p(x)$ such that E has a solution if and only if it has a solution $(n_1, \ldots, n_k) \in \mathbb{N}^k$ such that $n_i \leq p(|E|)$ for all $1 \leq i \leq k$. We obtain a bound of the form $2^{\mathcal{O}(N)}$ on the n_i. Hence, we can guess a tuple $(n_1, \ldots, n_k) \in \mathbb{N}^k$ with all n_i bounded by $2^{\mathcal{O}(N)}$ and then check whether it is a solution of E. The latter can be done in polynomial time by constructing from the SLPs $\mathcal{G}_i$ and $\mathcal{H}_i$ an SLP $\mathcal{G}$ for $v_0 u_1^{n_1} v_1 u_2^{n_2} v_2 \cdots u_k^{n_k} v_k$ using Proposition 9. Finally, we check in polynomial time whether $\mathsf{val}(\mathcal{G}) =_G 1$ using Corollary 1.

The second statement of Theorem 5 follows from the well known fact that every infinite hyperbolic group contains a copy of $\mathbb{Z}$ together with the above mentioned result for $\mathbb{Z}$. ◀

7 Conclusion and open problems

We proved that for every hyperbolic group G, several compressed decision problems (where input words are represented by straight-line programs) can be solved in polynomial time, namely the compressed versions of the following problems: the word problem, computing the order of a group element, the simultaneous conjugacy problem, computing the centralizer of a finite set of group elements, and the knapsack problem.

An important open problem is the precise complexity of the compressed word problem for finitely generated linear groups. We mentioned in the introduction that the compressed word problem for every finitely generated linear group belongs to the complexity class coRP. It is open, whether this upper bound can be improved to P for every finitely generated linear group. This is a very difficult question: as mentioned in the introduction, the compressed word problem for the finitely generated linear group $\mathsf{SL}_3(\mathbb{Z})$ is equivalent (up to polynomial time reductions) to polynomial identity testing. The precise complexity of the latter problem is an outstanding open problem in algebraic complexity theory that is tightly related to lower bounds in circuit complexity theory [24]. But there are many interesting subclasses of finitely generated linear groups, for which it is open whether a polynomial time algorithm for the compressed word problem exists. Let us mention braid groups and Baumslag-Solitar groups $\mathsf{BS}(1, p)$ (for $p \geq 2$) in this context.

Another interesting class of groups, where compressed decision problems have not been considered in depth so far are automaton groups. A concrete open problem is the complexity of the compressed word problem for the Grigorchuk group. The (uncompressed) word problem for the Grigorchuk group can be solved in deterministic logarithmic space [15].

Let us finally mention the compressed variant of the generalized word problem (or subgroup membership problem) for a finitely generated free group $F(\Sigma)$. The input consists of a finite list of SLPs for words $w, w_1, \ldots, w_n \in (\Sigma \cup \Sigma^{-1})^*$ and the question is whether the group element represented by w belongs to the subgroup generated by the group elements represented by the words $w_1, \ldots, w_n$. In the standard (uncompressed) setting this problem can be easily solved in polynomial time using Stalling's folding procedure, see [41] for an efficient implementation. It is open, whether also the compressed generalized word problem for a finitely generated free group can be solved in polynomial time.

References

1 Ian Agol. The virtual Haken conjecture. *Documenta Mathematica*, 18:1045–1087, 2013. With an appendix by Ian Agol, Daniel Groves, and Jason Manning.

2 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.

3 Gilbert Baumslag, Alexei G. Myasnikov, and Vladimir Shpilrain. Open problems in combinatorial group theory. In *Combinatorial and geometric group theory*, pages 1–38. American Mathematical Society, 2002.

4 Martin Beaudry, Pierre McKenzie, Pierre Péladeau, and Denis Thérien. Finite Monoids: From Word to Circuit Evaluation. *SIAM Journal on Computing*, 26(1):138–152, 1997.

5 David J. Buckley and Derek F. Holt. The conjugacy problem in hyperbolic groups for finite lists of group elements. *International Journal of Algebra and Computation*, 23(5):1127–1150, 2013.

6 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

7 François Dahmani and Vincent Guirardel. The Isomorphism Problem for All Hyperbolic Groups. *Geometric and Functional Analysis*, 21(2):223–300, 2011.

8 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144, 1911.

9 Volker Diekert, Jürn Laun, and Alexander Ushakov. Efficient Algorithms for Highly Compressed Data: the Word Problem in Higman's Group is in P. *International Journal of Algebra and Computation*, 22(8), 2012.

10 Will Dison, Eduard Einstein, and Timothy R. Riley. Ackermannian Integer Compression and the Word Problem for Hydra Groups. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

11 David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word Processing in Groups*. Jones and Bartlett, 1992.

12 David B. A Epstein and Derek F. Holt. The Linearity of the Conjugacy Problem in Word-hyperbolic Groups. *International Journal of Algebra and Computation*, 16(2):287–306, 2006.

13 Elizaveta Frenkel, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 74:96–108, 2016.

14 Moses Ganardi, Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack Problems for Wreath Products. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, pages 32:1–32:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

15 Max Garzon and Yechezkel Zalcstein. The complexity of Grigorchuk groups with application to cryptography. *Theoretical Computer Science*, 88(1):83–98, 1991.

16 Mikhail Gromov. Hyperbolic groups. In S. M. Gersten, editor, *Essays in Group Theory*, number 8 in MSRI Publ., pages 75–263. Springer, 1987.

17 Christoph Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, St Catherine's College, 2011.

18 Christian Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, 2000.

19 Frédéric Haglund and Daniel T. Wise. Coxeter groups are virtually special. *Advances in Mathematics*, 224(5):1890–1903, 2010.

20 Nico Haubold, Markus Lohrey, and Christian Mathissen. Compressed decision problems for graph products of groups and applications to (outer) automorphism groups. *International Journal of Algebra and Computation*, 22(8), 2013.

21 Derek F. Holt. Word-hyperbolic groups have real-time word problem. *International Journal of Algebra and Computation*, 10:221–228, 2000.

22 Derek F. Holt, Markus Lohrey, and Saul Schleimer. Compressed decision problems in hyperbolic groups. *CoRR*, 2018. arXiv:1808.06886.

23 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.

24 Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, STOC 1997*, pages 220–229. ACM, 1997.

25 Martin Kassabov and Francesco Matucci. The simultaneous conjugacy problem in groups of piecewise linear functions. *Groups, Geometry, and Dynamics*, 6(2):279–315, 2012.

26 Daniel König and Markus Lohrey. Evaluation of circuits over nilpotent and polycyclic groups. *Algorithmica*, 80(5):1459–1492, 2018.

27 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.

28 Markus Lohrey. Word problems and membership problems on compressed words. *SIAM Journal on Computing*, 35(5):1210–1240, 2006.

29 Markus Lohrey. Algorithmics on SLP-Compressed Strings: A Survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

30 Markus Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014.

31 Markus Lohrey. Knapsack in Hyperbolic Groups. In *Proceedings of the 12th International Conference on Reachability Problems, RP 2018*, volume 11123 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 2018.

32 Markus Lohrey and Georg Zetzsche. Knapsack in Graph Groups. *Theory of Computing Systems*, 62(1):192–246, 2018.

33 Jeremy Macdonald. Compressed words and automorphisms in fully residually free groups. *International Journal of Algebra and Computation*, 20(3):343–355, 2010.

34 Jeremy MacDonald, Alexei G. Myasnikov, and Denis Ovchinnikov. Low-complexity computations for nilpotent subgroup problems. *CoRR*, abs/1706.01092, 2017. arXiv:1706.01092.

35 Alexei G. Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack Problems in Groups. *Mathematics of Computation*, 84:987–1016, 2015.

36 Alexei G. Myasnikov, Alexander Ushakov, and Dong Wook Won. The Word Problem in the Baumslag group with a non-elementary Dehn function is polynomial time decidable. *Journal of Algebra*, 345(1):324–342, 2011.

37 Alexei G. Myasnikov, Alexander Ushakov, and Dong Wook Won. Power circuits, exponential algebra, and time complexity. *International Journal of Algebra and Computation*, 22(6), 2012.

38 Alexander Yu. Ol'shanskii. Almost every group is hyperbolic. *International Journal of Algebra and Computation*, 2(1):1–17, 1992.

39 Wojciech Plandowski. Testing Equivalence of Morphisms on Context-Free Languages. In *Proceedings of the 2nd Annual European Symposium on Algorithms, ESA 1994*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470. Springer, 1994.

40 Saul Schleimer. Polynomial-time word problems. *Commentarii Mathematici Helvetici*, 83(4):741–765, 2008.

41 Nicholas W. M. Touikan. A Fast Algorithm for Stallings' Folding Process. *International Journal of Algebra and Computation*, 16(6):1031–1046, 2006.

42 Stephan Waack. The Parallel Complexity of Some Constructions in Combinatorial Group Theory. *Journal of Information Processing and Cybernetics EIK*, 26:265–281, 1990.

43 Daniel T. Wise. Research announcement: the structure of groups with a quasiconvex hierarchy. *Electronic Research Announcements in Mathematical Sciences*, 16:44–55, 2009.

How to Secure Matchings Against Edge Failures

Felix Hommelsheim
Department of Mathematics, TU Dortmund University, Germany
felix.hommelsheim@math.tu-dortmund.de

Moritz Mühlenthaler
Department of Mathematics, TU Dortmund University, Germany
moritz.muehlenthaler@math.tu-dortmund.de

Oliver Schaudt
Department of Mathematics, RWTH Aachen University, Germany
schaudt@mathc.rwth-aachen.de

Abstract

Suppose we are given a bipartite graph that admits a perfect matching and an adversary may delete any edge from the graph with the intention of destroying all perfect matchings. We consider the task of adding a minimum cost edge-set to the graph, such that the adversary never wins. We show that this problem is equivalent to covering a digraph with non-trivial strongly connected components at minimal cost. We provide efficient exact and approximation algorithms for this task. In particular, for the unit-cost problem, we give a $\log_2 n$-factor approximation algorithm and a polynomial-time algorithm for chordal-bipartite graphs. Furthermore, we give a fixed parameter algorithm for the problem parameterized by the treewidth of the input graph. For general non-negative weights we give tight upper and lower approximation bounds relative to the Directed Steiner Forest problem. Additionally we prove a dichotomy theorem characterizing minor-closed graph classes which allow for a polynomial-time algorithm. To obtain our results, we exploit a close relation to the classical Strong Connectivity Augmentation problem as well as directed Steiner problems.

2012 ACM Subject Classification Hardware → Robustness; Mathematics of computing → Matchings and factors; Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms; Theory of computation → Fixed parameter tractability; Mathematics of computing → Mathematical optimization

Keywords and phrases Matchings, Robustness, Connectivity Augmentation, Graph Algorithms, Treewidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.38

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.01299`.

Funding *Felix Hommelsheim*: Research partially supported by the German Research Foundation (DFG), RTG 1855.
Moritz Mühlenthaler: Research partially supported by the German Research Foundation (DFG), RTG 1855.

Acknowledgements We would like to thank Viktor Bindewald for the fruitful discussions about some of the results in this paper.

1 Introduction

We say that a bipartite graph is *robust* if it admits a perfect matching after the removal of any single edge. Our goal is to make a bipartite graph robust at minimal cost by adding edges from its bipartite complement and we study the complexity of the corresponding optimization problem. We refer to this problem informally as *robust matching augmentation*. In general, an *augmentation problem* asks for a minimum-cost set of edges to be added to a graph in order to establish a certain property. In our context this property is robustness. As a

motivation, consider some assignment-type application, such as staff or task scheduling. The application requires that we choose a perfect matching that assigns, say, tasks to machines. By buying additional edges, we would like to ensure that after the failure of any single edge the resulting graph has a perfect matching, i.e., we may continue our operation. Buying edges may correspond for example to training staff or upgrading machines. Note that in many situations some kind of infrastructure is already available, so it may make sense to upgrade it instead of designing robust infrastructure from scratch.

A *design problem* asks for a minimum-cost subgraph with a certain property, for instance a minimum-cost k-edge-connected subgraph [12, 22]. Robust matching augmentation can be stated also as a design problem, where the given infrastructure is available at zero cost and the host graph is a complete bipartite graph. In fact, our problem is a special case of the bulk-robust assignment problem, a design problem introduced in [2]. Bulk-robustness is a redundancy-based robustness concept proposed by Adjiashvili, Stiller, and Zenklusen [3]. Roughly speaking, the input of a bulk-robust design problem is a host graph and a list of sets of edges, the *failure scenarios*. If a failure scenario emerges then the corresponding edges are deleted from the host graph. The task is to select a minimum-cost subgraph of the host graph that has a certain property (e.g., it contains an assignment [2], a spanning tree [3], or an st-path [4]), no matter which failure scenario emerges. Bulk-robust design problems are notoriously hard. For example, the bulk-robust assignment problem is known to be NP-hard even if only one of two fixed edges may fail [2]. Here, we consider the setting that any single edge of the host graph may fail.

We provide a detailed study of the complexity of the robust matching augmentation problem. For the unweighted problem we give a tight $\log n$-factor approximation algorithm as well as polynomial-time algorithms for chordal-bipartite graphs and graphs of bounded treewidth. For the weighted problem we give a characterization of minor-closed graph classes for which the problem admits a polynomial-time algorithm. Our algorithmic results are based on the following reformulation of the robust matching augmentation problem: Given a digraph, find a minimum-cost superset of its arcs, such that each vertex is contained in some non-trivial strongly connected component. In contrast, the classical *strong connectivity augmentation* problem asks for the minimal number of arcs that are needed to have all vertices covered by a *single* strongly connected component. It was shown by Eswaran and Tarjan that this problem admits a polynomial-time algorithm, but its edge-weighted variant is NP-hard [16]. It turns out that the classical algorithm for strong connectivity augmentation is useful in order to satisfy our more relaxed strong connectivity requirements at minimal cost.

Our Contribution

Recall that we call a bipartite graph *robust* if it admits a perfect matching after the removal of any single edge. For a bipartite graph (V, E), we denote by $\overline{E}$ the edge-set of its bipartite complement. We provide algorithms and hardness results for variants of the following problem.

Robust Matching Augmentation
instance: Undirected bipartite graph $G = (U+W, E)$ that admits a perfect matching.
task: Find a set $L \subseteq \overline{E}$ of minimum cardinality, such that the graph $G + L$ is robust.

Based on a characterization of robustness in terms of strong connectivity, we provide a deterministic $\log_2 n$-factor approximation for Robust Matching Augmentation, as well as a fixed parameter tractable (FPT) algorithm for the same problem parameterized by the

treewidth of the input graph. We also give a polynomial-time algorithm for instances on chordal-bipartite graphs, which are bipartite graphs without induced cycles of length at least six. Furthermore, we show that Robust Matching Augmentation admits no polynomial-time sublogarithmic-factor approximation algorithm unless $\mathsf{P} = \mathsf{NP}$, so our approximation guarantee is essentially tight. We also consider the following more general setting. Let us call a bipartite graph k-robust, if it admits a matching of cardinality k after the removal of any single edge. By a simple reduction we show that our algorithmic results carry over to the task of making a bipartite graph k-robust.

We refer by Weighted Robust Matching Augmentation to the generalization of Robust Matching Augmentation, where each edge $e \in \overline{E}$ has a non-negative cost c_e. The task is to find a minimum-cost set $L \subseteq \overline{E}$, such that $G + L$ is robust. First, we show that the approximability of Weighted Robust Matching Augmentation is closely linked to that of Directed Steiner Forest. In particular we show that an $f(n)$-factor approximation algorithm for Weighted Robust Matching Augmentation implies an $f(n+k)$-factor approximation algorithm for Directed Steiner Forest, where k is the number of terminal pairs. By a result of Halperin and Krauthgamer [23] it follows that there is no $\log^{2-\varepsilon}(n)$-factor approximation for Weighted Robust Matching Augmentation, unless $\mathsf{NP} \subseteq \mathrm{ZTIME}(n^{\mathsf{polylog}(n)})$. On the positive side, we show that an $f(k)$-factor approximation for the Directed Steiner Forest problem yields an $(f(k)+1)$-factor approximation for Weighted Robust Matching Augmentation. Hence, the algorithms from [10, 18] give an approximation guarantee of $1+n^{\frac{1}{2}+\varepsilon}$ for Weighted Robust Matching Augmentation, for every $\varepsilon > 0$.

Second, we prove a complexity dichotomy based on graph minors. Let $\mathcal{T}$ be a class of connected graphs closed under connected minors. We show that Weighted Robust Matching Augmentation restricted to input graphs from $\mathcal{T}$ is NP-complete if $\mathcal{T}$ contains at least one of two simple graph classes, which will be defined in Section 5, and admits a polynomial-time algorithm otherwise. The polynomial-time algorithm for the remaining instance classes uses a reduction to the Directed Steiner Forest problem with a constant number of terminal pairs, which in turn admits a (slice-wise) polynomial-time algorithm due to a result by Feldman and Ruhl [17]. The terminal pairs of the instance are computed by the Eswaran-Tarjan algorithm.

Summary of Algorithmic Techniques

By close inspection, it turns out that in order to make some bipartite graph G robust at minimum cost, we may restrict our attention to failures of single edges from any fixed perfect matching M of G. We then show that the resulting problem is equivalent to augmenting a minimum-cost set A of arcs to a given digraph D, such that in the graph $D + A$, each vertex is contained in a strongly connected component and each strongly connected component contains at least two vertices. In order to satisfy these connectivity requirements, we select certain sources and sinks of the condensation of the digraph and add a minimum-cardinality set of arcs, such that the selected sources and sinks are contained in a single strongly connected component. For this purpose, we use the classical Eswaran-Tarjan algorithm. From the arcs we added we obtain an optimal set L of edges such that $G + L$ is robust, provided that the selection of sources and sinks was optimal.

We model the task of selecting sources and sinks as a variant of the Set Cover problem with some additional structure. Given an acyclic digraph, the task is to select a minimum-cardinality subset of the sources, such that each sink is reachable from at least one of the selected sources. We refer to this problem as Source Cover and remark that its complexity may be of independent interest, since it generalizes Set Cover but is a special case of

Directed Steiner Tree. We give an FPT algorithm for the Source Cover problem parameterized by the treewidth of the input graph and a polynomial-time algorithm for chordal-bipartite graphs (ignoring orientations). The FPT algorithm is single exponential in the treewidth. Our reductions from Robust Matching Augmentation to Source Cover preserve chordal-bipartiteness and bounded treewidth, so efficient algorithms for Source Cover on these graph classes imply efficient algorithms for Robust Matching Augmentation on the same classes.

As a by-product of our analysis of the Source Cover problem, we obtain FPT algorithms for the node-weighted and arc-weighted versions of the Directed Steiner Tree problem on acyclic digraphs, which are exponential in the treewidth and linear in the number of nodes of the input graph.

Related work

In [2], Adjiashvili, Bindewald and Michaels proposed an LP-based randomized algorithm for the bulk-robust assignment problem. They claim an $O(\log n)$-factor approximation guarantee for their algorithm. Since the robust assignment problem generalizes Weighted Robust Matching Augmentation, an $O(\log n)$-factor approximation for our problem is implied. However, due to our inapproximability result for Weighted Robust Matching Augmentation, this can not be true, unless $\mathsf{NP} \subseteq \mathrm{ZTIME}(n^{\mathsf{polylog}(n)})$. The authors of [2] agree that their analysis is incorrect.

A connectivity augmentation problem related to strong connectivity, but of a different flavor, is the *tree augmentation problem* (TAP). The TAP asks for a minimum-cost edge-set that increases the edge-connectivity of a given tree from one to two. In contrast to robust matching augmentation, the TAP admits a constant-factor approximation [21]. The constant has recently been lowered to $3/2 + \varepsilon$ for bounded-weight instances [1, 19]. Similar to robust matching augmentation, the input graph is available at zero cost. Let us briefly remark that there is more conceptual similarity. The *matching preclusion number* of a graph is the minimal number of edges to be removed, such that the remaining graph has no perfect matching. Robust matching augmentation can be stated as the task of finding a minimum-cost edge-set that increases the matching preclusion number of a bipartite graph from one to two, while the TAP aims to increase connectivity from one to two. The matching preclusion number is considered to be a measure of robustness of interconnect networks [9, 11]. Determining the matching preclusion number of a graph is NP-hard [14, 25].

In our reduction from robust matching augmentation problem to a connectivity augmentation problem, we construct a digraph from the input graph and a fixed perfect matching. This construction is closely related to the classical Dulmage-Mendelsohn decomposition (DM-decomposition) introduced in [15]. In fact the digraph from our reduction can be obtained from the auxiliary digraph that is used for computing the DM-decomposition of a graph by contracting the edges of the perfect matching. In [5], the authors consider the problem of making a bipartite graph DM-irreducible, which means that its DM-decomposition consists of a single component. They show that the unweighted variant of this problem admits a polynomial-time algorithm. For balanced bipartite graphs that admit a perfect matching, the problem reduces to the strong connectivity augmentation problem. Hence, DM-irreducibility of such graphs implies robustness, but not vice versa.

Robust perfect matchings with a given recovery budget were studied by Dourado et al. in [14]. Our notion of robustness corresponds to *1-robust* ∞*-recoverable* in their terminology. They provide hardness results and structural insights mainly for fixed recovery budgets, which bound the number of edges that can be changed in order to repair a matching, after a certain number of edges has been removed from the graph.

Notation

Undirected and directed graphs considered here are simple. For sets U, W, we denote by $U+W$ their disjoint union. For an undirected bipartite graph $G = (U+W, E)$ with bipartition (U, W), we denote by $\overline{E}$ the edge-set of its bipartite complement. Let $D = (V, A)$ be a directed graph. We refer by $\overline{A}$ to the arcs not present in D. That is, we let $\overline{A} \subseteq (V \times V) \setminus A$. By $\mathcal{U}(D)$ we refer to the underlying undirected graph of D. For $L \subseteq \overline{E}$, we write $G + L$ for the graph $G' = (V(G), E(G) \cup L)$. Simple paths in graphs are given by a sequence of vertices. For graphs G, H we write $H \subseteq G$ if H is a subgraph of G. Recall that a graph H is an *induced minor* of a graph G if it arises from G by a sequence of vertex deletions and edge contractions. Similarly, the graph H is a *minor* of G if we additionally allow edge deletion. Furthermore, the graph H is a *connected minor* of G if H is connected and a minor of G. In general, contractions may result in parallel edges or loops, which we simply discard in order to keep our graphs simple. Let $\mathcal{G}$ be a class of graphs. We will refer to the restriction of (Weighted) Robust Matching Augmentation to instances where the graph G is bipartite, admits a perfect matching, and belongs to the class $\mathcal{G}$ as *(*Weighted*)* Robust Matching Augmentation *on* $\mathcal{G}$. Given a set of items X and sets $\mathcal{S} \subseteq 2^X$, the Set Cover problems asks for a minimum-cardinality subset $C \subseteq \mathcal{S}$, such that each $x \in X$ is contained in some $s \in C$. The *incidence graph* $G(\mathrm{I})$ of a Set Cover instance $\mathrm{I} = (X, \mathcal{S})$ is an undirected bipartite graph on the vertex set $X + \mathcal{S}$ that has an edge xs if and only if the item $x \in X$ is contained in the set $s \in \mathcal{S}$.

Organization of the Paper

The remainder of the paper is organized as follows. We illustrate the relation between robust matching augmentation and strong connectivity augmentation in Section 2. In Section 3 we show an even closer relation of Robust Matching Augmentation to the Source Cover problem. Algorithms for the Source Cover problem are given in Section 4 as well as our results on robust matching augmentation with unit costs. In Section 5 we give the complexity classification for the weighted version of the problem and Section 6 concludes the paper.

2 Robust Matchings and Strong Connectivity Augmentation

In this section we give some preliminary observations on the close relationship between robust matching augmentation with unit costs and strong connectivity augmentation. For this purpose, we fix an arbitrary perfect matching and construct an auxiliary digraph that is somewhat similar to the *alternating tree* used in Edmond's blossom algorithm. We show that the original graph is robust if the auxiliary graph is strongly connected (but not vice versa). Furthermore, we show that there is an optimal edge-set making the given graph robust, that corresponds to a set of arcs connecting sources and sinks in the auxiliary digraph. Finally, if no source or sink of the auxiliary digraph corresponds to a non-trivial robust part of the original graph, then we may use the algorithm for strong connectivity augmentation by Eswaran and Tarjan [16] to make the original graph robust. As a consequence, we have that Robust Matching Augmentation on trees can be solved efficiently by using the Eswaran-Tarjan algorithm. In Section 3, we will generalize this result.

Let $G = (U + W, E)$ be a bipartite graph that admits a perfect matching and let M be an arbitrary but fixed perfect matching M of G. We call an edge $e \in M$ *critical* if $G - e$ admits no perfect matching. Observe that an edge $e \in M$ is critical if and only if it is not

(a) Graph G and matching M (wiggly edges). **(b)** Digraph $D(G, M)$.

Figure 1 Illustration of the correspondence between the dotted edges of G and dotted arcs of $D(G, M)$.

contained in an M-alternating cycle. Furthermore, no edge in $E \setminus M$ is critical. Since M is perfect, each edge $e \in M$ is incident to a unique vertex u_e of U. We consider the following auxiliary digraph $D(G, M) = (U, A)$, whose arc-set A is given by

$$A := \{uu' \mid u, u' \in U : \text{there is a vertex } w \in W \text{ such that } uw \in M \text{ and } wu' \in E \setminus M\}.$$

We first note that the choice of the bipartition of G is irrelevant.

▶ **Fact 1.** *Let $G' = (U' + W', E)$, where (U', W') is a bipartition of G. Then $D(G, M)$ is isomorphic to $D(G', M)$.*

Note that we may perform the reverse construction as well. That is, from any digraph D' we may obtain a corresponding undirected graph G and a perfect matching M of G such that $D(G, M) = D'$. In fact, augmenting edges to G is equivalent to augmenting arcs to $D(G, M)$.

▶ **Fact 2.** *Let $\overline{A}$ be the set of arcs that are not present in $D(G, M)$. Then there is a 1-to-1 correspondence between $\overline{E}$ and $\overline{A}$.*

An example of the correspondence mentioned in Fact 2 is shown in Figure 1. In order to keep our notation tidy, we will make implicit use of Fact 2 and refer to $\overline{A}$ and $\overline{E}$ interchangeably. Observe that for edges $e, f \in M$ there is an M-alternating path containing e and f in G if and only if u_e is connected to u_f in $D(G, M)$. This implies the following characterization of robustness.

▶ **Fact 3.** *G is robust if and only if each strongly connected component of $D(G, M)$ is non-trivial, that is, it contains at least two vertices.*

Let D' be a digraph. A vertex of D' is called a *source* (*sink*) if it has no incoming (outgoing) arc. We refer to the set of sources (sinks) of D' by $V^+(D')$ ($V^-(D')$). Furthermore, we denote by $C(D')$ the *condensation* of D', that is, the directed acyclic graph of strongly connected components of D'. We call a source or sink of $C(D')$ *strong* if the corresponding strongly connected component of D' is non-trivial. From Fact 3 it follows that a subgraph of G that corresponds to a strong source or a strong sink is robust against the failure of a single edge. Furthermore, observe that the choice of the perfect matching M of G is irrelevant in the following sense.

▶ **Fact 4.** *Let M and M' be perfect matchings of G. Then $C(D(G, M))$ is isomorphic to $C(D(G, M'))$.*

Fact 4 is of key importance for our algorithmic results, for which we generally assume that some fixed perfect matching is given. Next, we observe that for unit costs we may restrict our attention to connecting sources and sinks of $C(D)$ in order to make G robust. It is easy to check that this does not hold for general non-negative costs.

▶ **Fact 5.** *Let $L \subseteq \overline{E}$ such that $G + L$ is robust. Then there is some $L' \subseteq \overline{E}$ of cardinality at most $|L|$, such that $G + L'$ is robust and L' connects only sinks to sources of $C(D(G, M))$.*

We remark that the construction of L' given in the proof of Fact 5 can be performed in polynomial time.

We denote by $\gamma(D')$ the minimal number of arcs to be added to a digraph D' in order to make it strongly connected. Eswaran an Tarjan have proved the following min-max relation [16].

▶ **Fact 6.** *Let D' be a digraph. Then $\gamma(D') = \max\{|V^+(D')|, |V^-(D')|\}$.*

From the proof of Fact 6 it is easy to obtain a polynomial-time algorithm that, given a digraph D', computes an arc-set L of cardinality $\gamma(D')$ such that $D' + L$ is strongly connected [20]. We will refer to this algorithm by Eswaran-Tarjan. The following proposition illustrates the usefulness of the algorithm Eswaran-Tarjan for Robust Matching Augmentation, and at the same time its limitations.

▶ **Fact 7.** *Suppose that $C(D(G, M))$ contains no strong sources or sinks. Then* Eswaran-Tarjan *computes a set $L \subseteq \overline{E}$ of minimum cardinality such that $G + L$ is robust.*

Fact 7 implies that Eswaran-Tarjan solves Robust Matching Augmentation on trees. If strong sources or sinks are present in $D(G, M)$, then we may or may not need to consider them in order to make G robust. This is precisely what makes the problem Robust Matching Augmentation hard. This close connection will be presented in Section 3. We will formalize the task of selecting strong sources and sinks in terms of the Source Cover problem, which is discussed in Section 4.

3 Robust Matching Augmentation

In this section we present our main technical tool for solving the problem Robust Matching Augmentation. By combining it with the results in Section 4 we obtain our algorithmic results. Let us first restate the problem in a slightly different way.

Robust Matching Augmentation
instance: Bipartite graph $G = (U + W, E)$ and perfect matching M of G.
task: Find a minimum-cardinality set $L \subseteq \overline{E}$ such that $G + L$ is robust.

Fixing the perfect matching M in the instance is just for notational convenience, since we can compute a perfect matching in polynomial time and our results do not depend on the exact choice of M, according to the discussion in Section 2. For the main theorem of this section we need to introduce the Source Cover problem. Given an acyclic digraph, the Source Cover problem asks for a minimum-cardinality subset of its sources, such that each sink is reachable from at least one selected source. The Source Cover problem is formally defined as follows.

Source Cover
instance: Weakly connected acyclic digraph $D = (V, A)$.
task: Find a minimum-cardinality subset S of the sources $V^+(D)$ of D, such that for each sink $t \in V^-(D)$ there is an S-t-path in D.

Note that the assumption that D is connected is needed only for technical reasons. Our main technical result is the following.

▶ **Theorem 8.** *There is a polynomial-time algorithm that, given an instance* $\mathrm{I} = (G, M)$ *of* Robust Matching Augmentation*, computes two instances* $\mathcal{A}_1 = (S_1)$ *and* $\mathcal{A}_2 = (S_2)$ *of* Source Cover *such that the following holds.*

1. $\mathcal{U}(S_1)$ *and* $\mathcal{U}(S_2)$ *are induced minors of* $\mathcal{U}(D(G, M))$*.*
2. $\mathrm{OPT}(\mathrm{I}) = \max\{\mathrm{OPT}(\mathcal{A}_1), \mathrm{OPT}(\mathcal{A}_2)\}$
3. *From a solution* C_1 *of* $\mathcal{A}_1$ *and a solution* C_2 *of* $\mathcal{A}_2$ *we can construct in polynomial time a solution* L *of* I *of cardinality* $\max\{|C_1|, |C_2|\}$*.*

Proof. Let $\mathrm{I} = (G, M)$ be an instance of Robust Matching Augmentation, where $G = (U + W, E)$. Our goal is to obtain from solutions of the Source Cover instances a suitable selection of sources and sinks of $C(D(G, M))$, such that we can make M robust by connecting the selected sources and sinks, using the algorithm Eswaran-Tarjan. Let us denote by u_e the vertex in U that is incident to an edge $e \in M$. Furthermore, let $D := D(G, M)$. We construct the Source Cover instance $\mathcal{A}_1$ as follows. For each critical edge $e \in M$, we remove from D each vertex $v \in U - u_e$, such that v is reachable from u_e in D. Let D' be the resulting graph and let the Source Cover instance $\mathcal{A}_1$ be given by $\mathcal{A}_1 := (C(D'))$. The construction of $\mathcal{A}_2$ is as for $\mathcal{A}_1$, but with the arcs of D reversed. This turns the sources of D into sinks. Clearly, the acyclic digraphs of $\mathcal{A}_1$ and $\mathcal{A}_2$ are induced minors of $\mathcal{U}(D)$, since they were constructed by deleting vertices of $\mathcal{U}(D)$ and contracting strong components. By Fact 3, the set of critical edges can be obtained efficiently by Tarjan's classical algorithm for computing strongly connected components. In order to generate $\mathcal{A}_1$ and $\mathcal{A}_2$, observe that D' and $C(D')$ can both be obtained by applying a breadth-first search starting at each vertex of D or D', respectively. So it remains to prove Statement 2 and 3.

Let C_1 (C_2) be a solution to $\mathcal{A}_1$ ($\mathcal{A}_2$). We show how to construct in polynomial time a solution L of I of cardinality $\max\{|C_1|, |C_2|\}$. Let $X \subseteq V(\widehat{D})$ be the set of vertices incident to critical edges. Moreover, let $\widehat{D} \subseteq C(D)$ be the graph induced by the vertices of $C(D)$ that are on C_1X-paths or on XC_2-paths in $C(D)$. Note that $\widehat{D}$ can be computed by a depth-first search applied on each source and sink. By running Eswaran-Tarjan on $\widehat{D}$ we obtain an arc-set L^* such that $\widehat{D} + L^*$ is strongly connected. Hence, each $u \in X$ is on some directed cycle in $\widehat{D} + L^*$. From L^* we can obtain in a straight-forward way an arc-set L of the same cardinality, such that each $u \in X$ is on some directed cycle of $D + L$. For each $ss' \in L^*$, we add to L an arc uu', where u (u') is some vertex in the strong component s (s') of D. By the construction of L, each $u \in X$ is on some directed cycle of D. By Fact 2 and 6 we have constructed a solution L of I of cardinality $|L| = |L^*| = \max\{|C_1|, |C_2|\}$. This completes the proof of Statement 3.

It remains to prove that $\mathrm{OPT}(\mathrm{I}) \geq \max\{\mathrm{OPT}(\mathcal{A}_1), \mathrm{OPT}(\mathcal{A}_2)\}$. Suppose for a contradiction that $\mathrm{OPT}(\mathrm{I}) < \max\{\mathrm{OPT}(\mathcal{A}_1), \mathrm{OPT}(\mathcal{A}_2)\}$. Without loss of generality, let $\mathrm{OPT}(\mathcal{A}_1)$ attain the maximum. Due to Fact 5, we may assume that an optimal solution L of I connects sources and sinks of $C(D)$. Let $R \subseteq V(C(D))$ be the corresponding sources of $C(D)$. Then for each critical edge $e \in M$, the vertex u_e must be reachable from some source $s \in R$. But then R is a solution of $\mathcal{A}_1$ of cardinality $|R| = \mathrm{OPT}(\mathrm{I}) < \mathrm{OPT}(\mathcal{A}_1)$, a contradiction. ◀

By Theorem 8, in order to solve Robust Matching Augmentation, is suffices to solve two instances of Source Cover. Due to Statement 1 of the theorem, structural features of the input graph, such as bounded treewidth and chordal-bipartiteness, are passed on to the digraphs of the source cover instances. We now consider the following more general setting.

We call a bipartite graph k-*robust* if it admits a matching of cardinality k after the removal of any single edge.

Robust k-Matching Augmentation
instance: Bipartite graph $G = (U + W, E)$ that admits a matching of size k.
task: Find a minimum-cardinality set $L \subseteq \overline{E}$ such the graph $G + L$ is k-robust.

Note that if k is less than the size of a maximum matching then $L = \emptyset$ is optimal due to the existence of a larger matching. We give a polynomial-time reduction from Robust k-Matching Augmentation to Robust Matching Augmentation. Let (G, M) be an instance of Robust k-Matching Augmentation, where the input graph G is given by $G = (V, E)$. Without loss of generality, we assume that M is U-perfect, so $|U| \leq |W|$. Otherwise, adding an edge joining two unmatched vertices solves the problem. We construct an instance (G', M') of Robust Matching Augmentation as follows. Let G' be a copy of G to which we add a leaf to each unmatched vertex of W. We then add a vertex z to U joined to each vertex of the other part of the bipartition. Finally, we add a vertex z' joined to z and each leaf added in the previous step. Furthermore, we extend the matching M of G to a perfect matching M' of G' by adding the edges between the leaves and the previously unmatched vertices to M'. Note that by construction, if e is a critical edge of G' then $G - e$ does not admit a matching of cardinality $|M|$.

Note that the construction increases the treewidth by at most two, but does not preserve chordal-bipartiteness of the input graph. However, the corresponding digraph contains no induced cycle of length at least six, so all our algorithmic results for Robust Matching Augmentation carry over to Robust k-Matching Augmentation.

▶ **Proposition 9.** *There is a polynomial-time reduction f from* Robust k-Matching Augmentation *to* Robust Matching Augmentation*, such that the following holds. Let* $\mathrm{I} := (G)$ *be an instance of* Robust k-Matching Augmentation *and let* $f(\mathrm{I}) = (G')$*. Then*

1. $\mathrm{OPT}(f(\mathrm{I})) = \mathrm{OPT}(\mathrm{I})$ *and from a solution L' of $f(\mathrm{I})$ we can construct in polynomial time a solution L of* I *such that $|L| \leq |L'|$.*
2. $tw(G') \leq tw(G) + 2$
3. *If G is chordal-bipartite then $\mathcal{U}(D(G', M'))$ has no induced cycle of length at least six.*

4 The Source Cover Problem

In Section 3 we made precise the close relation between Robust Matching Augmentation and the Source Cover problem. In this section we present our algorithmic results for the Source Cover problem as well as their consequences consequences for Robust k-Matching Augmentation. Recall that the Source Cover problem asks for a minimum-cardinality subset of the source of a given digraph, such that each sink is reachable from at least one selected source. It is easy to see that Source Cover is a special case of the Directed Steiner Tree problem and that it generalizes Set Cover. We give a simple polynomial-time algorithm for Source Cover if the input graph is chordal-bipartite (ignoring orientations). Furthermore, we show that Source Cover parameterized by treewidth (again ignoring orientations) is FPT. As a by-product, we obtain a simple FPT algorithm for the arc-weighted and node-weighted versions of the Directed Steiner Tree problem on acyclic digraphs, whose running time is linear in the size of the input graph and exponential in the treewidth of the underlying undirected graph. To the best of our knowledge, the parameterized complexity of the general Directed Steiner Tree problem with respect to treewidth is open. For the corresponding *undirected* Steiner Tree problem, an FPT algorithm was given by Bodlaender et al. in [8].

(a) A digraph D such that $\mathcal{U}(D)$ is balanced, but $\mathcal{U}(F(D))$ is not.

(b) Digraphs D such that $\mathcal{U}(D)$ has treewidth one, but the treewidth of $\mathcal{U}(F(D))$ is unbounded.

Figure 2 Examples showing that flattening does not preserve balancedness or bounded treewidth.

By "flattening" the input digraph, we can transform an instance $\mathrm{I} = (D)$ of Source Cover into a Set Cover instance as follows. Let $F(D) = (V^+(D) \cup V^-(D), A')$ be an acyclic digraph, where A' is given by

$$A' := \{st \mid s \in V^+(D),\, t \in V^-(D),\, t \text{ is reachable from } s \text{ in } D\}.$$

Then $\mathcal{U}(F(D))$ is the incidence graph of a Set Cover instance $\mathcal{A}$ on $V^-(F(D))$, such that the feasible solutions of I and $\mathcal{A}$ are in 1-to-1 correspondence.

As a first consequence of Theorem 8, Proposition 9, and this "flattening" we may use the classic Greedy-Algorithm for Set Cover obtain a simple $\log_2 n$-factor approximation algorithm for Robust k-Matching Augmentation.

▶ **Corollary 10.** Robust k-Matching Augmentation *admits a polynomial-time* $\log_2 n$*-factor approximation algorithm, where* n *is the number of vertices of the input graph.*

However, as illustrated in Figure 2, some useful properties of the input digraph may not be preserved by the "flattening" operation. In particular, if $\mathcal{U}(D)$ has treewidth at most r, then the treewidth of $\mathcal{U}(F(D))$ cannot be bounded by a constant in general. Furthermore, the graph $\mathcal{U}(F(D))$ is not necessarily balanced[1] (or planar) if $\mathcal{U}(D)$ is. Therefore, we cannot take advantage of polynomial-time algorithms for Set Cover on balanced incidence graphs or incidence graphs of bounded treewidth. Motivated by the example in Figure 2b we leave as an open question, whether Source Cover on balanced graphs admits a polynomial-time algorithm. By Theorem 8, the existence of such an algorithm implies a polynomial-time algorithm for Robust Matching Augmentation on balanced graphs.

4.1 Source Cover on Chordal Bipartite Graphs

We show that in contrast to the treewidth and balancedness, chordal-bipartiteness is indeed preserved by the flattening operation introduced above. From this we obtain the following result.

▶ **Theorem 11.** Source Cover *on chordal-bipartite graphs admits a polynomial-time algorithm.*

[1] A graph is called *balanced* if the length of each induced cycle is divisible by four.

To prove the theorem, we show that if $\mathcal{U}(D)$ is chordal-bipartite, so is $\mathcal{U}(F(D))$. The graph $\mathcal{U}(F(D))$ is the incidence graph of a Set Cover instance, whose optimal solutions correspond canonically to the optimal solutions of the Source Cover instance (D). It is known that Set Cover on chordal-bipartite incidence graphs (and more generally, balanced graphs) admits a polynomial-time algorithm: It is possible to use LP-methods and the fact that covering polyhedra of balanced matrices are integral, see [26, pp. 562-573]. Alternatively we can use a combinatorial algorithm by Hoffman et al. [24]. By combining Theorem 8, Proposition 9, and Theorem 11 we obtain the following result.

▶ **Corollary 12.** Robust k-Matching Augmentation *admits a polynomial-time algorithm on chordal-bipartite graphs.*

4.2 Source Cover on Graphs of Bounded Treewidth

We provide a fixed-parameter algorithm for Node Weighted Directed Steiner Tree on acyclic digraphs that is single-exponential in the treewidth of the underlying undirected graph and linear in the instance size. Since Source Cover is a restriction of Node Weighted Directed Steiner Tree on acyclic graphs, this implies a polynomial-time algorithm for Source Cover parameterized by the treewidth of the underlying undirected graph. Let us first recall some definitions related to Steiner problems and tree decompositions.

Node Weighted Directed Steiner Tree
instance: Acyclic digraph $D = (V, A)$, costs $c \in \mathbb{R}^V_{\geq 0}$, terminals $T \subseteq V$, root $r \in V$.
task: Find a minimum-cost subset $F \subseteq V$, such that r is connected to each terminal in $(F, E(F))$.

Arc Weighted Directed Steiner Tree is the corresponding problem, where the costs are on the arcs of the graph. A *tree decomposition* of a graph $G = (V, E)$ is a tree T as follows. Each node $x \in V(T)$ of T has a *bag* $B_x \subseteq V$ of vertices of G such that the following properties hold.

- $\bigcup_{x \in V(T)} B_x = V$.
- If B_x and B_y both contain a vertex $v \in V$, then the bags of all nodes of T in the path between x and y contain v as well. Equivalently, the tree nodes containing vertex v form a connected subtree of T.
- For each edge vw in G there is some bag that contains both v and w. That is, for vertices adjacent in G, the corresponding subtrees have a node in common.

The *width* of a tree decomposition is the size of its largest bag minus one. The *treewidth* $tw(G)$ of G is the minimum width among all possible tree decompositions of G.

To solve the Node Weighted Directed Steiner Tree on acyclic digraphs, we use a simple dynamic-programming algorithm over the tree decomposition of the underlying undirected graph of the input digraph D with n vertices.

▶ **Theorem 13.** Node Weighted Directed Steiner Tree *on acyclic digraphs can be solved in time* $O(5^w \cdot w \cdot n)$ *if a tree decomposition of* $\mathcal{U}(D)$ *of width* w *is provided.*

Note that an optimal tree-decomposition of a graph G can be computed in time $O(2^{O(tw(G)^3)} \cdot n)$ by Bodlaender's famous theorem [7]. Our algorithm intuitively works in the following way and is similar to the dynamic programming algorithm for Dominating Set (see, e.g., [13, Section 7.3.2]). We interpret a solution to Node Weighted Directed Steiner Tree as follows: each vertex of D may be *active* or not. Each active vertex needs a predecessor that is also active, unless it is the root. The cost to activate a vertex is given

by the cost function of the Node Weighted Directed Steiner Tree instance. Starting with all terminals active, it is easy to see that Node Weighted Directed Steiner Tree on acyclic graphs is equivalent to the problem of finding a minimum cost active vertex set satisfying the above conditions. We compute an optimal solution in a bottom-up fashion using a so-called *nice* tree decomposition of the input graph.

By a simple reduction, we also obtain an FPT-time algorithm for Arc Weighted Directed Steiner Tree on acyclic digraphs. We just subdivide each arc and assign the cost of the arc to the corresponding new vertex. Each old vertex receives cost zero. This transformation does not increase the treewidth.

Furthermore, we can reduce Source Cover to Node Weighted Directed Steiner Tree by adding a new vertex r and connecting r to each source by an arc. The sources have cost one, while all other vertices have cost zero. The root vertex is r and the set of terminals is the set of sinks. Adding a single new vertex increases the treewidth by at most one. As a consequence of this reduction and Theorem 13, we obtain the following result.

▶ **Corollary 14.** Source Cover *can be solved in time* $O(5^w \cdot w \cdot n)$ *if a tree-decomposition of* $\mathcal{U}(D)$ *of width* w *is provided.*

By combining Theorem 8, Proposition 9, Corollary 14, and the observation that treewidth is monotone under taking minors, we have:

▶ **Corollary 15.** Robust k-Matching Augmentation *parameterized by the treewidth of the input graph is FPT.*

5 Weighted Robust Matching Augmentation

We first demonstrate that the edge-weighted version of Robust Matching Augmentation is substantially more involved than the unit-cost version. To this end, we show that the approximability of Weighted Robust Matching Augmentation essentially corresponds to the approximability of Directed Steiner Forest. The latter problem is defined as follows:

Directed Steiner Forest
instance: Directed graph $G = (V, A)$, k terminal pairs $(s_i, t_i)_{1 \leq i \leq k}$, costs $w \in \mathbf{Z}^A_{\geq 0}$.
task: Find a minimum-cost subgraph $G' \subseteq G$ such that for each $1 \leq i \leq k$, the vertex s_i is connected to t_i in G'.

▶ **Proposition 16.** *Let* n' *be the number of vertices of a* Weighted Robust Matching Augmentation *instance and* n *and* k *be the number of vertices and terminal pairs of a* Directed Steiner Forest *instance, respectively.*

A polynomial-time $f(n')$*-factor approximation algorithm for* Weighted Robust Matching Augmentation *implies a polynomial-time* $f(n+k)$*-factor approximation algorithm for* Directed Steiner Forest*. Furthermore, a polynomial-time* $f(n)$*-factor (resp.,* $f(k)$*-factor) approximation algorithm for* Directed Steiner Forest *implies a polynomial-time* $(f(n)+1)$*-factor (resp.,* $(f(k)+1)$*-factor) approximation algorithm for* Weighted Robust Matching Augmentation*.*

On the one hand, Proposition 16 implies an $(n^{1/2+\varepsilon}+1)$-factor approximation algorithm for Weighted Robust Matching Augmentation for every $\varepsilon > 0$, due to the results in [10, 18]. On the other hand, an algorithm achieving a guarantee of $n^{1/3}$ or better for Weighted Robust Matching Augmentation would imply a better approximation

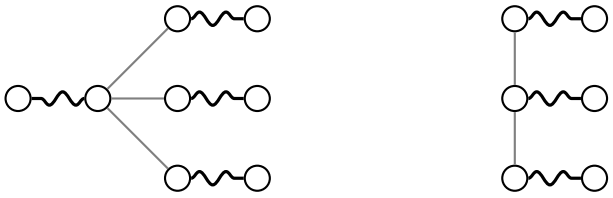


Figure 3 The graphs $K^*_{1,3}$ and P^*_3, each with its unique perfect matching.

algorithm for Directed Steiner Forest, as the number k of distinct terminal pairs is at most $O(n^2)$ and the current best approximation factor for Directed Steiner Forest in terms of n is $n^{2/3+\varepsilon}$ due to a result of Berman et al. [6]. Additionally, by a result of Halperin and Krauthgamer [23], Proposition 16 implies the following lower bound.

▶ **Corollary 17.** *For every $\varepsilon > 0$* Weighted Robust Matching Augmentation *does not admit a $\log^{2-\varepsilon}(n)$-factor approximation algorithm unless* NP $\subseteq$ ZTIME$(n^{\text{polylog}(n)})$.

Given this negative result we proceed to identify structural features that lead to polynomial-time algorithms for Weighted Robust Matching Augmentation. The main result of this section is a classification of the complexity of the problem Weighted Robust Matching Augmentation on minor-closed graph classes. In particular we show that the problem is NP-hard on a minor-closed class $\mathcal{G}$ of graphs if and only if $\mathcal{G}$ contains at least one of the two graph classes $\mathcal{K}^*$ and $\mathcal{P}^*$, which are defined as follows. Let $K_{1,r}$ be the star graph with r leaves and let P_r be the path on r vertices. For any graph H let H^* be the graph obtained by attaching a leaf to each vertex of H. Then $\mathcal{K}^* := \{K^*_{1,r} \mid r \in \mathbb{N}\}$ and $\mathcal{P}^* := \{P^*_r \mid r \in \mathbb{N}\}$. Note that each graph in $\mathcal{K}^*$ and $\mathcal{P}^*$ has a unique perfect matching. See Figure 3 for an illustration of the graphs $K^*_{1,3}$ and P^*_3.

▶ **Lemma 18.** Weighted Robust Matching Augmentation *is* NP*-hard on each of the classes $\mathcal{K}^*$ and $\mathcal{P}^*$.*

We complement Lemma 18 by showing that Weighted Robust Matching Augmentation on a class $\mathcal{G}$ of graphs admits a polynomial-time algorithm if $\mathcal{G}$ contains neither $\mathcal{K}^*$ nor $\mathcal{P}^*$.

▶ **Theorem 19.** *Let $\mathcal{G}$ be a class of connected graphs that is closed under connected minors. Then* Weighted Robust Matching Augmentation *on $\mathcal{G}$ admits a polynomial-time algorithm if and only if there is some $r \in \mathbb{N}$ such that $\mathcal{G}$ contains neither the graph $K^*_{1,r}$ nor P^*_r. The only if part holds under the assumption that* P $\neq$ NP.

In order to prove Lemma 18, we first show that Weighted Robust Matching Augmentation is NP-hard for graphs consisting only of a perfect matching by a reduction from Robust Matching Augmentation. The hardness of Weighted Robust Matching Augmentation on $\mathcal{K}^*$ and $\mathcal{P}^*$ follows from this result.

Before we give the proof of Theorem 19, we need the following key lemma. The polynomial-time algorithm described in the proof of the lemma uses the fact that Directed Steiner Forest can be solved in polynomial time if the number of terminal pairs is constant [17].

▶ **Lemma 20.** *Let $r \in \mathbb{N}$ be constant and let $\mathcal{T}$ be a class of perfectly matchable trees, each with at most r leaves. Then* Weighted Robust Matching Augmentation *on $\mathcal{T}$ admits a polynomial-time algorithm.*

We remark that the running time of the algorithm given in Lemma 20 is slicewise polynomial in the number of leaves of the input graph. We can now state the proof of our main result.

Proof of Theorem 19. According to Lemma 18, Weighted Robust Matching Augmentation is NP-hard if $\mathcal{G}$ completely contains the class $\mathcal{K} = \{K^*_{1,r} \mid r \in \mathbb{N}\}$ or the class $\mathcal{P} = \{P^*_r \mid r \in \mathbb{N}\}$. Assuming $\mathsf{P} \neq \mathsf{NP}$, this proves the *only if* statement of the theorem.

To see the *if* statement, let us consider $r \in \mathbb{N}$ such that $\mathcal{G}$ does not contain $K^*_{1,r}$ or P^*_r. First we will reduce the problem to the case when $\mathcal{G}$ contains only trees. For this, let $\mathcal{T}$ be the class of all trees in $\mathcal{G}$ that admit a perfect matching.

▷ Claim 1. There is a polynomial time reduction of Weighted Robust Matching Augmentation on $\mathcal{G}$ to Weighted Robust Matching Augmentation on $\mathcal{T}$.

The key idea for the proof is to define an equivalent instance on an arbitrary tree of G on an adapted cost function. We may hence restrict our attention to Weighted Robust Matching Augmentation on the class $\mathcal{T}$. As the next claim shows, the relevant trees contained in $\mathcal{T}$ have a bounded number of leaves.

▷ Claim 2. There is some number $f(r)$ depending only on r such that every tree in $\mathcal{T}$ has at most $f(r)$ many leaves.

According to the above claims, there is a polynomial reduction of Weighted Robust Matching Augmentation on $\mathcal{G}$ to Weighted Robust Matching Augmentation on a class of trees with a bounded number of leaves. Hence, Lemma 20 implies that Weighted Robust Matching Augmentation on $\mathcal{G}$ can be solved in polynomial time. ◀

6 Conclusion

We presented algorithms for the task of securing matchings of a graph against the failure of a single edge. For this, we established a connection to the classical strong connectivity augmentation problem. Not surprisingly, the unit weight case is more accessible, and we were able to give a $\log_2 n$-factor approximation algorithm, as well as polynomial-time algorithms for graphs of bounded treewidth and chordal-bipartite graphs. For general non-negative weights, we showed a close relation to Directed Steiner Forest in terms of approximability and gave a dichotomy theorem characterizing minor-closed graph classes which allow a polynomial-time algorithm.

In our opinion, the case of a single edge failure is well understood now and so one might go for the case of a constant number of edge failures next. Let us remark that if the number of edge failures is a part of the input, even checking feasibility is NP-hard [14, 25].

References

1 David Adjiashvili. Beating Approximation Factor Two for Weighted Tree Augmentation with Bounded Costs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2384–2399. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.157.

2 David Adjiashvili, Viktor Bindewald, and Dennis Michaels. Robust Assignments via Ear Decompositions and Randomized Rounding. In *43rd International Colloquium on Automata, Languages, and Programming*, volume 55, pages 71:1–71:14, Dagstuhl, Germany, 2016. doi:10.4230/LIPIcs.ICALP.2016.71.

3 David Adjiashvili, Sebastian Stiller, and Rico Zenklusen. Bulk-robust combinatorial optimization. *Mathematical Programming*, 149(1-2):361–390, 2015. doi:10.1007/s10107-014-0760-6.

4 David Adjiashvili and Rico Zenklusen. An $s-t$ connection problem with adaptability. *Discrete Applied Mathematics*, 159(8):695–705, 2011. doi:10.1016/j.dam.2010.12.018.

5 Kristóf Bérczi, Satoru Iwata, Jun Kato, and Yutaro Yamaguchi. Making Bipartite Graphs DM-Irreducible. *SIAM Journal on Discrete Mathematics*, 32:560–590, 2018. doi:10.1137/16M1106717.

6 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and Directed Steiner Forest. *Information and Computation*, 222:93–107, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011). doi:10.1016/j.ic.2012.10.007.

7 Hans L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

8 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

9 Robert C. Brigham, Frank Harary, Elizabeth C. Violin, and Jay Yellen. Perfect-matching Preclusion. *Congressus Numerantium*, 174:185–192, 2005.

10 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set Connectivity Problems in Undirected Graphs and the Directed Steiner Network Problem. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 532–541. Society for Industrial and Applied Mathematics, 2008.

11 Eddie Cheng and László Lipták. Matching preclusion for some interconnection networks. *Networks*, 50(2):173–180, 2007. doi:10.1002/net.20187.

12 Joseph Cheriyan, András Sebő, and Zoltán Szigeti. Improving on the 1.5-Approximation of a Smallest 2-Edge Connected Spanning Subgraph. *SIAM Journal on Discrete Mathematics*, 14(2):170–180, 2001. doi:10.1137/S0895480199362071.

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

14 Mitre C. Dourado, Dirk Meierling, Lucia D. Penso, Dieter Rautenbach, Fabio Protti, and Aline Ribeiro de Almeida. Robust recoverable perfect matchings. *Networks*, 66(3):210–213, 2015. doi:10.1002/net.21624.

15 Andrew L Dulmage and Nathan S Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10(4):516–534, 1958.

16 Kapali P. Eswaran and Robert E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976. doi:10.1137/0205044.

17 Jon Feldman and Matthias Ruhl. The Directed Steiner Network Problem is Tractable for a Constant Number of Terminals. *SIAM Journal on Computing*, 36(2):543–561, 2006. doi:10.1137/S0097539704441241.

18 Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved Approximating Algorithms for Directed Steiner Forest. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 922–931. Society for Industrial and Applied Mathematics, 2009.

19 Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating Weighted Tree Augmentation via Chvátal-Gomory Cuts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 817–831, 2018. doi:10.1137/1.9781611975031.53.

20 András Frank and Tibor Jordán. Graph Connectivity Augmentation. In *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*, chapter 14, pages 313–346. CRC Press, 2015.

21 Greg N Frederickson and Joseph Ja'Ja'. Approximation Algorithms for Several Graph Augmentation Problems. *SIAM Journal on Computing*, 10(2):270–283, 1981. doi:10.1137/0210019.

22 Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest k-edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009. doi:10.1002/net.20289.

23 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 585–594, 2003. doi:10.1145/780542.780628.

24 Alan J. Hoffman, Anthonius W. J. Kolen, and Michel Sakarovitch. Totally-Balanced and Greedy Matrices. *SIAM Journal on Algebraic Discrete Methods*, 6(4):721–730, 1985. doi:10.1137/0606070.

25 Mathieu Lacroix, A. Ridha Mahjoub, Sébastien Martin, and Christophe Picouleau. On the NP-completeness of the perfect matching free subgraph problem. *Theoretical Computer Science*, 423:25–29, 2012. doi:10.1016/j.tcs.2011.12.065.

26 Laurence A. Wolsey and George L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1999.

A Deterministic Polynomial Kernel for Odd Cycle Transversal and Vertex Multiway Cut in Planar Graphs

Bart M. P. Jansen
Eindhoven University of Technology
b.m.p.jansen@tue.nl

Marcin Pilipczuk
Institute of Informatics, University of Warsaw
malcin@mimuw.edu.pl

Erik Jan van Leeuwen
Department of Information & Computing Sciences, Utrecht University
e.j.vanleeuwen@uu.nl

Abstract

We show that Odd Cycle Transversal and Vertex Multiway Cut admit deterministic polynomial kernels when restricted to planar graphs and parameterized by the solution size. This answers a question of Saurabh. On the way to these results, we provide an efficient sparsification routine in the flavor of the sparsification routine used for the Steiner Tree problem in planar graphs (FOCS 2014). It differs from the previous work because it preserves the existence of low-cost subgraphs that are not necessarily Steiner trees in the original plane graph, but structures that turn into (supergraphs of) Steiner trees after adding all edges between pairs of vertices that lie on a common face. We also show connections between Vertex Multiway Cut and the Vertex Planarization problem, where the existence of a polynomial kernel remains an important open problem.

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

Keywords and phrases planar graphs, kernelization, odd cycle transversal, multiway cut

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.39

Related Version A full version of the paper is available at [18], `http://arxiv.org/abs/1810.01136`.

Funding *Bart M. P. Jansen*: Supported by NWO Gravitation grant "Networks".
Marcin Pilipczuk: Supported by the "Recent trends in kernelization: theory and experimental evaluation" project, carried out within the Homing programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

1 Introduction

Kernelization provides a rigorous framework within the paradigm of parameterized complexity to analyze preprocessing routines for various combinatorial problems. A *kernel* of size g for a parameterized problem Π and a computable function g is a polynomial-time algorithm that reduces an input instance x with parameter k of problem Π to an equivalent one with size and parameter value bounded by $g(k)$. Of particular importance are *polynomial kernels*, where the function g is required to be a polynomial, that are interpreted as theoretical tractability of preprocessing for the considered problem Π. Since a kernel (of any size) for a decidable problem implies fixed-parameter tractability (FPT) of the problem at hand, the question whether a *polynomial* kernel exists became a "standard" tractability question one asks about a problem already known to be FPT, and serves as a further finer-grained distinction criterion between FPT problems.

In the recent years, a number of kernelization techniques emerged, including the bidimensionality framework for sparse graph classes [10] and the use of representative sets for graph separation problems [20]. On the hardness side, a lower bound framework against polynomial kernels has been developed and successfully applied to a multitude of problems [1, 5, 7, 11]. For more on kernelization, we refer to the survey [22] for background and to the appropriate chapters of the textbook [3] for basic definitions and examples.

For this work, of particular importance are polynomial kernels for graph separation problems. The framework for such kernels developed by Kratsch and Wahlström in [20, 21], relies on the notion of *representative sets* in linear matroids, especially in gammoids. Among other results, the framework provided a polynomial kernel for Odd Cycle Transversal and for Multiway Cut with a constant number of terminals. However, all kernels for graph separation problems based on representative sets are randomized, due to the randomized nature of all known polynomial-time algorithms that obtain a linear representation of a gammoid. As a corollary, all such kernels have exponentially small probability of turning an input yes-instance into a no-instance.

The question of *deterministic* polynomial kernels for the cut problems that have *randomized* kernels due to the representative sets framework remains widely open. Saket Saurabh, at the open problem session during the Recent Advances in Parameterized Complexity school (Dec 2017, Tel Aviv) [27], asked whether a deterministic polynomial kernel for Odd Cycle Transversal exists when the input graph is planar. In this paper, we answer this question affirmatively, and prove an analogous result for the Multiway Cut problem.

▶ **Theorem 1.1.** Odd Cycle Transversal *and* Vertex Multiway Cut*, when restricted to planar graphs and parameterized by the solution size, admit deterministic polynomial kernels.*

Recall that the Odd Cycle Transversal problem, given a graph G and an integer k, asks for a set $X \subseteq V(G)$ of size at most k such that $G \setminus X$ is bipartite. For the Multiway Cut problem, we consider the Vertex Multiway Cut variant where, given a graph G, a set of terminals $T \subseteq V(G)$, and an integer k, we ask for a set $X \subseteq V(G) \setminus T$ of size at most k such that every connected component of $G \setminus X$ contains at most one terminal. In other words, we focus on the vertex-deletion variant of Multiway Cut with undeletable terminals. In both cases, the allowed deletion budget, k, is our parameter. (A deterministic polynomial kernel for Edge Multiway Cut in planar graphs is known [25, Theorem 1.4].)

Note that in general graphs, Vertex Multiway Cut admits a randomized polynomial kernel with $\mathcal{O}(k^{|T|+1})$ terminals [20], and whether one can remove the dependency on $|T|$ from the exponent is a major open question in the area. Theorem 1.1 answers this question affirmatively in the special case of planar graphs.

Our motivation stems not only from the aforementioned question of Saurabh [27], but also from a second, more challenging question of a polynomial kernel for the Vertex Planarization problem. Here, given a graph G and an integer k, one asks for a set $X \subseteq V(G)$ of size at most k such that $G \setminus X$ is planar. For this problem, an involved $2^{\mathcal{O}(k \log k)} \cdot n$-time fixed-parameter algorithm is known [17], culminating a longer line of research [17, 19, 23]. The question of a polynomial kernel for the problem has not only been posed by Saurabh during the same open problem session [27], but also comes out naturally in another line of research concerning vertex-deletion problems to minor-closed graph classes.

Consider a minor-closed graph class $\mathcal{G}$. By the celebrated Robertson-Seymour theorem, the list of minimal forbidden minors $\mathcal{F}$ of $\mathcal{G}$ is finite, i.e., there is a finite set $\mathcal{F}$ of graphs such that a graph G belongs to $\mathcal{G}$ if and only if G does not contain any graph from $\mathcal{F}$ as a minor. The $\mathcal{F}$-Deletion problem, given a graph G and an integer k, asks to find a set

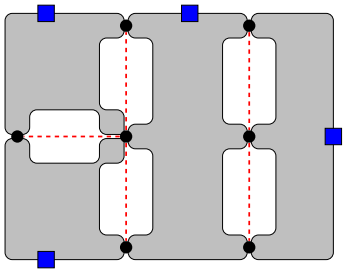


Figure 1 When all terminals (blue squares) lie on the infinite face, a solution to Vertex Multiway Cut (black circles) is a Steiner forest (red dashed connections) in the overlay graph.

$X \subseteq V(G)$ of size at most k such that $G \setminus X$ has no minor belonging to $\mathcal{F}$, i.e., $G \setminus X \in \mathcal{G}$. If $\mathcal{F}$ contains a planar graph or, equivalently, $\mathcal{G}$ has bounded treewidth, then the parameterized and kernelization complexity of the $\mathcal{F}$-Deletion problem is well understood [9]. However, our knowledge is very partial in the other case, when $\mathcal{G}$ contains all planar graphs. The understanding of this general problem has been laid out as one of the future research directions in a monograph of Downey and Fellows [6]. The simplest not fully understood case is when $\mathcal{G}$ is exactly the set of planar graphs, that is, $\mathcal{F} = \{K_{3,3}, K_5\}$, and the $\mathcal{F}$-Deletion becomes the Vertex Planarization problem. The question of a polynomial kernel or a $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$-time FPT algorithm for Vertex Planarization remains open [13, 27].

In Section 6, we observe that there is a simple polynomial-time reduction from Planar Vertex Multiway Cut to Vertex Planarization that keeps the parameter k unchanged. If Vertex Planarization would admit a polynomial kernel, then our reduction would transfer the polynomial kernel back to Planar Vertex Multiway Cut. In the presence of Theorem 1.1, such an implication is trivial, but the reduction itself serves as a motivation: a polynomial kernel for Planar Vertex Multiway Cut should be easier than for Vertex Planarization, and one should begin with the first before proceeding to the latter. Furthermore, we believe the techniques developed in this work can be of use for the more general Vertex Planarization case.

Techniques On the technical side, our starting point is the toolbox of [25] that provides a polynomial kernel for Steiner Tree in planar graphs, parameterized by the number of edges of the solution. The main technical result of [25] is a sparsification routine that, given a connected plane graph G with infinite face surrounded by a simple cycle ∂G, provides a subgraph of G of size polynomial in the length of ∂G that, for every $A \subseteq V(\partial G)$, preserves an optimal Steiner tree connecting A.

Both Odd Cycle Transversal and Vertex Multiway Cut in a plane graph G translate into Steiner forest-like questions in the *overlay graph* $\mathcal{L}(G)$ of G: a supergraph of G that has a vertex v_f for every face of G, adjacent to every vertex of G incident with f. To see this, consider a special case of Planar Vertex Multiway Cut where all terminals lie on the infinite face of the input embedded graph. Then, an optimal solution is a Steiner forest between some tuples of vertices on the outer face lying between the terminals, cf. Figure 1. Following [25], this suggest the following approach to kernelization of vertex-deletion cut problems in planar graphs:

1. By problem-specific reductions, reduce to the case of a graph of bounded radial diameter.
2. Using the diameter assumption, find a tree in the overlay graph that has size bounded polynomially in the solution size, and that spans all "important" objects in the graph (e.g., neighbors of the terminals in the case of Multiway Cut or odd faces in the case of Odd Cycle Transversal).

3. Cut the graph open along the tree. Using the Steiner forest-like structure of the problem at hand, argue that an optimal solution becomes an optimal Steiner forest for some choice of tuples of terminals on the outer face of the cut-open graph.
4. Sparsify the cut-open graph with a generic sparsification routine that preserves optimal Steiner forests, glue the resulting graph back, and return it as a kernel.

However, contrary to the Steiner tree problem [25], these Steiner forest-like questions optimize a different cost function than merely number of edges, namely *the number of vertices of* G, with the "face" vertices $v_f \in V(\mathcal{L}(G)) \setminus V(G)$ being for free. This cost function is closely related to (half of) the number of edges in case of paths and trees with constant number of leaves, but may diverge significantly in case of trees with high-degree vertices.

For this reason, we need an analog of the main technical sparsification routine of [25] suited for our cost function. To this end, we re-use most of the intermediate results of [25], changing significantly only the final divide&conquer argument. We provide a statement and an overview of the proof of such routine in Section 3. A full proof can be found on arXiv [18].

The application of the obtained sparsification routine to the case of Odd Cycle Transversal, presented in Section 4, follows the phrasing of the problem as a T-join-like problem in the overlay graph due to Fiorini et al [8]. For the sake of reducing the number of odd faces, we adapt the arguments of Suchý [28] for Steiner tree.

The arguments for Vertex Multiway Cut are somewhat more involved and sketched in Section 5. A full version can be found on arXiv [18]. Here, we first use known LP-based rules [4, 12, 14, 26] to reduce the number of terminals and neighbors of terminals to $\mathcal{O}(k)$ and then use an argument based on outerplanarity layers to reduce the diameter.

2 Preliminaries

A finite undirected graph G consists of a vertex set $V(G)$ and edge set $E(G) \subseteq \binom{V(G)}{2}$. We denote the open neighborhood of a vertex v in G by $N_G(v)$. For a vertex set $S \subseteq V(G)$ we define its open neighborhood as $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. For all standard but undefined here terms related to planar graph we refer to [25].

For vertex subsets X, Y of a graph G, we define an (X, Y)-cut as a vertex set $Z \subseteq V(G) \setminus (X \cup Y)$ such that no connected component of $G \setminus Z$ contains both a vertex of X and a vertex of Y. An (X, Y) cut Z is *minimal* if no proper subset of Z is an (X, Y)-cut, and *minimum* if it has minimum possible size.

2.1 Planar graphs

In a connected embedded planar (i.e. plane) graph G, the *boundary walk* of a face f is the unique closed walk in G obtained by going along the face in counter-clockwise direction. Note that a single vertex can appear multiple times on the boundary walk of f and an edge can appear twice if it is a bridge. We denote the number of edges of this walk by $|f|$; note that bridges are counted twice in this definition. The *parity* of a face f is the parity of $|f|$. Then a face is *odd (even)* if its parity is odd (even). The boundary walk of the outer face of G is called the *outer face walk* and denoted ∂G.

We define the *radial distance* in plane graphs, based on a measure that allows to hop between vertices incident on a common face in a single step. Formally speaking, a *radial path* between vertices p and q in a plane graph G is a sequence of vertices $(p = v_0, v_1, \ldots, v_\ell = q)$ such that for each $i \in [\ell]$, the vertices v_{i-1} and v_i are incident on a common face. The *length* of the radial path equals ℓ, so that a trivial radial path from v to itself has length 0. The *radial distance* in plane graph G between p and q, denoted $d^{\mathcal{R}}_G(u, v)$, is defined as the minimum length of a radial pq-path.

For a plane graph G, let $F(G)$ denote the set of faces of G. For a plane (multi)graph G, an *overlay graph* G' of G is a graph with vertex set $V(G) \cup F(G)$ obtained from G as follows. For each face $f \in F(G)$, draw a vertex with identity f in the interior of f. For each connected component C of edges incident on the face f, traverse the boundary walk of C starting at an arbitrary vertex. Every time a vertex v is visited by the boundary walk, draw a new edge between v and the vertex representing f, without crossing previously drawn edges. Doing this independently for all faces of G yields an overlay graph G'. Observe that an overlay graph may have multiple edges between some $f \in F(G)$ and $v \in V(G)$, which occurs for example when v is incident on a bridge that lies on f. The resulting plane multigraph G' is in general not unique, due to different homotopies for how edge bundles may be routed around different connected components inside a face. For our purposes, these distinctions are never important. We therefore write $\mathcal{L}(G)$ to denote an arbitrary fixed overlay graph of G. Observe that $F(G)$ forms an independent set in $\mathcal{L}(G)$.

Apart from the overlay graph, we will also use the related notion of *radial graph* (also known as face-vertex incidence graph). A *radial graph* of a connected plane graph G is a plane multigraph $\mathcal{R}(G)$ obtained from $\mathcal{L}(G)$ by removing all edges with both endpoints in $V(G)$. Hence a radial graph of G is bipartite with vertex set $V(G) \cup F(G)$, where vertices are connected to the representations of their incident faces. From these definitions it follows that $\mathcal{L}(G)$ is the union of G and $\mathcal{R}(G)$, which explains the terminology.

We need also the following simple but useful lemma.

▶ **Lemma 2.1.** *Let G be a connected graph, let $T \subseteq V(G)$ and assume that for each vertex $v \in V(G)$, there is a terminal $t \in T$ that can reach v by a path of at most K edges. Then G contains a Steiner tree of at most $(2K+1)(|T|-1)$ edges on terminal set T, which can be computed in linear time.*

Proof. Observe that there exists a spanning forest in G where each tree is rooted at a vertex of T, and each tree has depth at most K. Such a spanning forest can be computed in linear time by a breadth-first search in G, initializing the BFS-queue to contain all vertices of T with a distance label of 0. Consider the graph H obtained from G by contracting each tree into the terminal forming its root. Since G is connected, H is connected as well. An edge t_1t_2 between two terminals in H implies that in G there is a vertex in the tree of t_1 adjacent to a vertex of the tree of t_2. So for each edge in H, there is a path between the corresponding terminals in G consisting of at most $2K+1$ edges.

Compute an arbitrary spanning tree of the graph H, which has $|T|-1$ edges since H has $|T|$ vertices. As each edge of the tree expands to a path in G between the corresponding terminals of length at most $2K+1$, it follows that G has a connected subgraph F of at most $(2K+1)(|T|-1)$ edges that spans all terminals T. To eliminate potential cycles in F, take a spanning subtree of F as the desired Steiner tree. ◀

▶ **Lemma 2.2** ([16, Lemma 1])**.** *Let G be a planar bipartite graph with bipartition $V(G) = X \uplus Y$ for $X \neq \emptyset$. If all distinct $u, v \in Y$ satisfy $N_G(u) \not\subseteq N_G(v)$, then $|Y| \leq 5|X|$.*

3 Sparsification

A *plane partitioned graph* is an undirected multigraph G, together with a fixed embedding in the plane and a fixed partition $V(G) = \mathbf{A}(G) \uplus \mathbf{B}(G)$ where $\mathbf{A}(G)$ is an independent set. Consider a subgraph H of a plane partitioned graph G. The *cost* of H is defined as $\mathrm{cost}(H) := |V(H) \cap \mathbf{B}(G)|$, that is, we pay for each vertex of H in the part $\mathbf{B}(G)$. We say that H *connects* a subset $A \subseteq V(G)$ if $A \subseteq V(H)$ and A is contained in a single connected component of H.

Our main sparsification routine is the following.

▶ **Theorem 3.1.** *Given a connected plane partitioned graph G, one can in time $|\partial G|^{\mathcal{O}(1)} \cdot \mathcal{O}(|G|)$ find a subgraph $\widehat{G}$ in G, with the following properties:*

1. *$\widehat{G}$ contains all edges and vertices of ∂G,*
2. *$\widehat{G}$ contains $\mathcal{O}(|\partial G|^{212})$ edges,*
3. *for every set $A \subseteq V(\partial G)$ there exists a subgraph H of $\widehat{G}$ that connects A and has minimum possible cost among all subgraphs of G that connect A.*

In the subsequent sections, given a connected plane graph G, we will apply Theorem 3.1 to a graph G' that is either the overlay graph of G without the vertex corresponding to the outer face, or the radial graph of G. In either case, $\mathbf{A}(G') = V(G') \setminus V(G)$ is the set of face vertices and $\mathbf{B}(G') = V(G)$, i.e., we pay for each "real" vertex, not a face one. If the studied vertex-deletion graph separation problem in G turns into some Steiner problem in G', then we may hope to apply the sparsification routine of Theorem 3.1.

After this brief explanation of the motivation of the statement of Theorem 3.1, we proceed with an overview of its proof. We closely follow the divide&conquer approach of the polynomial kernel for Steiner Tree in planar graphs [25].

We adopt the notation of (strictly) enclosing from [25]. For a closed curve γ on a plane, a point $p \notin \gamma$ is strictly enclosed by γ if γ is not continuously retractable to a single point in the plane punctured at p. A point p is *enclosed* by γ if it is strictly enclosed or lies on γ. The notion of (strict) enclosure naturally extends to vertices, edges, and faces of a plane graph G being (strictly) enclosed by γ; here a face (an edge) is strictly enclosed by γ if every interior point of a face (every point on an edge except for the endpoints, respectively) is strictly enclosed. We also extend this notion to (strict) enclosure by a closed walk W in a plane graph G in a natural manner. Note that this corresponds to the natural notion of (strict) enclosure if W is a cycle or, more generally, a closed walk without self-intersections.

We start with restricting the setting to G being bipartite and ∂G being a simple cycle. Theorem 3.1 follows from Lemma 3.2 by simple manipulations, and its proof can be found on arXiv [18].

▶ **Lemma 3.2.** *The statement of Theorem 3.1 is true in the restricted setting with G being a connected bipartite simple graph with ∂G being a simple cycle and $\mathbf{A}(G)$ being one of the bipartite color classes (so that $\mathbf{B}(G)$ is an independent set as well).*

We now sketch the proof of Lemma 3.2. The full proof can be found on arXiv [18].

First observe that the statement of Lemma 3.2 is well suited for a recursive divide&conquer algorithm. As long as $|\partial G|$ is large enough, we can identify a subgraph S of G such that:

1. The number of edges of S is $\mathcal{O}(|\partial G|)$;
2. For every set $A \subseteq V(\partial G)$ there exists a subgraph H of G that connects A, has minimum possible cost among all subgraphs of G that connect A, and for every finite face f of $S \cup \partial G$, if G_f is the subgraph of G consisting of the edges and vertices embedded within the closure of f, then one of the following holds:
   a. $|\partial G_f| \leq (1-\delta)|\partial G|$ for some universal constant $\delta > 0$;
   b. H does not contain any vertex of degree more than 2 that is strictly inside f.

Similarly as in the case of [25], we show that such a subgraph S is good for recursion. First, we insert S into the constructed sparsifier $\widehat{G}$. Second, we recurse on G_f for every finite face f of $S \cup \partial G$ that satisfies Point 2a. Third, for every other finite face f (i.e., one satisfying Point 2b), we insert into $\widehat{G}$ a naive shortest-paths sparsifier: for every two vertices $u_1, u_2 \in V(\partial G_f)$, we insert into $\widehat{G}$ a minimum-cost path between u_1 and u_2 in G_f.

Property 1 together with the multiplicative progress on $|\partial G|$ in Point 2a ensure that the final size of $\widehat{G}$ is polynomial in $|\partial G|$, with the exponent of the polynomial bound depending on δ and the constant hidden in the big-$\mathcal{O}$ notation in Property 1.

The main steps of constructing S are the same as in [25]. First, we try minimum-size (i.e., with minimum number of edges, as opposed to minimum-cost) Steiner trees for a constant number of terminals on ∂G. If no such trees are found, the main technical result of [25] shows that one can identify a cycle C in G of length $\mathcal{O}(|\partial G|)$ with the guarantee that for any choice of $A \subseteq V(\partial G)$, there exists a minimum-size Steiner tree connecting A that does not contain any Steiner point strictly inside C. In [25] such a cycle is used to construct a desired subgraph S with the inside of C being a face satisfying the Steiner tree analog of Point 2b. In the case of Lemma 3.2, we need to perform some extra work here to show that – by some shortcutting tricks and adding some slack to the constants – one can construct such a cycle C' with the guarantee that the face f inside C' satisfies exactly the statement of Point 2b: that is, no "Steiner points" with regards to minimum-*cost* trees, not minimum-*size* ones.

In other words, the extra work is needed to at some point switch from "minimum-size" subgraphs (treated by [25]) to "minimum-cost" ones (being the main focus of Lemma 3.2). In our proof, we do it as late as possible, trying to re-use as much of the technical details of [25] as possible. Observe that for a path H in G, the cost of H equals $|E(H)|/2$ up to an additive $\pm\frac{1}{2}$ error. Similarly, for a tree H with a constant number of leaves, the cost of H is $|E(H)|/2$ up to an additive error bounded by a constant. Hence, as long as we focus on paths and trees with bounded number of leaves, the "size" and "cost" measures are roughly equivalent. However, if a tree H in G contains a high-degree vertex $v \in \mathbf{B}(G)$, the cost of H may be much smaller than half of the number of edges of H: a star with a center in $\mathbf{B}(G)$ has cost one and arbitrary number of edges. For this reason, the final argument of the proof of Lemma 3.2 that constructs the aforementioned cycle C' using the toolbox of [25] needs to be performed with extra care (and some sacrifice on the constants, as compared to [25]).

4 Odd Cycle Transversal

To understand the Odd Cycle Transversal problem, we rely on the correspondence between odd cycle transversals and T-joins. This correspondence was originally developed by Hadlock [15] for the edge version of Odd Cycle Transversal on planar graphs; for the vertex version discussed here, we build on the work of Fiorini et al. [8]. Given a graph H and set $T \subseteq V(H)$, a *T-join* in H is a set $J \subseteq E(H)$ such that T equals the set of odd-degree vertices in the subgraph of H induced by J. It is known that a connected graph contains a T-join if and only if $|T|$ is even.

▶ **Lemma 4.1** ([8, Lemma 1.1]). *Let T be the set of odd faces of a connected plane graph G. Then $C \subseteq V(G)$ is an odd cycle transversal of G if and only if $\mathcal{R}(G)[C \cup F(G)]$ contains a T-join, that is, each connected component of $\mathcal{R}(G)[C \cup F(G)]$ contains an even number of vertices of T.*

This leads to the following problem :

Bipartite Steiner T-join **Parameter:** k

Input: A connected bipartite graph G, a fixed partition $V(G) = \mathbf{A}(G) \uplus \mathbf{B}(G)$, $T \subseteq \mathbf{A}(G)$, and an integer k.

Question: Does there exist a set $C \subseteq \mathbf{B}(G)$ of size at most k such that $G[C \cup \mathbf{A}(G)]$ contains a T-join, that is, each connected component of $G[C \cup \mathbf{A}(G)]$ contains an even number of vertices of T?

In particular, we are interested in the problem when G is a plane graph, which we call Plane Bipartite Steiner T-join. We call T the set of *terminals* of the instance; $\mathbf{A}(G) \setminus T$ is the set of *non-terminals*. We call $C \subseteq \mathbf{B}(G)$ a *solution* to an instance of Bipartite Steiner T-join if $|C| \leq k$ and $G[C \cup \mathbf{A}(G)]$ contains a T-join.

▶ **Lemma 4.2.** *If* Plane Bipartite Steiner T-join *has a polynomial kernel, then* Plane Odd Cycle Transversal *has a polynomial kernel.*

Proof. By Lemma 4.1, the answer to a plane instance (G, T, k) of Odd Cycle Transversal is equivalent to the answer of the Plane Bipartite Steiner T-join instance on the graph $\mathcal{R}(G)$, with the face vertices $F(G)$ taking the role of $\mathbf{A}$, $V(G)$ taking the role of $\mathbf{B}$, and $T \subseteq F(G)$ being the odd faces. So if Plane Bipartite Steiner T-join has a polynomial kernel, then an instance of Plane Odd Cycle Transversal can be compressed to size polynomial in k by transforming it into an instance of Plane Bipartite Steiner T-join and applying the kernel to it. Since Plane Bipartite Steiner T-join is in NP and Plane Odd Cycle Transversal is NP-hard, by standard arguments (cf. [2]) the T-join instance can be reduced back to an instance of the original problem of size polynomial in k, which forms the kernel. ◀

Below, we will give a polynomial kernel for Plane Bipartite Steiner T-join. Combined with Lemma 4.2, this implies a polynomial kernel for Plane Odd Cycle Transversal.

4.1 Reducing the number of terminals

Let $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ be an instance of Plane Bipartite Steiner T-join. As a first step, we show that the graph can be reduced so that there remain at most $6k^2$ terminals. To this end, we adapt the rules that Suchý [28] developed for Plane Steiner Tree parameterized by the number of Steiner vertices of the solution tree. Each of the rules is applied exhaustively before a next rule will be applied.

▶ **Observation 4.3.** *Let C be a solution for the instance. Then each vertex of T has a neighbor in C.*

This is the analogue of [28, Lemma 2] and is immediate from the bipartiteness of G.

▶ **Observation 4.4.** *If $k < 0$ or there is a connected component containing exactly one terminal $t \in T$, then we can safely answer NO.*

▶ **Lemma 4.5.** *Let $X \subseteq T$ be a maximal set such that $N_G(x) = N_G(y)$ for all $x, y \in X$. Remove all but $2 - (|X| \bmod 2)$ vertices of X from the graph and T. The resulting instance $(G', \mathbf{A}(G'), \mathbf{B}(G'), T', k)$ has a solution if and only if $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ has a solution.*

Proof. Let $Y \subseteq X$ be the set of remaining vertices of X. Observe that $|X| \equiv |Y| \pmod 2$ and that $|Y| \geq 1$. The equivalence is now immediate. ◀

▶ **Lemma 4.6.** *Let $u, v \in \mathbf{B}(G)$ and let $L = N_G(u) \cap N_G(v) \cap T$ with $L \neq \emptyset$. If a connected component X of $G \setminus (L \cup \{u, v\})$ exists that contains no terminals, then remove X from the graph. The resulting instance $(G', \mathbf{A}(G'), \mathbf{B}(G'), T, k)$ has a solution if and only if $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ has a solution.*

Proof. If $(G', \mathbf{A}(G'), \mathbf{B}(G'), T, k)$ has a solution C, then C is a solution for the instance $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$, as G' is an induced subgraph of G with the same terminal set.

Suppose that C is a minimal solution for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. We construct a solution for $(G', \mathbf{A}(G'), \mathbf{B}(G'), T, k)$ such that $C' \cap X = \emptyset$. Suppose that $C \cap X \neq \emptyset$; otherwise, let $C' = C$. Let $C' = (C \setminus X) \cup \{v\}$. In either case, $|C'| \leq |C| \leq k$. We claim that C' is still a solution for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. To this end, first consider $C \cup \{v\}$. All connected components of $G[C \cup \mathbf{A}(G)]$ that neighbor v will then be unified into a single connected component Z of $G[C \cup \{v\} \cup \mathbf{A}(G)]$. The parity of $|Z \cap T|$ is equal to the sum (mod 2) of the parities of $|Z' \cap T|$ of the connected components Z' of $G[C \cup \mathbf{A}(G)]$ neighboring v. Since these parities are 0, their sum is 0, and $|Z \cap T|$ is even. Now consider the connected component ZZ of $G[C' \cup \mathbf{A}(G)]$ that contains v. Clearly, $ZZ \cap T = Z \cap T$, because $X \cap T = \emptyset$ and any path in $G[C \cup \{v\} \cup \mathbf{A}(G)]$ that intersects X can be re-routed through v and the vertices of $L \subseteq \mathbf{A}(G)$. The claim follows, and thus the lemma as well. ◀

We now present the final two reduction rules. Each relies on the following operation.

▶ **Lemma 4.7.** *Let $v \in \mathbf{B}(G)$. Let G' be obtained from G by contracting all edges between v and its neighbors in G. Let v' be the resulting vertex, and let $\mathbf{A}(G')$ and $\mathbf{B}(G')$ be the resulting color classes, where $v' \in \mathbf{A}(G')$. Let T' be obtained from T by removing $N_G(v) \cap T$, and adding v' to T' if and only if $|N_G(v) \cap T| \equiv 1 \pmod 2$.*

- *If $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ has a solution C with $v \in C$, then $(G', \mathbf{A}(G'), \mathbf{B}(G'), T', k-1)$ has a solution;*
- *if $(G', \mathbf{A}(G'), \mathbf{B}(G'), T', k-1)$ has a solution, then $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ has a solution.*

Proof. Suppose there is a solution C to $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ such that $v \in C$. Then the vertices of $T \cap N_G(v)$ are in the same connected component Z of $G[C \cup \mathbf{A}(G)]$. Let $C' = C \setminus \{v\}$ and let Z' be obtained from Z by contracting all edges between v and $N_G(v)$. Then Z' is a connected component of $G'[C' \cup \mathbf{A}(G')]$. By the construction of T', Z' contains an even number of vertices of T'. Moreover, $|C'| = |C| - 1 \leq k - 1$. Hence, C' is a solution to $(G', \mathbf{A}(G'), \mathbf{B}(G'), T', k-1)$.

Suppose there is a solution C' to $(G', \mathbf{A}(G'), \mathbf{B}(G'), T', k-1)$. Let $C = C' \cup \{v\}$. Let Z' be the connected component of $G'[C' \cup \mathbf{A}(G')]$ that contains v', and let Z be obtained from Z' by adding $N_G[v]$ and removing v'. Then Z is a connected component of $G[C \cup \mathbf{A}(G)]$. Moreover, by the construction of T', Z contains an even number of vertices of T. Finally, $|C| = |C'| + 1 \leq k$. Hence, C is a solution to $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. ◀

▶ **Lemma 4.8.** *Let $u, v \in \mathbf{B}(G)$ and let $L = N_G(u) \cap N_G(v) \cap T$ with $L \neq \emptyset$. If a connected component X of $G \setminus (L \cup \{u, v\})$ exists for which all terminals in $X \cap T$ neighbor v and there is a solution C to the instance $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$, then there is a solution that contains v.*

Proof. Assume that $v \notin C$, or the lemma would already follow. Since the rule of Lemma 4.6 is inapplicable, there is a terminal in X. Moreover, no terminal in $X \cap T$ neighbors u, because any such terminal would be in L and thus not in X. Since every terminal has to have a neighbor in C, it follows that $C \cap X \neq \emptyset$. Therefore, $C' = (C \setminus X) \cup \{v\}$ is not larger than C. We claim that C' is still a solution. To this end, first consider $C \cup \{v\}$. All connected components of $G[C \cup \mathbf{A}(G)]$ that neighbor v will then be unified into a single connected component Z of $G[C \cup \{v\} \cup \mathbf{A}(G)]$. In particular, Z contains $X \cap T$. The parity of $|Z \cap T|$ is equal to the sum (mod 2) of the parities of $|Z' \cap T|$ of the connected components Z' of $G[C \cup \mathbf{A}(G)]$ that neighbor v. Since these parities are 0, their sum is 0, and $|Z \cap T|$ is even. Now consider the connected component ZZ of $G[C' \cup \mathbf{A}(G)]$ that contains v. Clearly, $ZZ \cap T = Z \cap T$, because any path in $G[C \cup \{v\} \cup \mathbf{A}(G)]$ that intersects X can be re-routed through v and the vertices of L. The claim follows, and thus the lemma as well. ◀

▶ **Lemma 4.9.** *If there is a vertex $v \in \mathbf{B}(G)$ adjacent to more than $6k$ terminals and there is a solution C to the instance $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$, then there is a solution that contains v.*

Proof. The proof is completely analogous to the proof of [28, Lemma 11]. If $v \in C$, then we are done. So assume that $v \notin C$. Let $B \subseteq C$ be the set of vertices in C adjacent to at least two terminals in $N_G(v)$. Given $b \in B$, let x, y be any two terminals in $N_G(b) \cap N_G(v)$ and consider the region R that is enclosed by the cycle x, b, y, v and that does not contain the outer face. If R does not contain any other terminal of $N_G(b) \cap N_G(v)$, then R is called the *(internal) eye* of x, b, y, v. The *support* of $b \in B$, denoted $\mathrm{supp}(b)$, is the set of vertices $a \in B$ such that a is contained inside an eye R of b, but not inside an eye of any $b' \in B \setminus \{b\}$ for which b' is inside R. The bound of $6k$ (instead of $5k$) ensures that the proof of [28, Lemma 16] can be modified (straightforwardly) to yield a vertex $b \in \mathbf{B}(G)$ adjacent to more than $2\,|\mathrm{supp}(b)| + 4$ vertices of T. The further arguments then imply the existence of a twin set in T of size at least 3, thus contradicting the exhaustive execution of the rule of Lemma 4.5. ◀

Lemma 4.8 and 4.9, when combined with Lemma 4.7, naturally lead to two reduction rules. After exhaustively applying all the reduction rules in this section, each vertex of $\mathbf{B}(G)$ neighbors at most $6k$ terminals.

▶ **Observation 4.10.** *If $|T| > 6k^2$, then we can safely answer NO.*

This rule is immediate from Observation 4.3 and the fact that any solution contains at most k vertices that are each adjacent to at most $6k$ terminals by Lemma 4.9.

4.2 Reducing the diameter and obtaining the kernel

We now reduce the diameter of the graph. Our arguments here are a generalization of the arguments of Fiorini et al. [8] in their FPT-algorithm for Plane Odd Cycle Transversal.

▶ **Lemma 4.11.** *Suppose there is a solution for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. Let C be a minimal solution. Then each vertex $v \in C$ has distance at most $k+1$ in $G[C \cup \mathbf{A}(G)]$ to a vertex of T.*

Proof. Suppose for sake of contradiction that $v \in C$ has distance at least $k+1$ to each vertex of T in $G[C \cup \mathbf{A}(G)]$. Since C is minimal, there are two connected components X and Y of $G[(C \setminus \{v\}) \cup \mathbf{A}(G)]$ with an odd number of terminals. Let $x \in X \cap T$ and $y \in Y \cap T$. Consider a shortest path in $G[C \cup \mathbf{A}(G)]$ from x to v. This path P is fully contained in $G[V(X) \cup \{v\}]$ and has length at least $k+1$. As P connects vertices on opposite sides of the bipartite graph, $|V(P) \cap C| \geq 1 + k/2$. Hence, $|V(X) \cap C| \geq k/2$. Similarly, $|V(Y) \cap C| \geq k/2$. Since X and Y are vertex disjoint, it follows that $|C| \geq 2k/2 + 1 > k$, a contradiction. ◀

▶ **Corollary 4.12.** *Suppose there is a solution for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. Let C be a minimal solution. Then every vertex of $C \cup T$ has distance at most $k+2$ to T in $G[C \cup \mathbf{A}(G)]$.*

▶ **Lemma 4.13.** *We can safely answer NO, or we can compute, in polynomial time, disjoint subgraphs $G_1, \ldots, G_\ell$ of G for some $\ell \leq k$ such that:*

1. *the graphs G_i jointly contain all terminals;*
2. *for each i and for each vertex $v \in V(G_i)$, there is a terminal $t \in T \cap V(G_i)$ that can reach v by a path of at most $k+2$ edges;*
3. *for any solution C for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$, $C \cap V(G_i)$ is a solution for $(G_i, \mathbf{A}(G_i), \mathbf{B}(G_i), T \cap V(G_i), k_i)$ for each i, where $k_i = |C \cap V(G_i)|$;*
4. *if $(G_i, \mathbf{A}(G_i), \mathbf{B}(G_i), T \cap V(G_i), k_i)$ has a solution for each i for some $k_1, \ldots, k_\ell \geq 0$ that sum up to at most k, then $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$ has a solution.*

Proof. For each terminal $t \in T$, let $B(t)$ be the set of all vertices within distance $k+2$ of t. Let $G_1, \ldots, G_\ell$ be the connected components of $G[\bigcup_{t \in T} B(t)]$. If $\ell > k$, then G has more than k terminals with disjoint neighborhoods in $\mathbf{B}(G)$, and we can safely answer NO. We now consider the properties set forth in the lemma statement:

1. True by construction and the definition of the function B.
2. True by construction and the definition of the function B.
3. True by construction, the definition of the function B, and Corollary 4.12.
4. We take the union C of the solutions C_i of the sub-instances. Note that the subgraphs G_i are disjoint and thus contain disjoint sets of terminals. Hence, any connected component of $G[C \cup \mathbf{A}(G)]$ that contains connected components of $G[C_i \cup \mathbf{A}(G)]$ for multiple i, still contains an even number of terminals.

This finishes the proof. ◀

Property 2 of Lemma 4.13 implies that each constructed subgraph G_i has diameter $\mathcal{O}(k \cdot |T \cap V(G_i)|)$, which is $\mathcal{O}(k^3)$ using Observation 4.10. The proof of Theorem 4.14 employs an additional argument to obtain a quadratic-size Steiner tree to cut open.

▶ **Theorem 4.14.** Plane Bipartite Steiner T-join *has a kernel of size* $\mathcal{O}(k^{425})$.

Proof. We first exhaustively apply the reduction rules of Subsection 4.1 until each vertex of $\mathbf{B}(G)$ neighbors at most $6k$ terminals. The rules can clearly be executed exhaustively in polynomial time. As per Observation 4.10, we may assume that $|T| \leq 6k^2$. Then we apply Lemma 4.13 and consider each of the ℓ subgraphs G_i separately. Let $T_i = T \cap V(G_i)$; note that $|T_i| \leq 6k^2$. Moreover, we can assume that $|T_i|$ is even, or we can safely answer NO.

We construct a small set $A_i \subseteq V(G_i)$ such that $G[A_i]$ is connected and contains T_i. Start by adding T_i to A_i. Then, we find a subset T_i' of T_i such that the sets $N_{G_i}(t)$ are pairwise disjoint for $t \in T_i'$ by the following iterative marking procedure: add any unmarked $t \in T_i$ to T_i' and then mark all terminals in $N_{G_i}(N_{G_i}(t))$. It follows from Observation 4.3 that $|T_i'| \leq k$, or we can safely answer NO. Now apply Lemma 2.1 to find a Steiner tree of at most $(2(k+2)+1)(|T_i'|-1)$ edges (and vertices) on T_i'. Add these vertices to A_i. Finally, for each $t \in T_i$, let $t' \in T_i'$ be a terminal such that $t \in N_{G_i}(N_{G_i}(t')) \cup \{t'\}$ and add an arbitrary vertex of $N(t) \cap N(t')$ to A_i. Then $|A_i| \leq 6k^2 + 6k^2 + (2k+5)|T_i'| = \mathcal{O}(k^2)$. Moreover, by construction, $G_i[A_i]$ is connected and contains T_i.

Let S_i be a spanning tree of $G_i[A_i]$. Note that S_i has size $\mathcal{O}(k^2)$ by the construction of A_i and contains T_i. We cut the plane open along S_i and make the resulting face the outer face. Let $\hat{G}_i$ denote the resulting plane graph. That is, we create a walk W_i on the edges of S_i that visits each edge of S_i exactly twice. This walk has $\mathcal{O}(k^2)$ edges. Then we duplicate the edges of S_i and duplicate each vertex v of S_i exactly $d_{S_i}(v) - 1$ times, where $d_{S_i}(v)$ is the degree of v in S_i. We can then create a face in the embedding that has W_i as boundary. Then we obtain $\hat{G}_i$ by creating an embedding in which this new face is the outer face. See Figure 2. This also yields a natural mapping π from $E(\hat{G}_i)$ to $E(G_i)$ and from $V(\hat{G}_i)$ to $V(G_i)$. Finally, we observe that the terminals T_i are all on the outer face of $\hat{G}_i$ and that $\hat{G}_i$ is a connected plane partitioned graph.

Now apply Theorem 3.1 to $\hat{G}_i$ and let $\tilde{G}_i$ be the resulting graph. Let $F_i = \pi(\tilde{G}_i)$. Note that $\tilde{G}_i$ has $\mathcal{O}(|\partial \hat{G}_i|^{212}) = \mathcal{O}(|W_i|^{212}) = \mathcal{O}(k^{424})$ edges, and thus so has F_i. Finally, let $F = \bigcup_{i=1}^{\ell} F_i$. Clearly, $|F| = \mathcal{O}(k^{425})$, as $\ell < k$. Also note that each of the reduction rules, the above marking procedures, and F itself can be computed in polynomial time.

We claim that $(F, \mathbf{A}(F), \mathbf{B}(F), T, k)$ is a kernel. Since F is a subgraph of G, it follows that if $(F, \mathbf{A}(F), \mathbf{B}(F), T, k)$ has a solution, then so does $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. Now let C be a minimum solution for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. Then $C_i = C \cap V(G_i)$ is a solution for

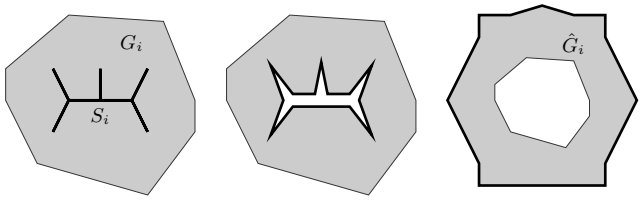


Figure 2 The process of cutting open the graph G_i along the tree S_i. Adapted from [24] with permission.

$(G_i, \mathbf{A}(G_i), \mathbf{B}(G_i), T \cap V(G_i), k_i)$ for each i, where $k_i = |C \cap V(G_i)|$. Consider some i and let J_i be a T-join of $G_i[C_i \cup \mathbf{A}(G_i)]$.

Let Z be a connected component of $G_i[J_i]$. We show how to find a connected subgraph Z' of F_i (and thus of G_i) such that $V(Z') \cap T_i \supseteq V(Z) \cap T_i$ and $|V(Z') \cap \mathbf{B}(G_i)| \leq |V(Z) \cap \mathbf{B}(G_i)|$. Consider the subgraph $\hat{Z}$ of $\hat{G}_i$ formed by $\pi^{-1}(V(Z) \cup E(Z))$. Note that any connected component Y of $\hat{Z}$ connects $A = V(Y) \cap \partial \hat{G}_i$. Then by Theorem 3.1, there is a subgraph $H(Y)$ of $\tilde{G}_i$ that connects A and has minimum possible cost among all subgraphs of G_i that connect A. Hence, $|V(H(Y)) \cap \mathbf{B}(G_i)| \leq |V(Y) \cap \mathbf{B}(G_i)|$. Now let H be the union of $H(Y)$ over all connected components Y of $\hat{Z}$. Observe that H is a subgraph of $\tilde{G}_i$. Let $Z' = (\pi(V(H)), \pi(E(H)))$. Observe that, by construction, Z' is a subgraph of F_i with the claimed properties. In particular, observe that although $\hat{Z}$ can be much larger than Z due to the duplication of vertices of $Z \cap S_i$ when G_i was cut open along S_i, we de-duplicate these vertices when using $\pi(V(H))$.

Consider the union J_i' of all these connected subgraphs Z' over all connected components Z of $G_i[J_i]$. Then $|V(G_i[J_i']) \cap \mathbf{B}(G_i)| \leq |V(G_i[J_i]) \cap \mathbf{B}(G_i)| = |C_i|$. Moreover, by construction, for each connected component ZZ of $G_i[J_i']$ there exists a set $\mathcal{Z}(ZZ)$ of connected components Z of $G_i[J_i]$ such that $ZZ \cap T$ is the union of $Z \cap T$ over all these connected components Z. We note that the sets $\mathcal{Z}(ZZ)$ induce a partition of the connected components of $G_i[J_i]$. Observe that the parity of $|ZZ \cap T|$ is equal to the sum (mod 2) of the parities of the corresponding connected components of $G_i[J_i]$, and thus, equal to 0. It follows that $G_i[J_i']$ contains a T_i-join. Hence, $V(G_i[J_i']) \cap \mathbf{B}(G_i)$ is a solution for $(G_i, \mathbf{A}(G_i), \mathbf{B}(G_i), T_i, k_i)$. By repeating this procedure for all i, it follows from the proof of Lemma 4.13 that the union of these solutions is a solution for $(G, \mathbf{A}(G), \mathbf{B}(G), T, k)$. Moreover, any T-join that is contained in this solution is fully contained in F. Hence, $(F, \mathbf{A}(F), \mathbf{B}(F), T, k)$ has a solution. ◀

▶ **Corollary 4.15.** Plane Odd Cycle Transversal *has a polynomial kernel.*

5 Vertex Multiway Cut

In this section we sketch our polynomial kernel for Planar Vertex Multiway Cut. Many important details are omitted in this presentation, but the full version can be found on arXiv [18]. Now, let (G, T, k) be an input of Planar Vertex Multiway Cut.

The first step towards the kernel is a preprocessing routine that ostensibly aims to reduce the diameter of G. Recall that for Odd Cycle Transversal, we could reduce the diameter of the radial graph to $\mathcal{O}(k^3)$ by Lemma 4.13, which enabled us to find a small tree connecting the terminals in Theorem 4.14 along which we could cut open the graph. For Planar Vertex Multiway Cut, we use a much more involved argument to find this small tree.

To be more precise, we partition the vertices of the input plane graph G using its outerplanarity layers. A vertex belongs to outerplanarity layer $k \geq 1$ if it is on the outer face after $k-1$ times simultaneously removing all vertices on the outer face. We then obtain a

tree, denoted $\mathbb{T}(G)$, by simultaneously contracting all edges whose endpoints belong to the same layer; note that this operation shrinks each connected component induced by each layer to a node of the tree, and each node u of the tree corresponds to a set $\kappa(u)$ of vertices of G. We call a node u in $\mathbb{T}(G)$ *important* if $\kappa(u)$ contains a terminal or if two distinct children of u have a descendant v in $\mathbb{T}(G)$ for which $\kappa(v)$ contains a terminal. We then argue that if the unique path of P in $\mathbb{T}(G)$ between two important nodes has length more than $2(k+1)$, then any optimal T-multiway cut will only use vertices corresponding to the first $k+1$ nodes (call this set Q), to the last $k+1$ nodes (call this set R), or to a $(\kappa(x), \kappa(y))$-cut of size at most k for some $x \in Q, y \in R$. This means that only $\mathcal{O}(k^3)$ nodes along P are relevant, which combined with the definition of important nodes leads to $\mathcal{O}(k^3|T|)$ relevant nodes in $\mathbb{T}(G)$.

The intuition behind relevant nodes is that we are only interested in the part of the graph induced by relevant nodes, and thus we simultaneously contract all edges of G whose endpoints both belong to a non-relevant node of $\mathbb{T}(G)$. We denote by Z the set of vertices that arise due to this contraction. We must forbid that Z belongs to the solution of the kernel, a detail later dealt with by replacing each vertex of Z by a suitably chosen grid. We now use the LP-based reduction rules of Cygan et al. [4] to reduce the number of terminals to $2k$, so that there are only $\mathcal{O}(k^4)$ relevant nodes. The definition of $\mathbb{T}(G)$ combined with the contraction we described earlier implies that the radial distance of each terminal to the outer face is $\mathcal{O}(k^4)$. This enables us to find a tree H of size $\mathcal{O}(k^5)$ in the overlay graph of G along which we cut it open (cf. Figure 2). Call the resulting graph $\hat{G}$.

The second step of the kernel is to establish a correspondence between vertex cuts X in G and Steiner trees in $\hat{G}$ that connect vertices along $\partial\hat{G}$. To this end, observe that each connected component of $G \setminus X$ can be bounded by a closed curve γ that intersects the drawing of G only in vertices of X. This curve corresponds to a closed curve γ^* in the overlay graph of G that intersects its drawing only in vertices of X or $F(G)$. This set of intersected vertices X' will contain vertices of H such that in $\hat{G}$, X' can be decomposed to induce several Steiner trees that connect vertices along $\partial\hat{G}$. Then it suffices to note that the cost of each Steiner tree only depends on the number of vertices of $V(G)$ it contains and that vertices of $F(G)$ are free. By picking $\mathbf{A}(\hat{G}) = V(G)$ and $\mathbf{B}(\hat{G}) = F(G)$, we can then apply Theorem 3.1.

Note that the kernel contains, for each cut $X \subseteq V(G)$, a set $X' \subseteq V(G)$ that *mimics* the set X in the following way: for each $Y \subseteq X \cap V(\partial\hat{G})$ which is contained in a single connected component of $\hat{G}[X \cup F]$, the set Y is contained in a single connected component of $\hat{G}[X' \cup F]$. Then for every pair of vertices $u, v \in T$, if X is a (u,v)-cut in G then X' is also a (u,v)-cut in G. Hence by preserving minimum connectors for subsets of $V(\partial\hat{G})$, we preserve minimum solutions to Planar Vertex Multiway Cut.

6 Reductions to Vertex Planarization

In this section we show two reductions from Planar Vertex Multiway Cut: one to the disjoint version of Vertex Planarization, and one to the regular one. We start with recalling formal problem definitions.

Vertex Planarization	**Parameter:** k
Input: A graph G and an integer k.	
Question: Does there exist a set $X \subseteq V(G)$ such that $G \setminus X$ is planar?	

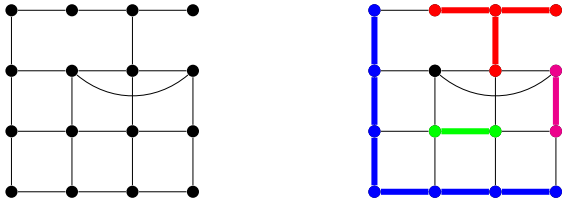


Figure 3 The graph H_0 of Observation 6.1 with the K_5 minor model on the right.

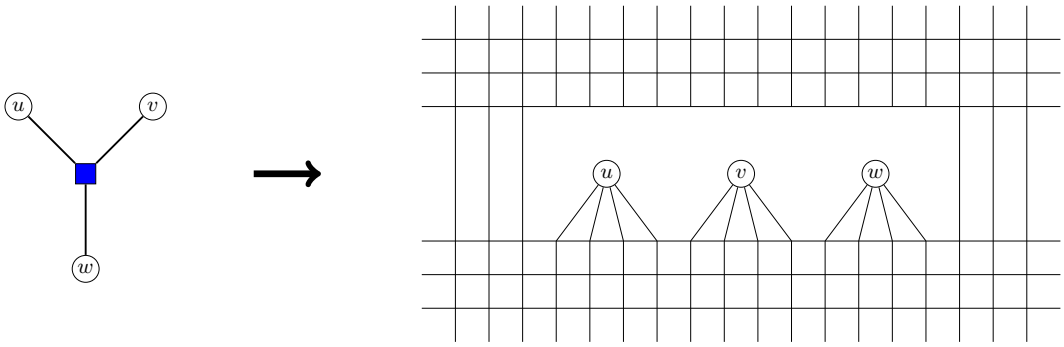


Figure 4 Embedding neighbors of a terminal (blue square) into a hole cut out in a large grid. Every neighbor of a terminal is connected to $k+1$ vertices of the grid ($k+1=4$ in the figure).

Disjoint Vertex Planarization	**Parameter:** $k+\lvert S\rvert$
Input: A graph G, a set $S \subseteq V(G)$ such that $G \setminus S$ is planar, and an integer k.	
Question: Does there exist a set $X \subseteq V(G) \setminus S$ of size at most k such that $G \setminus X$ is planar?	

Lemma 6.2 and 6.3 below give polynomial-parameter transformations from Planar Vertex Multiway Cut to Disjoint Vertex Planarization and Vertex Planarization.

Both reductions rely on the same idea: if a Vertex Planarization instance contains a large grid, the budget of k deletions is not able to effectively break it, and there is essentially only one way to embed it in the plane. If some parts of the graph are attached to vertices of the grid incident to faces far away from each other, a solution to Vertex Planarization needs to separate such parts from each other. This allows to embed a Planar Vertex Multiway Cut instance. Formally, we rely on the following observation (see Figure 3).

▶ **Observation 6.1.** *Consider the following graph H_0: we start with H_0 being a 4×4 grid with vertices $x_{a,b}$, $1 \leq a, b \leq 4$ (i.e., the vertex $x_{a,b}$ lies in a-th row and b-th column of the grid) and then add an edge $x_{2,2}x_{2,4}$ but delete edges $x_{1,2}x_{2,2}$ and $x_{1,4}x_{2,4}$. Then H_0 contains a K_5 minor and is therefore not planar.*

▶ **Lemma 6.2.** *Given a* Planar Vertex Multiway Cut *instance (G, T, k), one can in linear time compute an equivalent* Disjoint Vertex Planarization *instance (G', S, k) with $|S| \leq 8|T|$.*

Proof. If $|T| \leq 1$, then the input instance is trivial, and we can output $G' = S = \emptyset$. Otherwise, let $T = \{t_1, t_2, \ldots, t_{|T|}\}$. We start by constructing a $4 \times 2|T|$ grid H. Denote $S = V(H)$; note that $|S| = 8|T|$ as promised. For $1 \leq i \leq |T|$, let x_i be the $(2i)$-th vertex in

the second row of H. We construct the graph G' from $G \uplus H$ by identifying t_i with x_i for every $1 \leq i \leq |T|$. We claim that the resulting Disjoint Vertex Planarization instance (G', S, k) is equivalent to the input Planar Vertex Multiway Cut instance (G, T, k). Note that $V(G) \setminus T = V(G') \setminus S$.

In one direction, let $X \subseteq V(G) \setminus T$ be a solution to Planar Vertex Multiway Cut on (G, T, k). We show that X is also a solution to Disjoint Vertex Planarization on (G', S, k) by showing a planar embedding of $G' \setminus X$. First, embed H in the natural way. Second, for every connected component C of $G \setminus X$, proceed as follows. If C contains a terminal t_i, then fix a planar embedding of C that keeps t_i incident to the infinite face, and embed C in one of the faces of H incident with x_i. Otherwise, if C does not contain any terminal, embed C in the infinite face of H. Since every connected component C contains at most one terminal, this is a valid planar embedding of $G' \setminus X$.

In the other direction, let $X \subseteq V(G') \setminus S$ be a solution to Disjoint Vertex Planarization on (G', S, k). We claim that X is also a solution to Planar Vertex Multiway Cut on (G, T, k). Assume the contrary; since $|X| \leq k$ and $X \subseteq V(G') \setminus S = V(G) \setminus T$ by assumption, we have two terminals $t_i, t_j \in T$ and a $t_i - t_j$ path P in $G \setminus X$. Consider the subgraph $H \cup P$ of $G' \setminus X$ and contract P to a single edge $t_i t_j$. Then, this minor of $G' \setminus X$ contains H_0 from Observation 6.1 as a minor. By Observation 6.1, $G' \setminus X$ contains K_5 as a minor, contradicting its planarity. ◀

▶ **Lemma 6.3.** *Given a* Planar Vertex Multiway Cut *instance* (G, T, k)*, one can in polynomial time compute an equivalent* Vertex Planarization *instance* (G', k) *with* $|E(G')| + |V(G')| \leq \mathcal{O}(k(|E(G)| + |V(G)|))$.

Proof. We proceed as in the proof of Lemma 6.2, but we need to make H thicker in order not to allow any tampering.

If $|T| \leq 1$, then the input instance is trivial, and we can output $G' = \emptyset$. Similarly, we output a trivial no-instance if two terminals of T are adjacent. Otherwise, fix a planar embedding ϕ of G and let $T = \{t_1, t_2, \ldots, t_{|T|}\}$. For every $1 \leq i \leq |T|$, let d_i be the degree of t_i in G and let $v_i^1, \ldots, v_i^{d_i}$ be the neighbors of t_i in G in clockwise order around t_i in ϕ. Let $D = \sum_{i=1}^{|T|} d_i$.

We define a graph H as follows. We start with H being a $4(k+1) \times (D + |T|)(k+1)$-grid with vertices $x_{a,b}$, $1 \leq a \leq 4(k+1)$, $1 \leq b \leq (D + |T|)(k+1)$ (i.e., the vertex $x_{a,b}$ lies in a-th row and b-th column). For every $1 \leq i \leq |T|$, let $b_i^{\leftarrow} = (i + \sum_{j<i} d_j)(k+1)$ and $b_i^{\rightarrow} = b_i^{\leftarrow} + d_i(k+1)$; additionally, let $b_0^{\rightarrow} = 0$. For every $1 \leq i \leq |T|$ and every $b_i^{\leftarrow} < b \leq b_i^{\rightarrow}$, we delete from H the edge $x_{k+1,b}x_{k+2,b}$; see Figure 4.

We now define the graph G' as follows. We start with $G' = H \uplus (G \setminus T)$. Then, for every $1 \leq i \leq |T|$ and every $1 \leq j \leq d_i$, we make v_i^j adjacent to $x_{k+2,b}$ for every $b_i^{\leftarrow} + (j-1)(k+1) < b \leq b_i^{\leftarrow} + j(k+1)$. This finishes the construction of the Vertex Planarization instance (G', k). We now show that it is equivalent to Planar Vertex Multiway Cut on (G, T, k).

In one direction, let X be a solution to Planar Vertex Multiway Cut on (G, T, k). We show that X is also a solution to Vertex Planarization on (G', k) by constructing a planar embedding of $G' \setminus X$. First, we embed H naturally and for every $1 \leq i \leq |T|$ let f_i be the face of the embedding that is incident with vertices $x_{k+2,b}$ for every $b_i^{\leftarrow} < b \leq b_i^{\rightarrow}$. Then, for every connected component C of $G \setminus X$ we proceed as follows. If C does not contain a terminal, then since $X \cap T = \emptyset$, component C contains no neighbors of terminals either; hence the vertices of C are not adjacent to H in G'. We embed C in the infinite face of H. Otherwise, assume that the only terminal of C is t_i. We take the embedding of

C induced by ϕ, change the infinite face so that t_i is incident with the infinite face, and embed $C \setminus t_i$ with the induced embedding into f_i. The fact that $v_i^1, \ldots, v_i^{d_i}$ are embedded around t_i in ϕ in this order allows us now to draw all edges between vertices of $N_G(t_i)$ and $\{x_{k+2,b} | b_i^{\leftarrow} < b \leq b_i^{\rightarrow}\}$ in a planar fashion.

In the other direction, let X' be a solution to Vertex Planarization on (G', k). We claim that $X := X' \cap (V(G) \setminus T)$ is a solution to Planar Vertex Multiway Cut on (G, T, k). If this is not the case, then there exist two terminals t_{i_1}, t_{i_2}, $1 \leq i_1 < i_2 \leq |T|$ and a path P from t_{i_1} to t_{i_2} in $G \setminus X$. Let $v_{i_1}^{j_1}$ be the neighbor of t_{i_1} on P and $v_{i_2}^{j_2}$ be the neighbor of t_{i_2} on P. Since $|X'| \leq k$, there exist:

- indices $1 \leq a_1 \leq k+1$, $k+2 \leq a_2 \leq 2k+2$, $2k+3 \leq a_3 \leq 3k+3$, $3k+4 \leq a_4 \leq 4k+4$ such that no vertex of X' is in rows numbered a_1, a_2, a_3, nor a_4 of H;
- for every $1 \leq i \leq |T|$, an index $b_{i-1}^{\rightarrow} < b_i \leq b_i^{\leftarrow}$ with no vertex of X' in the b_i-th column of H; and
- for every $1 \leq i \leq |T|$ and every $1 \leq j \leq d_i$ an index $b_i^{\leftarrow} + (j-1)(k+1) < b_i^j \leq b_i^{\leftarrow} + j(k+1)$ with no vertex of X' in the b_i^j-th column of H.

We conclude by observing that the graph H_0 from Observation 6.1 is a minor of a subgraph of $G' \setminus X$ induced by P, the a_1-th, a_2-th, a_3-th, and a_4-th rows of H, and columns of H with numbers b_{i_1}, $b_{i_1}^{j_1}$, b_{i_2}, $b_{i_2}^{j_2}$. ◀

7 Conclusions

We conclude with several open problems. First, the exponents in the polynomial bounds of our kernel sizes are enormous, similarly as for planar Steiner tree [25]. Thus, we reiterate the question of reducing the bound of the main sparsification routine of [25] to quadratic. Second, we hope that our tools can pave the way to a polynomial kernel for Vertex Planarization, which remains an important open problem. Third, nothing is known about the kernelization of Multiway Cut parameterized above the LP lower bound [4], even in the case of planar graphs and edge deletions.

References

1 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.

2 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

3 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.

4 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *TOCT*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.

5 Holger Dell and Dieter van Melkebeek. Satisfiability Allows No Nontrivial Sparsification unless the Polynomial-Time Hierarchy Collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

6 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

7 Andrew Drucker. New Limits to Classical and Quantum Instance Compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015. doi:10.1137/130927115.

8 Samuel Fiorini, Nadia Hardy, Bruce Reed, and Adrian Vetta. Planar graph bipartization in linear time. *Discrete Applied Mathematics*, 156:1175–1180, 2008. doi:10.1016/j.dam.2007.08.013.

9 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

10 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and Kernels. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510. SIAM, 2010. doi:10.1137/1.9781611973075.43.

11 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

12 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in node weighted graphs. *J. Algorithms*, 50(1):49–61, 2004. doi:10.1016/S0196-6774(03)00111-1.

13 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform Kernelization Complexity of Hitting Forbidden Minors. *ACM Trans. Algorithms*, 13(3):35:1–35:35, 2017. doi:10.1145/3029051.

14 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011. doi:10.1016/j.disopt.2010.05.003.

15 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4:221–225, 1975. doi:10.1137/0204019.

16 Bart M. P. Jansen. Polynomial Kernels for Hard Problems on Disk Graphs. In *Proc. 12th SWAT*, volume 6139, pages 310–321. Springer, 2010. doi:10.1007/978-3-642-13731-0_30.

17 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A Near-Optimal Planarization Algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

18 Bart M. P. Jansen, Marcin Pilipczuk, and Erik Jan van Leeuwen. A deterministic polynomial kernel for Odd Cycle Transversal and Vertex Multiway Cut in planar graphs. *CoRR*, abs/1810.01136, 2018. arXiv:1810.01136.

19 Ken-ichi Kawarabayashi. Planarity Allowing Few Error Vertices in Linear Time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 639–648. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.45.

20 Stefan Kratsch and Magnus Wahlström. Representative Sets and Irrelevant Vertices: New Tools for Kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

21 Stefan Kratsch and Magnus Wahlström. Compression via Matroids: A Randomized Polynomial Kernel for Odd Cycle Transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.

22 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - Preprocessing with a Guarantee. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012. doi:10.1007/978-3-642-30891-8_10.

23 Dániel Marx and Ildikó Schlotter. Obtaining a Planar Graph by Vertex Deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

24 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network Sparsification for Steiner Problems on Planar and Bounded-Genus Graphs. *CoRR*, abs/1306.6593, 2013. arXiv:1306.6593.

25 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network Sparsification for Steiner Problems on Planar and Bounded-Genus Graphs. *ACM Trans. Algorithms*, 14(4):53:1–53:73, 2018. doi:10.1145/3239560.

26 Igor Razgon. Large Isolating Cuts Shrink the Multiway Cut. *CoRR*, abs/1104.5361, 2011. arXiv:1104.5361.

27 Saket Saurabh. Open problems from Recent Advances in Parameterized Complexity school, 2017. https://rapctelaviv.weebly.com/uploads/1/0/5/3/105379375/future.pdf.

28 Ondřej Suchý. Extending the Kernel for Planar Steiner Tree to the Number of Steiner Vertices. *Algorithmica*, 79:189–210, 2017. doi:10.1007/s00453-016-0249-1.

A Characterization of Subshifts with Computable Language

Emmanuel Jeandel
LORIA, Campus Scientifique - BP 239, 54506 Vandoeuvre-les-Nancy, France
emmanuel.jeandel@loria.fr

Pascal Vanier
Laboratoire d'Algorithmique, Complexité et Logique, Université de Paris-Est, LACL, UPEC, France
pascal.vanier@lacl.fr

Abstract

Subshifts are sets of colorings of $\mathbb{Z}^d$ by a finite alphabet that avoid some family of forbidden patterns. We investigate here some analogies with group theory that were first noticed by the first author. In particular we prove several theorems on subshifts inspired by Higman's embedding theorems of group theory, among which, the fact that subshifts with a computable language can be obtained as restrictions of minimal subshifts of finite type.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases subshifts, computability, Enumeration degree, Turing degree, minimal subshifts

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.40

Funding *Pascal Vanier*: Sponsored by grant TARMAC ANR 12 BS02 007 01.

Acknowledgements The authors wish to thanks the anonymous referees for many helpful remarks and improvements.

1 Introduction

Subshifts are colorings of $\mathbb{Z}^d$ by some finite alphabet Σ avoiding some family of forbidden patterns. They are closed shift invariant subsets of $\Sigma^{\mathbb{Z}^d}$. The most commonly studied family of subshifts are the subshifts of finite type (SFTs), those that can be defined via a finite family of forbidden patterns, which correspond to the sets of colorings by Wang tilesets.

It is well known since the work of Berger [5] that many problems or invariants in tiling theory, and therefore for subshifts of finite type, are not computable. A recent trend in multidimensional symbolic dynamics initiated by Hochman [16, 17] shows that computability is not a fluke but an integral part of the study of subshifts. Indeed, many recent results show precise correspondences between computability notions and invariants for subshifts [25, 19]. This has led to the study of another class of subshift, effective (or effectively closed) subshifts: subshifts which are defined by a recursively enumerable family of forbidden patterns.

Of particular interest is the embedding (simulation) theorem by Hochman [16], extended by Aubrun-Sablik and Durand-Romashchenko-Shen [2, 10], that characterizes effectively closed subshifts, as projections of higher dimensional subshifts of finite type,

This theorem is strikingly similar to theorems in combinatorial group theory and first order logic. The Higman embedding theorem [14] characterizes recursively presented groups, i.e. groups given by a *computable* set of relators, as subgroups of finitely presented groups, i.e. groups given by a *finite* set of relators. The Kleene-Craig-Vaught [21, 8] theorem characterizes recursively axiomatisable theories, i.e. theories given by a *computable* set of axioms, as syntactic restrictions of finitely axiomatisable theories, i.e. theories given by a *finite* set of axioms.

Based on this analogy, the first author described a general theory [18] in which many theorems of these three fields can be formulated using an unified framework, and a dictionary between similar notions can be established. The framework is quite abstract and it cannot be used to prove the embedding theorems above for all these theories at once: they rely after all in each case on properties of an encoding of Turing machines, and this encoding heavily depends on the theory under consideration. It suggests nonetheless that there is more than a similarity between these theorems, and that something deeper is to be found.

In this article, we study this by providing analogues in symbolic dynamics of the other embedding theorems of Higman:

- The relative Higman theorem [15] which, as its name indicates, is a relativized version of the classic Higman theorem
- The Boone-Higman-Thompson [6, 26] theorem that characterizes groups with computable word problem as subgroups of simple recursively presented groups.

The first theorem is presented in section 3.2. It is very similar to a theorem in a previous article by Aubrun and Sablik [1]. As we will explain, their article suffers however from unfortunate mistakes and the theorem they proved is regrettably wrong.

The second theorem is presented in section 3.3. The Boone-Higman-Theorem in our context, becomes: "A subshift has a computable language iff it is the restriction of a minimal subshift, itself a restriction of a subshift of finite type". Using recent results from Durand and Romashchenko [11], this can be simplified to "A subshift has a computable language iff it is the restriction of a minimal subshift of finite type". Whether such a simplification is possible for groups (i.e. whether any group with a computable word problem is a subgroup of a finitely presented simple group) is a long standing open question.

The article is organized as follows. We first start with defining the relevant notions from symbolic dynamics, computability theory, and group theory. We will then explain how concepts from group theory translate into notions of symbolic dynamics. The remaining part is devoted to the proof of the three Higman theorems for subshifts: the classic Higman theorem (a slight reformulation of the Hochman-Aubrun-Sablik-Durand-Romashchenko-Shen theorem), the relative Higman theorem and the Boone-Higman-Thompson theorem.

2 Preliminary definitions

2.1 Subshifts

The d-dimensional full shift is the set $\Sigma^{\mathbb{Z}^d}$ where Σ is a finite alphabet whose elements are called *letters* or *symbols*. Each element of the full shift may be seen as a coloring of $\mathbb{Z}^d$ with the letters of Σ. For $v \in \mathbb{Z}^d$, the shift function $\sigma_v : \Sigma^{\mathbb{Z}^d} \to \Sigma^{\mathbb{Z}^d}$ is defined by $\sigma_v(x_z) = x_{z+v}$. The full shift equipped with the distance $d(x, y) = 2^{-\min\{\|v\| \mid v \in \mathbb{Z}^d, x_v \neq y_v\}}$ forms a compact metric space on which the shift functions act as homeomorphisms. A closed shift invariant subset X of $\Sigma^{\mathbb{Z}^d}$ is called a *subshift* or *shift*. An element of a subshift X is called a *configuration* or *point*.

Subshifts are exactly the subsets of $\Sigma^{\mathbb{Z}^d}$ that avoid some *family of forbidden patterns*. A *pattern* of shape P, where P is a *4-connected*[1] finite subset of $\mathbb{Z}^d$, is an element of Σ^P or alternatively a function $p : P \to \Sigma$. A configuration x avoids a pattern p of shape P if $\forall z \in \mathbb{Z}^d, p \neq \sigma_z(x)_{|P}$.

[1] The exact notion of connectedness we use is irrelevant. However it is crucial in what follows to look only at connected patterns.

Subshifts can thus be defined by some family of patterns they avoid. When a subshift can be defined this way by a finite family, it is called a *subshift of finite type*. When a subshift can be defined by a recursively enumerable family of forbidden patterns, it is called an *effectively closed subshift*.

If X is a subshift, we denote by $\mathcal{L}(X)$ its *language*, *i.e.* the set of patterns that appear somewhere in one of its points.

▶ **Example 1.** The set X_1 of all biinfinite words over the alphabet $\{a, b\}$ that do not contain the word aa is, by definition, a subshift. It is defined by the set of forbidden patterns $\mathcal{F} = \{aa\}$. Another possible defining set of forbidden patterns is $\mathcal{F} = \{aab, aaa\}$

▶ **Example 2.** The set X_2 of all biinfinite words over the alphabet $\{a, b\}$ where the letter a appears at most once is a subshift. It is defined e.g. by the set of forbidden patterns $\mathcal{F} = \{ab^n a, n \in \mathbb{N}\}$. It can be proven that it is not a subshift of finite type, although it is certainly an effectively closed subshift.

We denote by $\Sigma^{d\star}$ the set of d-dimensional patterns over the alphabet Σ. For $d = 1$, we write this $\Sigma^\star$. As an abuse of notation, we consider a d-dimensional pattern to be also a k-dimensional pattern for $k > d$ along the d first dimensions: as an example if X is a d-dimensional subshift, $\mathcal{L}(X) \cap A^\star$ is the set of one dimensional patterns (i.e. horizontal words) over the alphabet A that appear in X.

▶ **Example 3.** Let X_3 be the two-dimensional subshift over the alphabet $\{0, 1\}$ defined with the set of forbidden patterns $\mathcal{F} = \{\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1&1\end{smallmatrix}\right)\}$. X_3 is therefore the set of colorings of the plane with 0 and 1 s.t. no two symbols 1 can be put next to each other. It is easy to see that $\left(\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right) \in \mathcal{L}(X_3)$ but $\left(\begin{smallmatrix}1&0\\1&0\end{smallmatrix}\right) \notin \mathcal{L}(X_3)$.

Notice that any subshift X can always be defined by its set $\mathcal{L}(X)^c$. In particular X is an effectively closed subshift iff $\mathcal{L}(X)^c$ is recursively enumerable.

2.2 Combinatorial Group Theory

We assume the reader has a passing familiarity with group theory, and will focus this brief description to the specifics of combinatorial group theory.

A good introduction to this particular aspect may be found in [22, 24]. The book by Higman and Scott [15] contains invaluable information about the interplay between group theory and computability.

A set of generators for a group G is a set S s.t. for any $g \in G$, there exist $s_1^{\pm 1}, \dots, s_n^{\pm 1} \in S$ such that $g = s_1 \cdots s_n$. A group is *finitely generated* if there exists a finite such S.

Let $a_1 \dots a_k$ be a set of generators for some finitely generated group G. The *word problem* for G, denoted $\mathcal{WP}(G, \{a_1 \dots a_k\})$ is the language of all formal words over the alphabet $\{a_1^{\pm 1} \dots a_k^{\pm 1}\}$ that evaluates to 1 (the identity element) in G. The computability properties of $\mathcal{WP}(G, \{a_1 \dots a_k\})$ do not depend on the set of generators (as long as it is finite), so that we will usually speak of the word problem as $\mathcal{WP}(G)$ without specifying the generators.

There is (up to isomorphism) a unique largest group generated by n elements, which is called the free group F_n on n generators. If the generators are written $a_1 \dots a_n$, F_n can be thought of as the set of all irreducible words over the alphabet $\{a_1^{\pm 1} \dots a_n^{\pm 1}\}$, i.e. all words that do not contain $a_i a_i^{-1}$ or $a_i^{-1} a_i$ as factors, with the obvious product operation.

F_n is the largest group with n generators $a_1 \dots a_n$ in the sense that if G is a group with n generators $s_1 \dots s_n$, then there is a unique onto morphism ϕ s.t. $\phi(a_i) = s_i$.

In particular any group with n generators can be seen as a quotient of a free group. This gives rise to the notion of groups given by generators and relations.

If $\mathcal{R}$ is a set of formal words over $\{a_1^{\pm 1} \dots a_n^{\pm 1}\}$, we denote by $\langle a_1, a_2, \dots a_n \mid \mathcal{R}\rangle$ the largest group G generated by n elements $a_1 \dots a_n$ s.t. all relations in $\mathcal{R}$ evaluate to 1 in the group G. Formally, G is the quotient of the free group F_n by the smallest normal subgroup N of F_n that contains all relations $\mathcal{R}$.

A finitely generated group G is finitely presented if $G = \langle S \mid \mathcal{R}\rangle$ for some finite S and $\mathcal{R}$, or more generally if G is isomorphic to such a group. G is *recursively presented* if $G = \langle S \mid \mathcal{R}\rangle$ for some finite[2] S and recursively enumerable set $\mathcal{R}$.

▶ **Example 4.** The group $G = \mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$ is finitely presented. A possible finite presentation is $G = \left\langle a_1, a_2 \;\middle|\; a_1 a_2 a_1^{-1} a_2^{-1}, a_2^3 \right\rangle$. There are of course other presentations with the same generators, for example $G = \left\langle a_1, a_2 \;\middle|\; a_2 a_1 a_2 a_1^{-1} a_2, a_2^3 \right\rangle$.

For this group G, we have $a_1 a_2 a_1 a_2 \notin \mathcal{WP}(G, \{a_1, a_2\})$ and $a_1^2 a_2 a_1^{-1} a_2 a_1^{-1} a_2 \in \mathcal{WP}(G, \{a_1, a_2\})$.

Notice that for all groups G with generators S, we have that $G = \langle S \mid \mathcal{WP}(G, S)\rangle$ and that G is recursively presented iff $\mathcal{WP}(G)$ is recursively enumerable.

2.3 Subshifts as analogs of subgroups

There is a natural analogy between subshifts and subgroups, which is obtained in the following way: the alphabet plays the role of the generators and the forbidden patterns play the role of the relations.

If X is a d-dimensional subshift over the alphabet Σ given by the forbidden patterns $\mathcal{F}$, we will write $X = \langle \Sigma \mid \mathcal{F}\rangle^d$ to further stress the analogy between groups and subshifts.

Continuing the analogy, the word problem $\mathcal{WP}(G)$ of G correspond naturally to the complement of the language of X, $\mathcal{L}(X)^c$. In particular, if S is a set of generators, $G = \langle S \mid \mathcal{WP}(G, S)\rangle$. If X is a subshift over alphabet Σ, then $X = \langle \Sigma \mid \mathcal{L}(X)^c\rangle^d$.

To further the correspondence, we need an analogy in subshifts of the operations of adding/removing generators and relations. In terms of groups, H is obtained from G by adding relations iff H is a quotient of G. In terms of subshifts, Y is obtained from X by adding forbidden patterns iff $X \subseteq Y$. So taking a quotient corresponds to subshift containment.

If H is obtained from G by removing generators, it means that H is a subgroup of G (of course not all subgroups can be obtained this way). What the operations of removing symbols means for subshifts is discussed in the following section.

2.3.1 Removing symbols and dimensions

Removing symbols, or removing dimensions, is intuitively easy:

▶ **Definition 5.** *Let X' be a subshift over an alphabet Σ' of dimension d' and let $\Sigma \subseteq \Sigma'$ and $d < d'$, the (Σ^k, d)* restriction *X of X' is the set of d-dimensional configurations of width k over the alphabet Σ that appear in X'. We write $X \prec X'$ if X is some restriction of X'.*

By compactness X is exactly the subshift of dimension d over the alphabet Σ^k that forbids all patterns in $\mathcal{L}(X')^c \cap \left(\Sigma^k\right)^{\star d}$.

[2] One could take more generally S to be recursively enumerable

▶ **Example 6.** Let $X' = \langle a, b \mid \left(\begin{smallmatrix} a & a \end{smallmatrix}\right), \left(\begin{smallmatrix} b & b \end{smallmatrix}\right), \left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right), \left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right) \rangle^2$, it is easy to see that X' contains only two configurations: a and b alternate on every row, and columns are uniform. That is every configuration locally look like figure 1.

$$\begin{array}{cccccccc}
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
\dots & a & b & a & b & a & b & \dots \\
\dots & a & b & a & b & a & b & \dots \\
\dots & a & b & a & b & a & b & \dots \\
\dots & a & b & a & b & a & b & \dots \\
\dots & a & b & a & b & a & b & \dots \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots
\end{array}$$

Figure 1 Configurations of X'.

The $(\{a, b\}, 1)$ restriction of X' is therefore the one-dimensional subshift that contains all two configurations, that alternate a and b. The $(\{a\}, 1)$ restriction of X' is the empty subshift, and the $(\{a, b\}^2, 1)$ restriction contains exactly the two configurations that alternate $\left(\begin{smallmatrix} a \\ a \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} b \\ b \end{smallmatrix}\right)$.

In terms of computability, the restriction is significant : If $X \prec X'$ then X can be more complicated than X':

▶ **Proposition 7.** *If $X \prec X'$ then $\mathcal{L}(X)$ is corecursively enumerable in $\mathcal{L}(X')$.*

Indeed $P \in \mathcal{L}(X)$ iff for all n there exists a d' dimensional pattern of size n in $\mathcal{L}(X')$ with P at its center. (More precisely, $\mathcal{L}(X)^c$ is enumeration-reducible to $\mathcal{L}(X')^c$, see below for the definition.)

▶ **Proposition 8.** *There exist two subshifts $X \prec X'$ s.t. $\mathcal{L}(X')$ is computable and $\mathcal{L}(X)$ is not computable.*

Proof. Let X be any one-dimensional effectively closed subshift over the alphabet $\{a, b\}$ with a noncomputable language. It is well known that X can be given by a *computable* family of forbidden patterns $\mathcal{F}$ (see e.g. [3]).

Now let X' be the subshift over the alphabet $\{a, b, \#\}$ given by the same family of forbidden patterns. It is clear that $\mathcal{L}(X')$ is computable. Indeed, let $w \in \{a, b, \#\}^\star$ and write $w = \#u_1\#u_2 \dots \#u_k\#$ with $u_i \in \{a, b\}^\star$, with the $\#$ symbols at the ends possibly missing. Then $w \in \mathcal{L}(X')$ iff each u_i does not contain any element of $\mathcal{F}$. For the nontrivial direction, observe that in this case the biinfinite word ${}^\omega\#w\#^\omega$ does not contain any forbidden word of $\mathcal{F}$. As $\mathcal{F}$ is computable, we can test whether each u_i contains any element of $\mathcal{F}$, and therefore $\mathcal{L}(X')$ is computable.

On the other hand, the restriction of X' to the alphabet $\{a, b\}$ is our initial subshift X, which has an uncomputable language. ◀

This is in contrast with combinatorial group theory, where a (f.g.) subgroup of a group with a computable word problem has immediately a computable word problem. This is due to the fact that looking at subshifts makes us look at infinite objects given by finite words. To obtain theorems similar to Higman's, we will have to force an additional restriction:

▶ **Definition 9.** *Let X' be a subshift over an alphabet Σ' of dimension d' and X be a subshift over an alphabet $\Sigma \subseteq \Sigma'$ of dimension $d < d'$. We say that X is a* full restriction *of X', in symbols $X \sqsubseteq X'$ if $\mathcal{L}(X) = \mathcal{L}(X') \cap \Sigma^{\star d}$*

In other words, if X the (Σ, d) restriction of X', then every d-dimensional infinite word over Σ that can be found in X' is in X. Here we also ask that every *finite* word over Σ that can be found in X' is already in X. In this case:

▶ **Proposition 10.** *If $X \prec X'$ then $\mathcal{L}(X)$ is many-one reducible to $\mathcal{L}(X')$. In particular if $\mathcal{L}(X')$ is computable, then $\mathcal{L}(X)$ is computable.*

Proof. Obvious by definition: $\mathcal{L}(X) = \mathcal{L}(X') \cap \Sigma^{\star d}$. ◀

In this paper we will not be using restrictions of width more than 1.

2.3.2 Adding symbols and dimensions

The operation of adding a dimension is quite obvious.

▶ **Definition 11.** *Let X be a subshift of dimension d over the alphabet Σ. The* extension X' of X to dimension d' *is the subshift of dimension d' that avoids all patterns of $\mathcal{L}(X)^c$.*

A point of X' therefore looks like elements of X stacked in the additional dimensions[3]. Notice that by definition $X \sqsubseteq X'$.

Adding symbols is also easy to define:

▶ **Definition 12.** *Let X be a subshift of dimension d over the alphabet Σ. The* extension X' of X to alphabet $\Gamma \supset \Sigma$ *is the subshift over the alphabet Γ that avoid all patterns of $\mathcal{L}(X)^c$.*

Notice that X' is defined using all patterns of $\mathcal{L}(X)^c$, not only a defining set of forbidden patterns. Notice also that $X \sqsubseteq X'$.

To understand what the points of X' look like, we will first look at an example where $\Gamma = \Sigma \cup \{\#\}$ and X is one-dimensional. In this case, a typical element of X' is of the form $\dots \# u_{-1} \# u_0 \# u_1 \# u_2 \dots$ where each u_i is a finite word in $\mathcal{L}(X)$. Notice in particular that there is no relation between the word u_i and the word u_{i+1}.

If we look at a similar construction in dimension 2, we would see patterns of $\mathcal{L}(X)$ that are separated by # symbols, see Figure 2. The # symbol in the example is what is typically called a *safe symbol* [7, 20] in symbolic dynamics, and is one of the typical conditions needed to obtain good mixing properties on subshifts.

More generally, we could define in the same way the free product of two subshifts:

▶ **Definition 13.** *Let X and Y be two subshifts of the same dimension on disjoint alphabets A and B respectively. The* free product $X * Y$ *is the subshift Z on $A \cup B$ with forbidden patterns $\mathcal{L}(X)^c \cup \mathcal{L}(Y)^c$*

A typical example of a point in Z is depicted in Figure 3. The discrete plane is divided into 4-connected zones that each correspond to a valid pattern of X or a valid pattern of Y.

We note that, for this construction to work, we need the language of a subshift to be defined in terms of *connected* patterns. If we took as the language of a subshift to be all patterns, connected or not, then the extension of X to alphabet $\Sigma \cup \{\#\}$ would merely consist in points of X with some symbols changed to #, which is a different beast altogether.

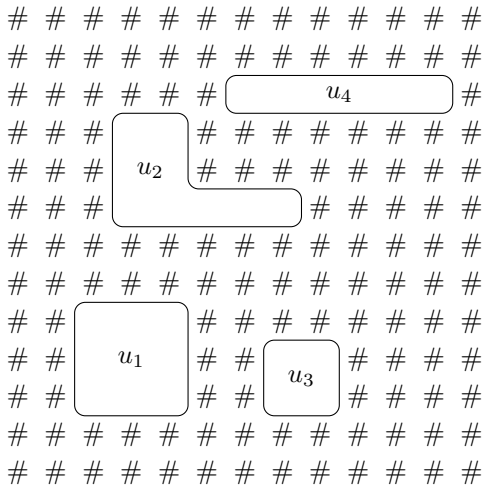


Figure 2 A configuration of X', the extension of $X \subseteq \Sigma^{\mathbb{Z}^2}$ to alphabet $\Gamma = \Sigma \cup \{\#\}$: any (connected) pattern of X can appear anywhere, as long as there are some # separating it from other patterns of X. The (unconnected) pattern consisting of u_1, u_2, u_3 and u_4 may not appear in a valid configuration of X.

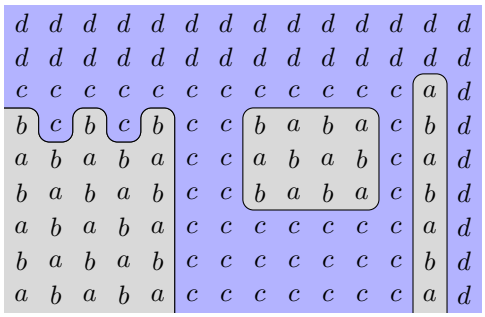


Figure 3 A portion of a valid configuration of the free product of $X = \langle a, b \mid \left(\begin{smallmatrix} a & a \end{smallmatrix}\right), \left(\begin{smallmatrix} b & b \end{smallmatrix}\right), \left(\begin{smallmatrix} a \\ a \end{smallmatrix}\right), \left(\begin{smallmatrix} b \\ b \end{smallmatrix}\right) \rangle^2$ and $Y = \langle c, d \mid \left(\begin{smallmatrix} c & d \end{smallmatrix}\right), \left(\begin{smallmatrix} d & c \end{smallmatrix}\right), \left(\begin{smallmatrix} c \\ d \end{smallmatrix}\right) \rangle^2$, the 4-connected components of X and Y are gray and blue respectively.

Table 1 Dictionary between groups and subshifts.

Group G	Subshift X
Group with n generators	Subshift on n symbols
Free group with n generators	Full shift on n symbols
Word problem $\mathcal{WP}(G)$	co-language $\mathcal{L}(X)^c$
Finitely presented group	SFT
Recursively presented group	Effectively closed subshift
Simple group	Minimal subshift
G_1 is a quotient of G_2	$X_1 \subseteq X_2$
$G_1 \subseteq G_2$	$X_1 \sqsubseteq X_2$ (Definition 9)

3 The three embedding theorems

In this section we prove the equivalent versions of the Higman embedding and Higman relative embedding theorems. To make the article easier to read, we will take some liberties when stating the theorems of Higman. To obtain more exact statements, "G is a subgroup of H" should be replaced by "G is isomorphic to a subgroup of H".

Table 1 gives the correspondence we will use between the vocabulary of groups and the vocabulary of subshifts. It is based on the previous discussion and on the article [18]. The correspondence is not exact, but serves as an intuition for the theorems.

[3] Note that this definition readily generalizes with X over an alphabet Σ^k and X' with alphabet Σ: every row of with k must avoid all patterns of $\mathcal{L}(X)^c$.

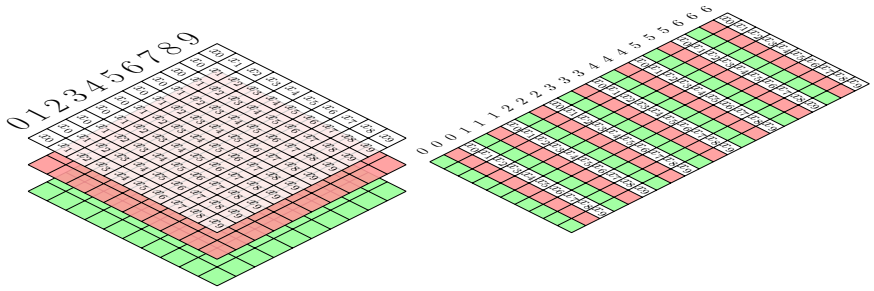


Figure 4 In [16, 2, 9] some layer contains a vertically repeated sequence $(x_i)_{i\in\mathbb{Z}}$ that is checked by some other layers which are superimposed as on the left. It is quite straigtforward to tranform a construction which has layers to an interleaving of the layers as seen on the right.

3.1 The Higman embedding theorem

We start with the first Higman embedding theorem:

▶ **Theorem 14** (Higman embedding theorem [14]). *A f.g. group G is recursively presented iff there exists a finitely presented group H s.t. $G \subseteq H$.*

▶ **Theorem 15** (Higman embedding theorem for subshifts). *A d-dimensional subshift X over an alphabet Σ is effectively closed iff there exists a $d+1$ dimensional SFT X' over an alphabet $\Gamma \supseteq \Sigma$ s.t. $X \sqsubseteq X'$*

As stated in the introduction, this theorem corresponds very closely to a result on subactions of subshifts first discovered by Hochman [16] and then improved by Aubrun-Sablik-Durand-Romashchenko-Shen. We first restate the theorem in a suitable form:

▶ **Theorem 16** ([2, 9]). *A d-dimensional subshift X over an alphabet Σ is effectively closed iff there exists a $(d+1)$-dimensional SFT $X' \subseteq (\Sigma \times \Gamma)^{\mathbb{Z}^{d+1}}$ such that*

$$X = \left\{x \;\middle|\; (x^{\uparrow}, y) \in X'\right\}$$

where $x^{\uparrow}$ is the configuration where for any $z \in \mathbb{Z}^d$ and $j \in \mathbb{Z}$, $x^{\uparrow}_{z,j} = x_z$.

Proof of Theorem 15. All these constructions have one or several computation layers that check a layer on which the effectively closed subshift is written. In our case instead of superimposing the computation layer and the verified layer, we interleave them : if $c = (x, y) \in X \times Y$ in the original construction, the new configuration c' would be formed by $c'_{(i,2j)} = x_{(i,j)}$ and $c'_{(i,2j+1)} = y_{(i,j)}$. This remains an SFT.

We may further assume that the alphabet for the computation is disjoint from the alphabet of the checked subshift. Thus, by restricting the language to the words belonging to the alphabet of the checked layer, only this layer remains. ◀

Note that Higman's original theorem is valid non only for finitely generated groups but for general groups. To obtain a similar statement for subshifts, one would need to deal with subshifts over an infinite alphabet. We think that Hochman's original result [16] on effective dynamical systems provides such a generalization.

3.2 Higman's relative embedding theorem

The relative Higman theorem is, as its name indicates, the relativized version of the Higman embedding theorem, and states conditions on when a group G can be obtained as a subgroup of an extension of a group H. We first need a definition:

▶ **Definition 17** ([15]). *A group K is* finitely presented over G *if K can be obtained from G by adding finitely many generators and finitely many relations*

See [15, Definition 6.1] for the exact definition. The Higman relative embedding theorem then characterizes when a group G can be obtained as a subgroup of a group finitely presented in H. The classical relative embedding theorem correspond to the case where H is trivial. It turns out that the necessary computability criterion has to do with enumeration-reducibility, that we now define:

▶ **Definition 18** ([13]). *If L and M are two sets we say that L is* enumeration-reducible *to M, in symbols $L \leq_e M$ if there exists a partial computable function $f : \mathbb{N} \times \mathbb{N} \to P_f(\mathbb{N})$ where $P_f(\mathbb{N})$ is (a computable representation of) the set of all finite subsets of $\mathbb{N}$ s.t.*

$$x \in L \iff \exists n, f(n, x) \subseteq M$$

The definition might seem quite obtuse at first. Intuitively, $L \leq_e M$ if there is a computable procedure that can enumerate L from any enumeration of M.

The relative embedding theorem is then as follows:

▶ **Theorem 19** (The relative Higman embedding theorem [15]). *K is a subgroup of a group that is finitely presented over G iff $\mathcal{WP}(K) \leq_e \mathcal{WP}(G)$.*

We will now prove our version of the theorem. We first need an analog of "finitely presented over" in terms of subshift:

▶ **Definition 20.** *Let Y be a subshift over an alphabet Σ. U is* of finite type over Y *if U is obtained from Y by adding finitely many new symbols, dimensions, and finitely many new forbidden patterns.*

That is, $U = Y_1 \cap Y_2$, where Y_1 is an extension to a larger alphabet and higher dimension (in the sense of Definitions 11 and 12) of Y, and Y_2 is a subshift of finite type. To be consistent with the exact definition for groups, we also need that $Y \sqsubseteq U$, that is for none of the new forbidden patterns to contain only symbols of Σ.

This definition is straightforwardly extendable to effective subshifts:

▶ **Definition 21.** *Let Y be a subshift over an alphabet Σ. U is* effectively closed over Y *if U is obtained from Y by adding finitely many new symbols, dimensions and a recursively enumerable set of new forbidden patterns. As before, it is required that $Y \sqsubseteq U$.*

A straightforward corollary of Theorem 15 is the following:

▶ **Corollary 22.** *If Y is effectively closed over X, then there exists a subshift Z of finite type over Y such that $X \sqsubseteq Y \sqsubseteq Z$.*

We can now formulate our theorem.

▶ **Theorem 23** (The relative Higman embedding theorem for subshifts). *Let X be a subshift over an alphabet A and Y be a subshift over an alphabet B disjoint from A.*

Then $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$ iff there exists a subshift U of finite type over Y such that $X \sqsubseteq U$.

Let $Y = \emptyset$ be the empty subshift over the alphabet $\{0\}$. Then a subshift of finite type over Y is exactly the same as a subshift of finite type. Furthermore $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$ means that $\mathcal{L}(X)^c$ is enumeration reducible over the full set, which is equivalent to saying that $\mathcal{L}(X)^c$ is recursively enumerable. In the case $Y = \emptyset$ this theorem is therefore equivalent to Theorem 15. Before going into the proof, we will give a few remarks.

First we want to state that this result is very similar, but incompatible with a result of Aubrun and Sablik[1]. The result of Aubrun and Sablik states that X can be obtained from Y using some operations (very similar to ours) iff $\mathcal{L}(X)^c \leq_s \mathcal{L}(Y)^c$ where $\leq_s$ is strong enumeration reducibility [13]. It turns out that there are many mistakes in the proofs so that the result as stated in their paper is actually provably wrong (the authors have been contacted and a corrigendum is being worked on). Problems arise in both directions in the proof. First, if X can be obtained from Y, then it is not true that $\mathcal{L}(X)^c \leq_s \mathcal{L}(Y)^c$. The authors use in their proof a lot of dovetailing arguments, but dovetailing arguments cannot be used for their reduction $\leq_s$. As an example, $A \leq_s B$ does not imply $A \times A \leq_s B$ or $A^\star \leq_s B$ [23]. In fact, the smallest reducibility relation that contains $\leq_s$ and that satisfy these statements *is* the reduction $\leq_e$ we used [23]. There are also some mistakes in the reverse direction that have been patched in Aubrun's PhD thesis, but only for the case of mixing subshifts. In fact, the set of operations the authors were taking is not sufficient to do the operations for general subshifts.

Proof. For simplicity, we focus on the case where the two subshifts are one-dimensional. Let $X \subseteq A^{\mathbb{Z}}$ and $Y \subseteq B^{\mathbb{Z}}$ be subshifts.

$\Leftarrow$: It is clear that if there exists U of finite type over Y with alphabet $C \subseteq A \cup B$ such that $\mathcal{L}(X) = \mathcal{L}(U) \cap A^\star$ then $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$: it is clear that $\mathcal{L}(X)^c \leq_e \mathcal{L}(U)^c$, so we only need to prove that $\mathcal{L}(U)^c \leq_e \mathcal{L}(Y)^c$. Take an enumeration of $\mathcal{L}(Y)^c$ since U is of finite type over Y, any pattern not in $\mathcal{L}(Y)$ is not in $\mathcal{L}(U)$ and furthermore, to determine that a pattern p is in $\mathcal{L}(U)^c$, by compactness, one only needs to find some size at which it is impossible to form a valid pattern with p in its center. The procedure is the following: for every k, enumerate the k first patterns of $\mathcal{L}(Y)^c$ and check for all radiuses smaller than k whether each extension of p to this radius contains either some forbidden pattern enumerated this far or one of the patterns defining U from Y (which are in finite number). If there exists such a radius, it will be found at some step, p is then added to the enumeration. Thus $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$.

$\Rightarrow$: Assume $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$, we will construct a 2D subshift U effectively closed over Y such that $\mathcal{L}(X) = \mathcal{L}(U) \cap A^\star$, the result then follows by applying Corollary 22. In order to achieve this, from Y we will construct two intermediary subshifts:

- First we will construct Y_L: a $2D$ subshift in which the language of Y will be arranged in a dyadic like fashion. This subshift will be effective in Y. This subshift will serve as an "oracle" allowing to know whether a pattern is or is not in Y in a bounded manner: one configuration at least will contain all the patterns appearing in Y in bounded windows with computable sizes.
- From Y_L we can then construct U, in which one row out of two will be identical and belong to X and one row out of two will be in Y_L. This subshift is obtained by adding a recursively enumerable set of forbidden patterns and is thus effective over Y. By restricting the alphabet of this subshift we obtain X.

Let us now describe more precisely the different intermediate subshifts and how they are constructed, starting with Y_L:

- While Y is one dimensional, Y_L consists of two dimensions, its alphabet is $\Sigma_{Y_L} = B \cup \{\#\}$, with $\#$ a special symbol not belonging to B.

  Y_L will consist of rows, each of which will have a type: an integer $n \in \mathbb{N} \cup \{\infty\}$. A row of type $i \neq \infty$ has to be periodic. One row out of two will be of type 1, one row out of two in the remaining ones will be of type 2 and so on.

Let us define inductively a row of type i: a row of type i consists of a sequence of $|B|^{2^i-1}$ words of length 2^i-1 each, separated by $\#$ and repeated periodically. All rows of some type in a configuration must be identical and a word appearing in a row of type i must be a subword of some word of a row of type $(i+1)$, see Figure 5. Thus, a word of some type i must appear as a subword of some word in a line of type k for any $k > i$ which does not contain a forbidden pattern of Y and thus appears in some configuration of Y.

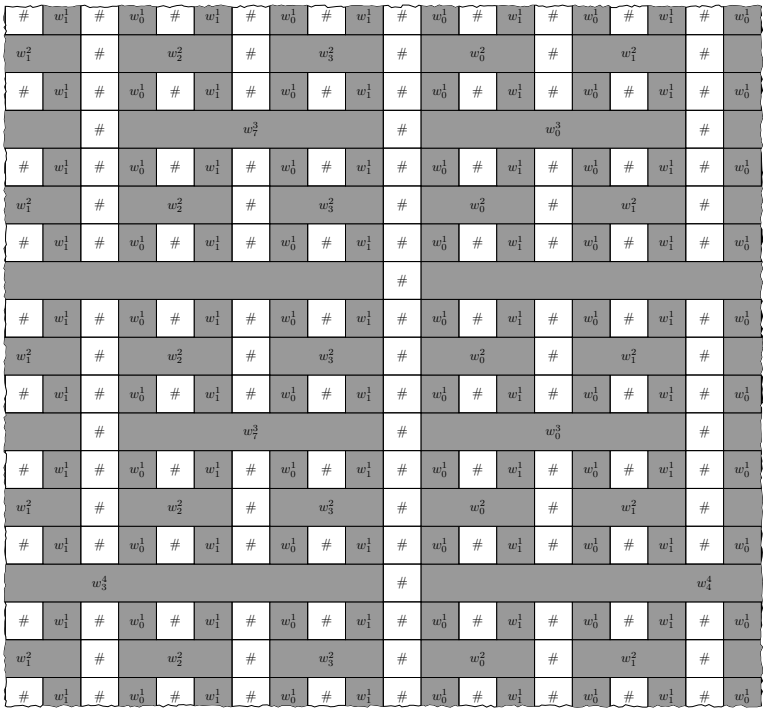


Figure 5 A typical point of Y_L : each line of type i is periodic of period $|B|^{2^{i-1}} \cdot 2^i$ and each word w_k^i in included in some word $w_{k'}^j$ for all $j > i$.

Thus Y_L is a 2D arrangement of words of $\mathcal{L}(Y)$ in a uniformly recurrent way, and there exists at least one configuration containing all of $\mathcal{L}(Y)$. Furthermore, Y_L is effective over Y.

We describe how to construct U from Y_L: we know that $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$. Thus, there exists a computable $f : \mathbb{N} \times \mathbb{N} \to \mathfrak{P}_{finite}(\mathbb{N})$ such that:

$$x \in \mathcal{L}(X)^c \text{ iff } \exists n \in \mathbb{N}, f(x,n) \subseteq \mathcal{L}(Y)^c$$

In other words, some word w is in $\mathcal{L}(X)$ iff for any $n \in \mathbb{N}$, there is some word of $f(x,n)$ in $\mathcal{L}(Y)$. That is to say, supposing $\mathcal{L}(Y)$ is given as an oracle, we have an enumerable way to check that a word w belongs to $\mathcal{L}(X)$: enumerate the $n \in \mathbb{N}$ and compute $f(w,n)$ and check that at least one element belongs to $\mathcal{L}(Y)$, if not halt. The computations that do not halt are the ones where w belongs to $\mathcal{L}(X)$.
Given Y_L, this can be implemented in an effective way: take $x \in A^{\mathbb{Z}}$ and $y \in Y_L$. We interleave x in y by using the same technique as in figure 4: we insert a copy of x between each pair of lines of y.
We now need to ensure that all words on the lines with alphabet A belong to $\mathcal{L}(X)$. This may also be done by adding a recursively enumerable set of forbidden patterns: in order to check that some subword w of x is in $\mathcal{L}(X)$, one needs to check that for each n,

$f(w, n)$ appears in some line of type $i > |w|$: for every pattern w we forbid all patterns that contain w but no pattern of $f(w, n)$, since rows of type i appear every 2^{i+2} rows, for each w this constitutes a finite number of forbidden patterns for each n. Thus we may recursively enumerate the forbidden patterns for each $w \in A^\star$. ◀

3.3 The Boone-Higman-Thompson theorem

The Boone-Higman-Thompson theorem is a theorem that characterizes groups with a *computable word problem*. It turns out that the characterization is obtained with the notions of a simple group:

▶ **Theorem 24** (The Boone-Higman-Thompson theorem [6, 26]). *A group G has a computable word problem iff it is a subgroup of a simple recursively presented group.*

Recall that a simple group is a group with no proper (nontrivial) quotient. By Dictionary 1, the equivalent should be a subshift with no proper (nontrivial) subshift, i.e. what is called in the literature a minimal subshift. This seems to be indeed, the good analogy, as argued for in [18], and we will prove:

▶ **Theorem 25.** *Let X be a 1 dimensional subshift over an alphabet Σ. Then X has a computable language iff there exists a two dimensional minimal effective subshift Y over an alphabet $\Gamma \supset \Sigma$ such that $X \sqsubseteq Y$.*

Recently, Durand and Romashchenko [11] have proved that given a d-dimensional minimal effectively closed subshift, it can be realized as a subaction of a $(d+1)$-dimensional minimal SFT:

▶ **Theorem 26** ([11]). *Let X be a minimal effectively closed subshift. There exists a minimal SFT Y such that X is a subaction of Y: X is the projection by a letter to letter map of the lines of Y.*

This together with Theorem 25 gives us the subshift counterpart to the Boone-Higman-Thompson theorem:

▶ **Corollary 27** (The Boone-Higman-Thompson theorem for subshifts). *Let X be a 1 dimensional subshift over an alphabet Σ. Then X has a computable language iff there exists a three dimensional minimal subshift of finite type Y over an alphabet $\Gamma \supset \Sigma$ s.t. $X \sqsubseteq Y$.*

Both Theorem 25 and Corollary 27 translate to higher dimensions, the details of the proofs are left to the reader.

Before proving the Theorem 25, one needs a good intuition on what a minimal subshift looks like. Minimal subshifts are defined as subshifts that do not contain any nontrivial subshifts, but an equivalent, more palatable definition, is that minimal subshifts are uniformly recurrent subshifts, that is subshifts X where, for every pattern $u \in \mathcal{L}(X)$, there exists a size n s.t. the pattern u occurs in every pattern of X of size n. In particular, all configurations x of X have the same patterns, and every pattern that appear should appears everywhere, i.e. in any sufficiently large part of x.

We now proceed to the proof of the theorem. One direction is well known: A minimal effectively closed subshift has a computable language, see [4] for example. Therefore $\mathcal{L}(Y)$ is computable and therefore $\mathcal{L}(Y) \cap \Sigma^\star$ is computable.

The other direction essentially amounts to the following: Given a set of patterns L on an alphabet A, find a *minimal* subshift X that contains all patterns of L (and other patterns). Before reading the proof, the reader should try by itself as an exercise to find a two-dimensional minimal subshift X over an alphabet $\{a, b, c\}$ that contains all one-dimensional words over the alphabet $\{a, b\}$.

Our proof is quite similar to a construction by Elek and Monod [12] of a subshift with a non-amenable topological full group. Our construction is done however with more care to ensure that everything we are doing remains computable and that our subshift is already minimal, but the idea is essentially the same.

Let us now start with a 1 dimensional subshift X over an alphabet Σ with a computable language.

We define recursively a set $(w^i)_{i\in\mathbb{N}}$ of biinfinite rows. Each row will be periodic. We will denote by p_i the period of the row and by v_i the word that repeats, so that v_i is of length p_i and for all $k \in \mathbb{Z}$, $(w^i)_k = v_{k \mod p_i}$.

The row w^0 is the row of period $p_1 = 1$ corresponding to the word $v_0 = \#$. Suppose the row w^n is given, of period p_n.

Let $\{u_1, u_2 \dots u_{k_{n+1}}\}$ be the (computable) list of all words of length $2p_n - 1$ that appear in X. We define v_{n+1} to be the word consisting of all possible pairs of words of size $2p_n - 1$, separated by the # symbol

$$\#u_1\#u_1\#u_1\#u_2\#u_1\#u_3 \dots u_{k_{n+1}}\#u_{k_{n+1}-1}\#u_{k_{n+1}}\#u_{k_{n+1}}$$

and w_{n+1} is the biinfinite word where v_{n+1} repeats periodically. Notice that v_{n+1} is of size $p_{n+1} = 2k_{n+1}^2 p_n$ so that p_{n+1} is strictly greater than p_n and p_n divides p_{n+1} .

We repeat some properties of our set of rows:

- The row w^n is periodic of period p_n. Furthermore the symbol # appears in w^n only in positions multiple of $2p_{n-1}$.
- p_i divides p_j if $i < j$.
- $p_n > n$.

▶ **Lemma 28.** *Let u be a word of length k that appears in w^n for $n \geq k$ in position i. Then u appears in position $i + tp_{k-1}$ in w^k for some integer t.*

Proof. The result is clear for $n = k$. Now suppose that $n > k$. There are two cases for u: either $u = s_1\#s_2$ for two words $s_1, s_2 \in \Sigma^\star$ or $u = s$ for some word $s \in \Sigma^\star$.

We start with the first case, $u = s_1\#s_2$. The words s_1 and s_2 are words of size $< k$ that are factors of some word of size $2p_{(n-1)} - 1$ that appears in X. Therefore there are also respectively suffix and prefix of some words t_1, t_2 that appear in X, each of size $2p_{(k-1)} - 1$. By definition $t_1\#t_2$ appears in w^k therefore u appears in w^k. As every symbol # inside w^k appears at positions that are multiples of $2p_{k-1}$ and that it is also the case inside w^n (as p_k divides p_{n-1}), the position where u appears in w^k must be of the form $i + tp_{k-1}$ for some t.

Now the second case. Suppose that $u = s$ for some word s that appears in X of size k. u appears from position i to position $i + k - 1$ in w^n. Let $0 \leq j < p_{k-1}$ so that $j = i - 1 \mod p_{k-1}$. u is a word of size k that appears in X and therefore can be completed as a word v of size $2p_{k-1} - 1$ that appears in X by adding j letters at the beginning and $2p_{k-1} - 1 - (j + k)$ letters at the end. This word v appears at position $tp_{k-1} + 1$ in w^k for some t and therefore u appears in position $tp_{k-1} + 1 + j = t'p_{k-1} + i$ in w^k. ◀

▶ **Definition 29.** *If i is an integer, the* level *of i, denoted by $lvl(i)$ is the greatest power of 2 that divides i, i.e. $i = k \times 2^{lvl(i)}$ with k odd. The level of 0 is $+\infty$ by convention.*

The two following lemmas are clear.

▶ **Lemma 30.** *Let $n > k$, then $lvl(i + 2^n) = lvl(i)$.*

▶ **Lemma 31.** *Let $i \neq j$ s.t. $lvl(i) \geq k$ and $lvl(j) \geq k$. Then $|i - j| \geq 2^k > k$.*

We now define a configuration y in the following way: the i-th row of y is the row w^j where j is the level of i.

For $i = 0$, we take any word w that is a limit point of $\{w^j, j \in \mathbb{N}\}$.

Notice that y is likely not computable, as the row 0 might be arbitrarily complex. However all other rows are computable.

To simplify notations, we will denote the rows of y in exponent, so that the symbol in the i-th row and j-th column of y is y^i_j and the i^{th} row of y is y^i. By definition, we therefore have $y^i_j = w^{lvl(i)}_j$ for $i \neq 0$.

▶ **Lemma 32.** *Let u be a pattern defined over $[1,k] \times [1,k]$ that appears in y.*

Then u also appears inside y at position $(i+1, j+1)$ with $i \in [0, 2^k - 1]$ and therefore u appears inside the first $2^{k+1} - 1$ rows of y (in the rows labeled 1 to $2^{k+1} - 1$).

Proof. Let u be a pattern defined on the square $[1,k] \times [1,k]$.

Suppose that u appears inside y at position $(i+1, j+1)$. That is: for all $(l,m) \in [1,k]^2, u^m_l = y^{i+m}_{j+l}$.

There are two cases. First, suppose that all of the integers $i+1, i+2, \dots i+k$ are of level strictly less than k. Then for all $l \in [i+1, i+k]$ and all integers t, $y^{l+2^k t} = y^l$. We can therefore suppose wlog that $i \in [0, 2^{k-1}]$ and the result is proven.

Otherwise some of the integers $i+1, \dots i+k$ is of level at least k. By Lemma 31, this happens only for one of the integers, say the integer $i + r = z \times 2^n$ for $n \geq k$.

The word u^r appears by definition in y^{i+r} which is of level at least k. By Lemma 28, it also appears in w^k at position $j + 1 + tp_{k-1}$ and therefore in y^{2^k} at position $j + 1 + tp_{k-1}$. (Lemma 28 also applies if $i + r = 0$, as the row 0 of u is the limit of rows of arbitrary large level). In other words for all $l \in [1,k], u^r_l = y^{2^k}_{j+tp_{k-1}+l}$.

We now claim that the word u appears in position $[2^k - r + 1, j + 1 + tp_{k-1}]$ inside y. That is, for all $l, m \in [1,k]^2, u^m_l = y^{2^k - r + m}_{j+tp_{k-1}+l}$. The result is clear for $m = r$. Now let $m \in [1,k], m \neq r$. As $i + m$ is of level strictly less than k, $y^{i+m+t2^k} = y^{i+m}$ for all t by Lemma 30. In particular as $i + r$ is dividible by 2^k, we get that $y^{i+m} = y^{i+m+2^k-(i+r)} = y^{2^k - r + m}$. Furthermore the row y^{i+m} is periodic of period p_s for some $s < k$ and in particular it is periodic of period p_{k-1}. Therefore

$$u^m_l = y^{i+m}_{j+l} = y^{i+m}_{k+l+tp_{k-1}} = y^{2^k - r + m}_{k+l+tp_{k-1}}$$ ◀

▶ **Corollary 33.** *Let u be a pattern of size $k \times k$ inside y. Then u appears in any window of size $2^{k+2} \times 2p_k$.*

Proof. Indeed, by the previous lemma, u appears inside y in position (i, j) for some $i \in [1, 2^k]$ However the rows from 1 to $2^{k+1} - 1$ are all periodic of period p_k, and repeat vertically with period 2^{k+1} by Lemma 30. Therefore the pattern u itself repeats horizontally with period p_k and repeats vertically with period 2^{k+1} and consequently appears in any window of size $2^{k+2} \times 2p_k$. ◀

▶ **Corollary 34.** *Let Y be the subshift that forbids all patterns of size $k \times k$ that do not appear in the square $[1, 2^{k+2}] \times [1, 2p_k]$ of y.*

Then Y is minimal, effectively closed, and $\mathcal{L}(X) = \mathcal{L}(Y) \cap \Sigma^\star$.

Proof. The first conclusions are immediate from the previous corollary. The second one comes from the fact that y (apart from the row 0) is computable. The third one is by definition of y. ◀

4 Discussion

In this article, we have introduced the analogues of the three Higman theorems, originally for groups, in terms of subshifts. This reinforces the convictions of the authors that symbolic dynamics has a deep connection with objects from combinatorial algebra. To obtain these theorems, we had to introduce the following concepts:

- An equivalent of the notion of free product for groups (Definitions 12 and 13).
- An equivalent of the notion of subgroup containement (Definition 9).

Compared to existing constructions, these two new ideas are rather *combinatorial* rather than *dynamic*. In particular, they cannot be defined easily in terms of the infinite words in the subshifts; they are defined in terms of the finite words that constitute the language of the subshift. This could be seen as a drawback of the construction, so we will give some arguments explaining why more "dynamical" constructions cannot work.

The concept of the free product of subshifts is used for the relativized Higman theorem. This operation does not appear in the work of Aubrun and Sablik [1] (which was flawed as we saw) and is probably mandatory; Suppose that a minimal subshift X is defined from a subshift Y by various dynamical constructions (cartesian product, factor, subactions, etc) as in [1]. Let Y' be the smallest subshift of Y that contains all uniformly recurrent points of Y. Then it is easy to see that Y' also defines X using the same constructions. However Y' may have very different computability properties than Y. In particular it is possible to have $\mathcal{L}(X)^c \leq_e \mathcal{L}(Y)^c$ but $\mathcal{L}(X)^c \not\leq_e \mathcal{L}(Y')^c$.

In fact, when looking at our whole construction (and the construction of [1]), we see that it is important to start with a subshift X with the property that, for every finite collection of words $u \in \mathcal{L}(X)$, there exists a *uniformly recurrent* point of X that contains all of them. So we either have to assume that our subshift has this property (this is what Aubrun did in her PhD thesis, by assuming some mixing properties), or use a free product: The free product of X with any (nonempty) subshift always has this property.

The full restriction operator $\sqsubseteq$, the analogue of subgroup containment, is mostly used in the equivalent of the Boone-Higman-Thompson theorem. In fact, it can be replaced in the other theorems by more traditional dynamical operators, like factor maps (the original Hochman theorem was indeed stated in terms of factor maps). It is not clear if we could obtain a analogue of Boone-Higman-Thompson theorem in terms of factor map. It is certainly true that factors of minimal subshifts of finite type have a computable language; However, they also have the additional property that the set of their uniformly recurrent points is dense, and therefore not all subshifts with computable language can be obtained this way.

References

1 Nathalie Aubrun and Mathieu Sablik. An Order on Sets of Tilings Corresponding to an Order on Languages. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, pages 99–110, 2009.

2 Nathalie Aubrun and Mathieu Sablik. Simulation of Effective Subshifts by Two-dimensional Subshifts of Finite Type. *Acta Applicandae Mathematicae*, 126(1):35–63, 2013. doi:10.1007/s10440-013-9808-5.

3 Alexis Ballier. *Propriétés structurelles, combinatoires et logiques des pavages*. PhD thesis, Aix-Marseille Université, 2009.

4 Alexis Ballier and Emmanuel Jeandel. Computing (or not) Quasi-periodicity Functions of Tilings. In Jarkko Kari, editor, *Second Symposium on Cellular Automata "Journées Automates Cellulaires", JAC 2010, Turku, Finland, December 15-17, 2010. Proceedings*, pages 54–64. Turku Center for Computer Science, 2010.

5 Robert Berger. *The Undecidability of the Domino Problem*. Number 66 in Memoirs of the American Mathematical Society. The American Mathematical Society, 1966.

6 William W. Boone and Graham Higman. An algebraic characterization of groups with soluble word problem. *Journal of the Australian Mathematical Society*, 18(1):41–53, August 1974. doi:10.1017/S1446788700019108.

7 Mike Boyle, Ronnie Pavlov, and Michael Schraudner. Multidimensional sofic shifts without separation and their factors. *Transactions of the AMS*, 362(9):4617–4653, September 2010. doi:10.1090/S0002-9947-10-05003-8.

8 W. Craig and R. L. Vaught. Finite Axiomatizability Using Additional Predicates. *The Journal of Symbolic Logic*, 23(3):289–308, September 1958.

9 Bruno Durand, Andrei Romashchenko, and Alexander Shen. Effective Closed Subshifts in 1D Can Be Implemented in 2D. In *Fields of Logic and Computation*, number 6300 in Lecture Notes in Computer Science, pages 208–226. Springer, 2010. doi:10.1007/978-3-642-15025-8_12.

10 Bruno Durand, Andrei Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78(3):731–764, May 2012. doi:10.1016/j.jcss.2011.11.001.

11 Bruno Durand and Andrei E. Romashchenko. On the expressive power of quasiperiodic SFT. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 1–14, 2017.

12 Gábor Elek and Nicolas Monod. On the Topological Full Group of a Minimal Cantor $\mathbb{Z}^2$-System. *Proceedings of the American Mathematical Society*, 141(10):3549–3552, October 2013.

13 Richard M. Friedberg and Hartley Rogers. Reducibility and Completeness for Sets of Integers. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 5:117–125, 1959.

14 Graham Higman. Subgroups of Finitely Presented Groups. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 262(1311):455–475, August 1961.

15 Graham Higman and Elizabeth Scott. *Existentially Closed Groups*. Oxford University Press, 1988.

16 Michael Hochman. On the dynamics and recursive properties of multidimensional symbolic systems. *Inventiones Mathematicae*, 176(1):2009, April 2009.

17 Michael Hochman and Tom Meyerovitch. A characterization of the entropies of multidimensional shifts of finite type. *Annals of Mathematics*, 171(3):2011–2038, May 2010. doi:10.4007/annals.2010.171.2011.

18 Emmanuel Jeandel. Enumeration Reducibility in Closure Spaces with Applications to Logic and Algebra. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–11, 2017.

19 Emmanuel Jeandel and Pascal Vanier. Characterizations of periods of multidimensional shifts. *Ergodic Theory and Dynamical Systems*, 35(2):431–460, April 2015. doi:10.1017/etds.2013.60.

20 Aimee Johnson and Kathleen Madden. Factoring higher-dimensional shifts of finite type onto the full shift. *Ergodic Theory and Dynamical Systems*, 25:811–822, 2005.

21 S.C. Kleene. *Two Papers on the Predicate Calculus*, chapter Finite Axiomatizability of Theories in the Predicate Calculus Using Additional Predicate Symbols, pages 31–71. Number 10 in Memoirs of the American Mathematical Society. American Mathematical Society, 1952.

22 R.C. Lyndon and P.E. Schupp. *Combinatorial Group Theory*. Classics in Mathematics. Springer Berlin Heidelberg, 2001.

23 Daniele Marsibilio and Andrea Sorbi. Bounded Enumeration Reducibility and its degree structure. *Archive for Mathematical Logic*, 51:163–186, 2012.

24 Mark Sapir. *Combinatorial Algebra: Syntax and Semantics*. Springer Monographs in Mathematics. Springer, 2014.

25 Stephen G. Simpson. Mass problems associated with effectively closed sets. *Tohoku Mathematical Journal*, 63(4):489–517, 2011.

26 Richard J. Thompson. Embeddings into Finitely Generated Simple Groups which Preserve the Word Problem. In Sergei I. Adian, William W. Boone, and Graham Higman, editors, *Word Problems II*, volume 95 of *Studies in Logic and the Foundations of Mathematics*, pages 401–441. North Holland, 1980.

Lower Bounds for DeMorgan Circuits of Bounded Negation Width

Stasys Jukna
Institute of Computer Science, Goethe University Frankfurt, Frankfurt am Main, Germany
Institute of Data Science and Digital Technologies, Vilnius University, Lithuania
http://www.thi.informatik.uni-frankfurt.de/~jukna/
stjukna@gmail.com

Andrzej Lingas
Department of Computer Science, Lund University, Box 118, 22100 Lund, Sweden
http://fileadmin.cs.lth.se/cs/Personal/Andrzej_Lingas/
Andrzej.Lingas@cs.lth.se

Abstract

We consider Boolean circuits over $\{\vee, \wedge, \neg\}$ with negations applied only to input variables. To measure the "amount of negation" in such circuits, we introduce the concept of their "negation width." In particular, a circuit computing a monotone Boolean function $f(x_1, \ldots, x_n)$ has negation width w if no nonzero term produced (purely syntactically) by the circuit contains more than w distinct negated variables. Circuits of negation width $w = 0$ are equivalent to monotone Boolean circuits, while those of negation width $w = n$ have no restrictions. Our motivation is that already circuits of moderate negation width $w = n^\epsilon$ for an arbitrarily small constant $\epsilon > 0$ can be even exponentially stronger than monotone circuits.

We show that the *size* of any circuit of negation width w computing f is roughly at least the minimum size of a monotone circuit computing f divided by $K = \min\{w^m, m^w\}$, where m is the maximum length of a prime implicant of f. We also show that the *depth* of any circuit of negation width w computing f is roughly at least the minimum depth of a monotone circuit computing f minus $\log K$. Finally, we show that *formulas* of bounded negation width can be balanced to achieve a logarithmic (in their size) depth without increasing their negation width.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases Boolean circuits, monotone circuits, lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.41

Funding *Stasys Jukna*: Research supported by the DFG grant JU 3105/1-1 (German Research Foundation).
Andrzej Lingas: Research supported in part by VR grant 2017-03750 (Swedish Research Council).

1 Introduction

Understanding the power of negations in computations is one of the most basic objectives in computational complexity. While strong, even exponential, lower bounds for explicit monotone Boolean functions are already known for *monotone* Boolean $\{\vee, \wedge\}$ circuits, we can currently prove only depressingly small (linear) lower bounds on the size of $\{\vee, \wedge, \neg\}$ circuits when there are no restrictions on the number or the usage of negation gates.

In this paper, we concentrate on *DeMorgan circuits*, that is, on $\{\vee, \wedge, \neg\}$ circuits with fanin-2 OR and AND gates, and with negation applied only to input variables. In other words, a DeMorgan circuit is a circuit with fanin-2 OR and AND gates, while inputs are variables $x_1, \ldots, x_n$ and their negations $\overline{x}_1, \ldots, \overline{x}_n$; to simplify notation, we will write $\overline{x}_i$ instead of $\neg x_i$. DeMorgan circuits are sometimes called *normalized* circuits [17], *standard* circuits [31, Section 6.13] or circuits *with tight negations* [25]. A circuit is a *formula* if its underlying graph is a tree. A *monotone circuit* is a DeMorgan circuit with no negated input

variables at all. By just doubling the circuit size and using DeMorgan rules, any circuit over $\{\vee, \wedge, \neg\}$ of size s can be converted to a DeMorgan circuit computing the same function and having size at most $2s$ (see, for example, [6, Theorem 3.1]).

The effect of negations on the size or depth of $\{\vee, \wedge, \neg\}$ circuits was mainly considered by either restricting the total *number* of used negation gates, or by restricting the *usage* of negations. There is an extensive literature on the research in the first direction, when the total number of NOT gates is bounded; here negations can be applied not only to input variables. We refer to [13, Chapter 10] and the papers cited therein for this line of research; see also [26, 3, 9] for more recent developments in this direction.

Another line of research (which attracted much less attention, and which we follow in this paper) was to restrict the "amount of negation" in circuits. One of the first results in this direction was proved by Raz and Wigderson [21, Theorem 4.1]: if $w \leq n^{2-\epsilon}$ for a constant $\epsilon > 0$, then any DeMorgan circuit with at most w negated input variables computing the s-t connectivity function of n-vertex graphs must have depth $\Omega(\log^2 n)$. Guo et al. [9] have proved that any DeMorgan circuit with at most w negated input variables computing a monotone Boolean function f must have depth at least the monotone circuit depth of f minus w. Koroth and Sarma [16] relax this restriction (on the total number of allowed negated input variables), and say that a (not necessarily DeMorgan) circuit over $\{\vee, \wedge, \neg\}$ has *orientation weight* w if the function computed at each gate is monotone in all but at most w variables. They prove that the depth of any circuit over $\{\vee, \wedge, \neg\}$ of orientation weight w computing a monotone function f is at least the minimum depth of a monotone circuit computing f divided by $4w + 1$.

In this paper, as the measure of the "amount of negation" in DeMorgan circuits, we consider their "negation width" (see Definition 1.2 below). This measure (without calling it the negation width) was already considered by Amano and Maruoka [2, Sect. 4]. They used a modification of Razborov's Method of Approximation [23, 24] to show that DeMorgan circuits of small negation width for the Clique function must still be large (we recall their result right before Corollary 6.2). Our main results (Theorems 1.6 and 1.8) give a general reduction of DeMorgan circuits of bounded negation width to monotone circuits, from which the bound of [2], as well as new lower bounds, follow (see Section 6).

▶ Notation. We use standard terminology regarding Boolean functions (see, for example, [31]). In particular, a *term* is an AND of *literals*, each being a variable or its negation. The *length* of a term is the number of distinct literals in it. A term is a *zero term* if it contains a variable and its negation. An *implicant* of a Boolean function $f(x_1, \ldots, x_n)$ is a nonzero term p such that $p \leq f$ holds, that is, $p(a) \leq f(a)$ holds for all $a \in \{0,1\}^n$. An implicant of f is a *prime implicant* of f if after the removal of all occurrences of any single literal in p the resulting term is not an implicant of f anymore. The set of all prime implicants of f will be denoted by $PI(f)$. A Boolean function f is *monotone* if $a \leq b$ implies $f(a) \leq f(b)$. Note that if f is monotone, then all prime implicants of f are positive, that is, consist solely of not negated variables.

1.1 Negation width of circuits

Our goal is to understand to what extent the usage of negated input variables can decrease the size or the depth of DeMorgan circuits computing *monotone* Boolean functions. As a measure of the "amount of negation" in DeMorgan circuits, we will use their "negation width." This measure is motivated by a trivial fact that every DeMorgan circuit not only computes a particular Boolean function but also *produces* (purely syntactically) some set of terms in a natural way.

▶ **Definition 1.1** (Terms produced by circuits). *The set of terms produced at an input gate holding a literal z is a singleton-set $\{z\}$. The set of terms produced at an OR gate is a union of sets of terms produced at its two inputs, while the set produced at the AND gate is obtained by taking the AND of every term produced at one of its inputs with every term produced at the other input.*

The set $T(C)$ of terms produced by the entire circuit C is the set of terms produced at the output gate of C. During the production of terms, we use the "shortening" axiom $x \wedge x = x$, but do not use the "annihilation" axiom $x \wedge \overline{x} = 0$. So, $T(C)$ can contain also zero terms, those having a variable and its negation.[1] Easy induction on the circuit size shows that the Boolean function f computed by a circuit C is the function computed by the OR of all terms produced by C.

If the circuit C is *monotone* (has no negated inputs at all), then we clearly have $PI(f) \subseteq T(C)$, that is, every prime implicant of f must then be produced by the circuit. But even then, the equality $T(C) = PI(f)$ does not need to hold: already in 1981, Okol'nishnikova [18] exhibited an explicit monotone Boolean function f of n variables which can be computed by a monotone circuit of size $O(n)$, but any monotone circuit C satisfying $T(C) = PI(f)$ must have $2^{\Omega(n^{1/4})}$ gates.

The situation when the computed by the circuit C function f is monotone, but the circuit C itself is *not* necessarily monotone, is even more subtle: then even the inclusion $PI(f) \subseteq T(C)$ does not need to hold. For example, the function $f = x \vee y$ is computed by a circuit $C = x\overline{y} \vee y$, but $T(C) = \{x\overline{y}, y\}$ whereas $PI(f) = \{x, y\}$.

However, we have the following simple and well-known property of (not necessarily monotone) DNFs computing monotone Boolean functions (see, for example, [5, Theorem 1.24 on p. 37]): if D is a (not necessarily monotone) DNF computing a monotone Boolean function f, then the monotone DNF obtained from D by first removing all zero terms, and then removing all occurrences of negated variables from the remaining terms, also computes f. This, in particular, implies that the obtained monotone DNF must contain all prime implicants of f.

If C is a DeMorgan circuit computing f, then the OR of terms in $T(C)$ computes f. So, by the aforementioned fact [5, Theorem 1.24 on p. 37], for every prime implicant p of f, the set $T(C)$ must contain either p itself or at least one *extension* of p, that is, a nonzero term of the form $p \cdot r$, where the term $r = \overline{x}_{i_1} \cdots \overline{x}_{i_l}$ consists solely of negated variables. This motivates the following measure of DeMorgan circuits computing monotone Boolean functions.

▶ **Definition 1.2** (Negation width). *A DeMorgan circuit computing a monotone Boolean function f has* negation width w *if for every prime implicant p of f, the circuit produces either p itself or some its extension containing at most w negated variables.*

There are no other restrictions on the remaining produced terms, except the trivial one that the function computed by the OR of all produced terms must coincide with f. Note that the negation width w of any DeMorgan circuit computing f satisfies $0 \leq w \leq n - m$, where m is the minimum length of a prime implicant of f. Also, minimal circuits of negation width $w = 0$ are monotone circuits: just replace each negated input gate $\overline{x}_i$ by constant 0.

[1] At a "functional" level, zero terms are redundant: they contribute nothing to the values of the computed function. The only reason to keep them in $T(C)$ is to ensure that "syntactical" changes of circuits (replacements of some input gates by constants), which we will latter make, do not turn some previously zero terms into nonzero terms.

Examples of sufficient conditions for a circuit to have negation width at most w are any of the following.

- The circuit has at most w negated input variables; such circuits were considered, for example, by Raz and Wigderson [21], and Guo et al.[9].
- No input-output path has more than[2] $\log w$ AND gates; such circuits computing quadratic forms (multi-output functions) were considered in [17].
- No nonzero term produced by the circuit contains more than w distinct negated variables. Note that this restriction is a relaxation of both two previous restrictions.

None of these sufficient conditions is necessary. In particular, the negation width puts no restrictions on the length of produced *zero* terms. So, at intermediate gates, the circuit can produce very long terms, and then cancel them (turn them into zero terms). At this point, it is worth to mention that DeMorgan circuits computing monotone Boolean functions f *more efficiently* than monotone circuits *must* use cancellations (must produce zero terms): otherwise, we could just replace all negated input variables by constants 1, and the resulting monotone circuit would still compute f.

We shall also consider DeMorgan circuits of bounded *average* negation width. Let C be a DeMorgan circuit computing a monotone Boolean function f.

▶ **Definition 1.3** (Average negation width)**.** *The* negation width *of a prime implicant $p \in PI(f)$ in the circuit C is the minimum number w such that $T(C)$ contains an extension of p with at most w negated variables. The* average negation width *of the circuit C is the average, over all prime implicants $p \in PI(f)$, of the negation width of p in C.*

Note that a circuit C computing f has negation width w if *every* prime implicant of f has negation width at most w in C. Average negation width relaxes this "every" requirement.

1.2 Motivation

Our motivation to consider circuits of bounded negation width w is that allowance of even moderately large negation width $w = n^\epsilon$ for an arbitrarily small constant $\epsilon > 0$ *can* substantially reduce the size of monotone circuits.

▶ **Example 1.4.** The *triangle function* CLIQUE$(n, 3)$ has one variable for every edge of the complete graph K_n on $\{1, \ldots, n\}$, and accepts a subgraph G of K_n if and only if G contains a triangle. It is known that this function requires monotone circuits of almost cubic size $n^{3-o(1)}$ [23, 1]. According to Claim A.3 in Appendix A, the function can be computed in already *sub-cubic* size $n^{3-\epsilon/4}$ if negation width $w = n^\epsilon$ is allowed.

▶ **Example 1.5.** The threshold-k function Th_k^n accepts a Boolean input of length n if and only if it contains at least k ones. The smallest known monotone circuits for Th_k^n have size of order $n \log k$ (see, for example, [15]). On the other hand, for $k \le n^{1/3}$, the function Th_k^n *can* be computed by a DeMorgan circuit of *linear* size $O(n)$ if negation width $w = k^3$ is allowed (see Claim A.5 in Appendix A.3).

Using monotone circuit lower bounds of Razborov [24] and Tardos [29], one can show that, on some monotone Boolean functions, super-polynomial, and even exponential gaps between the size of monotone circuits and circuits of moderate negation width can be achieved; see Examples A.1 and A.2 in Appendix A. We are not aware of any similar separating examples for restrictions on the use of negations considered in [21, 16, 9]: restricted number of allowed negated input variables, or restricted orientation weight.

[2] All logarithm in this paper are to the base 2.

1.3 Our contributions

Our first general result relates (non-monotone) DeMorgan circuits and formulas of bounded negation width to *monotone* circuits and formulas.

▶ **Theorem 1.6.** *Let f be a monotone Boolean function with all prime implicants of length at most m. Let s be the minimum size of a monotone circuit computing f, and d the minimum depth of such a circuit. Then any DeMorgan circuit of negation width w computing f must have size at least $s/K-1$ and depth at least $d-\log K$, where $K = 8\min\{m^w, w^m\}\cdot\log|PI(f)|$.*

This theorem allows to extend known lower bounds for monotone circuits to lower bounds for (non-monotone) DeMorgan circuits of bounded negation width. We provide several such extensions for specific monotone Boolean functions following from Theorem 1.6 in Section 6 (see Corollaries 6.1-6.5). In particular, Theorem 1.6 implies that any DeMorgan circuit of negation with $w \le n^\epsilon$ computing the triangle function CLIQUE$(n,3)$ must have $\Omega(n^{3-4\epsilon})$ gates (see Corollary 6.1). This bound is not very far from the truth because for $w = n^\epsilon$, $O(n^{3-\epsilon/4})$ gates are also sufficient (Claim A.3 in Appendix A).

Our second general result concerns circuits of bounded *average* width. It complements the general framework for converting lower bounds for monotone circuits to those for DeMorgan circuits of bounded negation width given in Theorem 1.6.

▶ **Definition 1.7.** *A monotone Boolean function h K-*approximates *a monotone Boolean function f if there is an OR g of at least a $1/K$ portion of prime implicants of f such that $g \le h \le f$ holds.*

▶ **Theorem 1.8.** *Let f be a monotone Boolean function with all prime implicants of length at most m. Let $w \ge 0$ and $K = 8\cdot\min\{m^{2w},(2w)^m\}$. If every monotone circuit K-approximating f requires at least t gates, then every DeMorgan circuit of average negation width w computing f must also have at least t gates.*

Let us note the difference between Theorems 1.6 and 1.8. The advantage of Theorem 1.6 is that one can *directly* use known lower bounds on the monotone circuit complexity of the function f *themselves*. Theorem 1.8 is more general: it applies to circuits when only the *average* negation width is bounded, and we do not have the additional $\log|PI(f)|$ factor in the "blow down" parameter K. However, in order to apply Theorem 1.8, one has to show that not only the function f itself but also any sufficiently close approximation of f requires large monotone circuits. So, one has to analyze the monotone lower bound *proofs* to ensure this latter property. We will demonstrate this by proving that every DeMorgan circuit of average negation width $w = o(\sqrt{k}/\log k)$ computing the clique function CLIQUE(n,k) must have $2^{\Omega(\sqrt{k})}$ gates (see Corollary 6.8 in Section 6).

Our third general result extends the well-known Spira's depth reduction theorem [27] to DeMorgan formulas of bounded negation width: it shows that such formulas can also be balanced *without* increasing their negation width.

▶ **Theorem 1.9.** *If a monotone Boolean function f can be computed by a DeMorgan formula of size s and negation width w, then f can be also computed by a DeMorgan formula of depth at most $3\cdot\log s$ and the same negation width w.*

The rest of the paper is organized as follows. In Section 2, a special type of "random subcircuits" is introduced. Sections 3–5 are devoted to the proof of our main results (Theorems 1.6–1.9). In Section 6, we give several applications of our general results to specific Boolean functions. Appendix A contains proofs of the upper bounds claimed in our motivating examples (Examples 1.4–A.1).

2 Random subcircuits

Let $f(x_1,\dots,x_n)$ be a monotone Boolean function, and C be a DeMorgan circuit computing f. For a subset $Y=\{x_i\colon i\in I\}$ of variables, the *monotone Y-subcircuit* of C is obtained as follows.

1. First, set to 0 all variables in Y; so, for every $i\in I$, the input gate x_i is set to 0, while the negated input gate $\overline{x}_i$ is set to 1.
2. Then replace by constant 0 each of the remaining negated input gates $\overline{x}_j$ for $j\notin I$.
3. Finally, eliminate constant input gates through repeated replacements of $0\wedge u$ by 0, $1\vee u$ by 1, and $0\vee u$, $1\wedge u$ by u.

Schematically:

$$C(x,y,\overline{x},\overline{y}) \overset{\text{Step 1}}{\mapsto} C(x,0,\overline{x},1) \overset{\text{Step 2}}{\mapsto} C(x,0,0,1) \overset{\text{Step 3}}{\mapsto} C_+(x)\,.$$

▶ **Example 2.1.** Consider the DeMorgan formula $C=(x_1\vee x_2\vee\overline{x}_3)(\overline{x}_1\vee\overline{x}_2\vee x_5)(x_3\vee x_4\vee\overline{x}_5)$, and $Y=\{x_1,x_4\}$. After the first step, we obtain the formula $(0\vee x_2\vee\overline{x}_3)(1\vee\overline{x}_2\vee x_5)(x_3\vee 0\vee\overline{x}_5)$. After the second step, we obtain the formula $(0\vee x_2\vee 0)(1\vee 0\vee x_5)(x_3\vee 0\vee 0)$ and, after the elimination of constants, the resulting monotone sub-formula of C is x_2x_3.

The following lemma is just a simple observation.

▶ **Lemma 2.2.** *If a DeMorgan circuit C computes a monotone Boolean function f, then the monotone Boolean function h computed by any monotone subcircuit of C satisfies $h\le f$.*

Proof. Take an arbitrary subset $Y=\{x_i\colon i\in I\}$ of variables, and let C_+ be a monotone Y-subcircuit of C. Let h be the monotone Boolean function computed by C_+. We have to show that $h\le f$ holds.

Let g be a monotone Boolean function computed by the circuit C' obtained from C by setting all variables in Y to 0. Since the function f is monotone, we have $g\le f$, and even $PI(g)\subseteq PI(f)$. Now, the circuit C_+ is obtained from C' by replacing by zeroes all remaining (not yet set to constant 1) negated input variables. So, the set $T(C_+)$ of terms produced by C_+ is obtained from $T(C')$ by removing all terms with at least one negated variable (including all zero terms). Since g is the OR of all terms in $T(C')$, and h is the OR of all terms in $T(C_+)$, the inclusion $T(C_+)\subseteq T(C')$ yields $h\le g$. So, $h\le g\le f$, as desired. ◀

Let $m\ge 3$ and $w\ge 1$ be integers. A *random (m,w)-subcircuit* $\boldsymbol{C}$ of C is a monotone $\boldsymbol{Y}$-subcircuit of C for $\boldsymbol{Y}\subseteq\{x_1,\dots,x_n\}$ being a random subset of variables with each variable included in $\boldsymbol{Y}$ independently with probability $1-\epsilon$, where

$$\epsilon := \begin{cases} \frac{1}{w} & \text{if } w\ge m,\\ 1-\frac{1}{m} & \text{if } w<m.\end{cases}$$

The next lemma is just a refinement of [17, Lemma 3].

▶ **Lemma 2.3.** *Let C be a DeMorgan circuit computing a monotone Boolean function f, and $\boldsymbol{C}$ be a random (m,w)-subcircuit of C for $m\ge 3$ and $w\ge 1$. If a prime implicant p of f has length at most m, and has negation width at most w in C, then p is produced by $\boldsymbol{C}$ with probability at least $1/K$, where $K\le 4m^w$ for $w=1,2$, and $K\le 4\cdot\min\{m^w,w^m\}$ for $w\ge 3$.*

Proof. Since the negation width of the prime implicant p in the (deterministic) circuit C is at most w, the set $T(C)$ of terms produced by C must contain a nonzero term $p\cdot r$, where term r consists solely of $l\le w$ negated variables. The probability that all these negated l

variables are set to 0 (and hence, that the term r is set to 1) is at least $(1-\epsilon)^l \geq (1-\epsilon)^w$. The probability that none of the $t \leq m$ variables of p is set to 0 is $\epsilon^t \geq \epsilon^m$. So, the prime implicant p is produced by $\boldsymbol{C}$ with probability at least $\alpha := \epsilon^m(1-\epsilon)^w$. So, it remains to show that $\alpha \geq 1/K$. When doing this, we will use two simple facts: $(1-1/t)^t \geq 1/4$ holds for all integers $t \geq 2$, and $t^s \geq s^t$ holds for all integers $3 \leq t \leq s$.

Now, if $w \geq m$, then $\epsilon = 1/w$, and we obtain $\alpha = (1/w)^m(1-1/w)^w \geq \frac{1}{4}w^{-m} \geq \frac{1}{4}m^{-w}$, where the last inequality holds because $m \geq 3$. If $w < m$, then $\epsilon = 1-1/m$, and we obtain $\alpha = (1-1/m)^m(1/m)^w \geq \frac{1}{4}m^{-w} \geq \frac{1}{4}w^{-m}$, where the last inequality holds, as long as $w \geq 3$. In both cases, we have that α is at least $\frac{1}{4} \cdot \max\{m^{-w}, w^{-m}\} \geq 1/K$, as desired. If $w = 1$ or $w = 2$, then $w < m$, and we have $\alpha \geq \frac{1}{4}m^{-w}$. ◀

3 Proof of Theorem 1.6

Theorem 1.6 is a direct consequence of the following lemma.

▶ **Lemma 3.1** (Reduction lemma). *Let f be a monotone Boolean function with all prime implicants of length at most m. If C is a DeMorgan circuit of negation width w computing f, then there exist at most $K = 8 \cdot \min\{m^w, w^m\} \cdot \log|PI(f)|$ monotone sub-circuits of C whose OR also computes f.*

In particular, if C has size s and depth d, then the resulting monotone circuit has size $s_+ \leq (s+1)K$ and depth $d_+ \leq d + \log K$. Hence, the lower bounds $s \geq s_+/K - 1$ and $d \geq d_+ - \log K$ claimed in Theorem 1.6 follow.

Proof. Let $\boldsymbol{C}$ be a random (m,w)-subcircuit of C, and take K independent copies $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_K$ of $\boldsymbol{C}$. Since the circuit C has negation width w, every prime implicant of f must have negation width at most w in C. By Lemma 2.3, we have $\Pr\{p \in T(\boldsymbol{C})\} \geq 1/t$ for every prime implicant $p \in PI(f)$ of f, where $t := 4 \cdot \min\{w^m, m^w\}$. Note that $K/t = 2 \cdot \log|PI(f)|$. Hence, for every prime implicant $p \in PI(f)$, we have

$$\Pr\{p \notin T(\boldsymbol{C}_i) \text{ for all } i = 1, \ldots, K\} \leq (1-1/t)^K \leq \mathrm{e}^{-K/t} \leq |PI(f)|^{-2}\,.$$

By the union bound, the probability that some prime implicant of f is produced by *none* of the circuits $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_K$ is strictly smaller than 1. Consequently, there must be a sequence $C_1, \ldots, C_K$ of realizations of these circuits such that *every* prime implicant of f is produced by at least one of these circuits. Consider the monotone Boolean function $h = h_1 \vee \cdots \vee h_K$, where h_i is the (monotone) Boolean function computed by C_i. By Lemma 2.2, we have $h \leq f$. On the other hand, the inclusion $PI(f) \subseteq T(C_1) \cup \cdots \cup T(C_K)$ yields the converse inequality $f \leq h$. So, the OR of the circuits $C_1, \ldots, C_K$ computes $h = f$, as desired. ◀

4 Proof of Theorem 1.8

Let f be a monotone Boolean function with all prime implicants of length at most m. Let C be a DeMorgan circuit of average negation width w computing f. Recall that a monotone Boolean function h *K-approximates* a monotone Boolean function f if there is an OR g of at least a $1/K$ portion of prime implicants of f such that $g \leq h \leq f$ holds. Now suppose that every monotone circuit K-approximating f for $K = 8 \cdot \min\{m^{2w}, (2w)^m\}$ requires t gates. Our goal is to show that then the circuit C must have at least t gates.

Since the average negation width of C is w, some set $P \subseteq PI(f)$ of $|P| \geq \frac{1}{2}|PI(f)|$ prime implicants of f have negation width at most $2w$ in C. Let $\boldsymbol{C}$ be a random (m, w)-subcircuit of C. By Lemma 2.3, we have $\Pr\{p \in T(\boldsymbol{C})\} \geq 2/K$ for every prime implicant $p \in P$. So, the expected number of prime implicants $p \in P$ produced by $\boldsymbol{C}$ is at least $2|P|/K \geq |PI(f)|/K$.

There must therefore be a realization C_+ of $\boldsymbol{C}$ such that the set $P' = P \cap T(C_+)$ has $|P'| \geq |PI(f)|/K$ terms. Let g be the OR of the terms in P', and h be the monotone Boolean function computed by C_+. Since $P' \subseteq T(C_+)$, we have $g \leq h$, while the second inequality $h \leq f$ follows from Lemma 2.2. This means that the circuit C_+ K-approximates f and, by our assumption about the function f, the monotone circuit C_+ and, hence, also the original (non-monotone) circuit C must have at least t gates, as desired. ◀

5 Proof of Theorem 1.9

It is long known that DeMorgan formulas can be balanced: every DeMorgan formula of size s can be simulated by a DeMorgan formula of depth at most $c \log s$. This was first proved by Spira [27] with $c < 3.42$, while the best currently known constant $c < 1.73$ is due to Khrapchenko [14].

In our context (when the negation width of formulas is bounded), the following natural question arises: can also DeMorgan formulas of bounded negation width be balanced *without* increasing the negation width of the resulting (balanced) formulas? The question is nontrivial because Spira's argument, as well as subsequent ones introduce negation gates applied to sub-formulas (not just to input variables), which may result in a much larger negation width.

We therefore will argue a bit differently: we first show that *monotone* formulas can be turned into balanced formulas with an additional property that all terms produced by the original formula are also produced by the balanced formula. As before, for a DeMorgan circuit or formula F, $T(F)$ denotes the set of terms produced by F. Two formulas are *equivalent* if they compute the same function.

▶ **Lemma 5.1.** *For every monotone formula F of size s, there is an equivalent monotone formula F' of depth at most $3 \log s$ such that $T(F) \subseteq T(F')$.*

Proof. Let F be a monotone formula of size s. Our goal is to show that there is an equivalent monotone formula F' of depth at most $3 \log s$ such that $T(F') \supseteq T(F)$. That is, the balanced formula F' produces all terms produced by the original formula F.

We argue by induction on s. The claim is trivially true for $s = 2$ (just take $F' = F$). Now assume that the claim holds for all formulas with fewer than s leaves, and prove it for formulas with s leaves. Take an arbitrary monotone formula F with s leaves. By walking from the output-gate of F we can find a sub-formula H such that H has $\geq s/2$ leaves but its left and right sub-formulas each have $< s/2$ leaves. Now replace the sub-formula H of F by constants 0 and 1, and let F_0 and F_1 be the resulting formulas. The key observation (already made by Brent, Kuck and Maruyama [4], and Wegener [30]) is that, due to the monotonicity, $F_1(x) = 0$ implies $F_0(x) = 0$. Thus the formula $(H \wedge F_1) \vee F_0$ is equivalent to F.

The formulas F_0 and F_1 as well as the left and right sub-formulas of H each have at most $s/2$ leaves. By the induction hypothesis, F_0 and F_1 can be replaced by formulas F_0' and F_1' of depth at most $3 \log(s/2)$, and the formula H can be replaced by a formula H' of depth at most $1 + 3 \log(s/2)$ such that

$$T(F_1) \subseteq T(F_1')\,, \quad T(F_0) \subseteq T(F_0') \quad \text{and} \quad T(H) \subseteq T(H')\,. \tag{1}$$

Thus, the resulting entire formula

$$F' = (H' \wedge F'_1) \vee F'_0 \tag{2}$$

is equivalent to F and has depth at most $2 + 1 + 3\log(s/2) = 3\log s$.

It remains to show that the set $T(F')$ of terms produced by the (balanced) formula F' satisfies $T(F') \supseteq T(F)$. Let F_z be the formula obtained from F by replacing the subformula H by a new variable z. Then the set of terms produced by F_z has the form $T(F_z) = (\{z\} * Q) \cup R$, where Q is some set of terms, R consists of all terms in $T(F_z)$ with no occurrences of the variable z, and $T_1 * T_2$ stands for the set of terms $\{t_1 \wedge t_2 \colon t_1 \in T_1, t_2 \in T_2\}$. This yields

$$T(F) = [T(H) * Q] \cup R\,, \quad T(F_1) = Q \cup R \quad \text{and} \quad T(F_0) = R\,. \tag{3}$$

So,

$$\begin{aligned} T(F') &\overset{(2)}{=} [T(H') * T(F'_1)] \cup T(F'_0) \overset{(1)}{\supseteq} [T(H) * T(F_1)] \cup T(F_0) \\ &\overset{(3)}{=} [T(H) * (Q \cup R)] \cup R \supseteq [T(H) * Q] \cup R \overset{(3)}{=} T(F)\,. \end{aligned}$$

◀

Proof of Theorem 1.9. Let f be a monotone Boolean function, and $w \geq 0$. Suppose that f can be computed by a DeMorgan formula $G = G(x, \overline{x})$ of size s and negation width w. Our goal is to show that then f can be computed by a DeMorgan formula of negation width at most w and depth at most $3 \cdot \log s$.

Replace all negated input variables $\overline{x}_i$ in G by new variables y_i, and consider the monotone formula $F = G(x, y)$. Since the formula G has negation width w, we know that the monotone formula F has the following property:

(a) for every prime implicant $p = \bigwedge_{i \in S} x_i$ of f there is a term $p \cdot r \in T(F)$ with $r = \bigwedge_{j \in T} y_j$, $T \cap S = \emptyset$ and $|T| \leq w$.

Apply Lemma 5.1 to the formula $F(x, y)$. This gives us a monotone formula $F'(x, y)$ of depth at most $3\log s$ whose set $T(F')$ of produced terms contains all terms produced by the formula F. This latter property implies that the (balanced) formula F' also has property (a). So, if we replace back in $F'(x, y)$ the input variables y_i by negated variables $\overline{x}_i$, the obtained (also balanced) DeMorgan formula $F''(x, \overline{x})$ computes our function f and has negation width w, as desired. ◀

6 Explicit lower bounds

For a monotone Boolean function $f(x_1, \ldots, x_n)$, $\mathrm{C}_w(f)$ will denote the minimum size of a DeMorgan circuit of negation width w computing f, while $\mathrm{C}_+(f)$ will denote the minimum size of a monotone circuit computing f. In the case of DeMorgan *formulas*, these measures are denoted by $\mathrm{L}_w(f)$ and $\mathrm{L}_+(f)$; in this case, the *size* of a formula is the number of leaves of the underlying tree. Let also $\mathrm{D}_w(f)$ denote the minimum *depth* of a DeMorgan circuit of negation width w computing f, and let $\mathrm{D}_+(f)$ denote the minimum depth of a monotone circuit computing f.

Theorem 1.6 directly yields the following lower bounds on the size and depth of DeMorgan circuits of bounded negation width. Let $f(x_1, \ldots, x_n)$ be a monotone Boolean function with M prime implicants, each of length at most m. Then for any $w \geq 0$, we have

$$\mathrm{C}_w(f) \geq \frac{\mathrm{C}_+(f)}{K} - 1\,, \quad \mathrm{L}_w(f) \geq \frac{\mathrm{L}_+(f)}{K} - 1 \quad \text{and} \quad \mathrm{D}_w(f) \geq \mathrm{D}_+(f) - \log K\,, \tag{4}$$

where

$$K = 8 \cdot \min\{m^w, w^m\} \cdot \log M\,. \tag{5}$$

The *k-clique function* CLIQUE(n,k) has $\binom{n}{2}$ variables, one for each edge of the complete graph K_n on $[n] = \{1,\ldots,n\}$. Every assignment of Boolean values to these variables specifies a subgraph of K_n, and the function accepts the assignment if and only if the specified graph contains a complete graph on k or more vertices; note that we do not require k to be an integer.

▶ **Corollary 6.1** (Small cliques). *There are absolute constants $c_1, c_2 > 0$ such that, if $f =$ CLIQUE$(n,3)$, and $w \le n^\epsilon$ for $\epsilon > 0$, then*

$$c_1 n^{3-4\epsilon} \le \mathrm{C}_w(f) \le c_2 n^{3-\epsilon/4}\,.$$

Proof. Here we only show the lower bound; the proof of the upper bound $\mathrm{C}_w(f) = O(n^{3-\epsilon/4})$ is given in Appendix A.2 (see Claim A.4). As shown by Alon and Boppana [1, Lemma 3.14], $\mathrm{C}_+(f) = \Omega\left(n^3/\log^3 n\right)$ holds. Since f has $M = \binom{n}{3} \le n^3$ prime implicants, each of length $m = 3$, the parameter K in Equation (5) is at most a constant times $w^m \cdot \log M \le 3n^{3\epsilon}\log n$, and Equation (4) yields the desired lower bound $\mathrm{C}_w(f) \ge \mathrm{C}_+(f)/K - 1 = \Omega(n^{3-4\epsilon})$. ◀

Amano and Maruoka [2, Theorem 4.2] proved that, for any $3 \le k \le n^{2/3}$, DeMorgan circuits of negation width $w = o(\sqrt{k})$ computing $f =$ CLIQUE(n,k) require $2^{\Omega(\sqrt{k})}$ gates. (In their definition of the negation width [2, Definition 4.1], they use different terminology, but it is not difficult to see that their measure coincides with that given in our Definition 1.2.) Note, however, that here the allowed negation width $w = o(\sqrt{k})$ is much smaller than the clique size k. When combined with the lower bound of Alon and Boppana [1] for cliques of moderate (logarithmic) size, Equation (4) directly yields super-polynomial lower bounds also when the allowed negation width is much larger, even *exponential*, in the cliques size.

▶ **Corollary 6.2** (Moderate cliques). *Let $f =$ CLIQUE(n,k) with $k = \log^{1/3} n$. Then $\mathrm{C}_w(f) = n^{\Omega(k)}$ holds for $w = 2^k$.*

Proof. It is shown in [1, Theorem 3.16] that $\mathrm{C}_+(f) \ge n^k/(8k^2\mathrm{e}^k \log n)^k$ holds for any $3 \le k \le \frac{1}{4}\log n$. In particular, for $k = \log^{1/3} n$, we have $\mathrm{C}_+(f) = n^{\Omega(k)}$. On the other hand, since f has $|PI(f)| = \binom{n}{k} \le n^k$ prime implicants, each of length $m = \binom{k}{2} \le k^2$, the parameter K in Equation (5) is at most a constant times $w^m \cdot \log M \le 2^{k^3} k \log n \le n \log^2 n$, and Equation (4) yields $\mathrm{C}_w(f) \ge \mathrm{C}_+(f)/K - 1 = n^{\Omega(k)}$. ◀

▶ **Corollary 6.3** (Large cliques). *Let $f =$ CLIQUE$(n, n/2)$. If $w \le \epsilon n/\log n$ for a sufficiently small constant $\epsilon > 0$, then $\mathrm{D}_w(f) = \Omega(n)$.*

Proof. Raz and Wigderson [22, Corollary 4.1] have proved that $\mathrm{D}_+(f) = \Omega(n)$. Since f has $M = \binom{n}{n/2} \le 2^n$ prime implicants, each of length $m = \binom{n/2}{2} \le n^2$, the logarithm of the parameter K in Equation (5) is at most a constant times $w \log m + \log\log M = O(w \log n)$. Equation (4) yields $\mathrm{D}_w(f) \ge \mathrm{D}_+(f) - \log K = \mathrm{D}_+(f) - O(w \log n) = \Omega(n)$, as desired. ◀

▶ **Corollary 6.4.** *If $f =$ CLIQUE$(n, n/2)$, then $\mathrm{L}_w(f) = 2^{\Omega(n)}$ holds for DeMorgan formulas of negation width $w = o(n/\log n)$.*

Proof. The desired lower bound follows directly from Corollary 6.3 and our refinement of Spira's depth-reduction given in Theorem 1.9. ◀

▶ **Corollary 6.5** (Tardos' function). *There is a monotone Boolean function T_n of n variables such that T_n can be computed by a DeMorgan circuit of polynomial in n size, but $\mathrm{C}_w(T_n) = 2^{\Omega(n^{1/7})}$ holds when the allowed negation width is $w \le n^{1/7}$.*

Proof. Tardos [29] observed that an efficient algorithm for computing the Lovász theta function, designed by Grötschel, Lovász and Schrijver [8], gives us a monotone Boolean function T_n of $n = \binom{v}{2}$ variables which is computable by DeMorgan circuits of polynomial in n size, and shares common properties with Clique functions sufficient for Alon and Boppana [1] to yield a lower bound $\mathrm{C}_+(T_n) = 2^{\Omega(v/\log v)^{1/3}} = 2^{\Omega(n/\log n)^{1/6}}$. On the other hand, the parameter K in Equation (5) is exponential in at most a constant times $w \log n \le n^{1/7} \log n \ll (n/\log n)^{1/6}$. So, Equation (4) immediately yields the claimed lower bound $\mathrm{C}_w(T_n) \ge \mathrm{C}_+(T_n)/K - 1 = 2^{\Omega(n^{1/7})}$ for circuits of negation width $w = n^{1/7}$. ◀

▶ Remark 6.6. Note that the total number N of variables in each clique function $\mathrm{CLIQUE}(n,k)$ is $N = \binom{n}{2} = \Theta(n^2)$. The highest known lower bound on the monotone circuit complexity of an explicit Boolean function of N variables was proved by Harnik and Raz [10], and is exponential in $(N/\log N)^{1/3}$ Recently, Pitassi and Robere [19] gave an explicit monotone Boolean function f of N variables such that $\mathrm{D}_+(f) = \Omega(N)$. The lower bound in Equation (4) implies that any (non-monotone) DeMorgan circuit of negation width $w = \epsilon N$ for a sufficiently small constant $\epsilon > 0$ must have linear depth $\Omega(N)$. Together with Theorem 1.9, this result implies a truly exponential lower bound $\mathrm{L}_w(f) = 2^{\Omega(N)}$ on the size of DeMorgan *formulas* of negation width $w = \epsilon N$. Note that the ultimate goal is to prove lower bounds for DeMorgan circuits of negation width $w = N$ (or only $w = N - m$, where m is the minimum length of a prime implicant): these bounds then would hold for *unrestricted* circuits.

Finally, let us give an application of our Theorem 1.8 concerning DeMorgan circuits of bounded *average* negation width. As we already mentioned in Section 1.3, in order to apply this theorem, we need lower bounds on the size of monotone circuits that only *approximate* a given monotone Boolean function (see Definition 1.7).

Fortunately, known lower bound arguments for monotone circuits (see, for example, [13, Chapter 9] and the literature cited herein) work also when the monotone circuits are only required to produce a large enough subset of prime implicants (not necessarily *all* prime implicants). Just to give an example, let us show the following simple consequence of [12, Theorem 3.4].

▶ **Lemma 6.7.** *Let $3 \le k \le \sqrt{n}$, and let f be a monotone Boolean function which rejects all graphs of chromatic number at most $k-1$, and accepts a $1/K$-fraction of all k-cliques. Then $\mathrm{C}_+(f) \ge 2^{\Omega(\sqrt{k})}/K$.*

Proof. Every q-coloring $h : [n] \to [q]$ of the vertices of K_n defines the graph G_h whose edges are pairs of vertices receiving the *same* color. Note that the chromatic number of the complement of every G_h does not exceed q; so, for $q := k-1$, the complements of graphs G_h must be rejected by f. An *s-forest* is a forest with s edges.

As shown in [12, Theorem 3.4], if f can be computed by a monotone circuit of size t, then for any integer parameters $1 \le r, s \le n-1$ there exist a family of $t \cdot (2s)^{2r}$ r-cliques, a family of $t \cdot (2r)^{2s}$ s-forests, and a set E of r^2 edges such that at least one of the following two assertions holds:

(1) every k-clique accepted by f contains at least one of the given r-cliques;

(2) for every q-coloring h, the graph G_h either intersects E or contains at least one of the given s-forests.

Every r-clique is contained in exactly $\binom{n-r}{k-r}$ k-cliques. So, under the first alternative (1), the size t of the circuit must be at least $\binom{n}{k}/K$ divided by $(2s)^{2r}\binom{n-r}{k-r}$, which is at least $(n/4ks^2)^r/K$. On the other hand, out of all q^n possible q-colorings h of the vertices of K_n, at most q^{n-l} of the graphs G_h can contain a *fixed* forest with l edges. This is directly shown in the proof of Theorem 3.4 in [12], but also follows from the fact that random q-coloring colors two vertices by the same color with probability $1/q$, and these events are independent for edges in a *forest*. So, under the second alternative (2), the size t of the circuit must be at least $q^n - r^2 \cdot q^{n-1} = q^n(1 - r^2/q)$ divided by $(2r)^{2s}q^{n-s}$ which, for any $r \leq \sqrt{q/2}$, is at least $\frac{1}{2}(k/4r^2)^s$.

By taking the parameters $r := \lfloor\sqrt{k/16}\rfloor$ and $s := \lfloor\sqrt{n/8k}\rfloor$, the first alternative yields a lower bound $t \geq 2^r/K$, while the second one yields $t \geq \frac{1}{2}4^s \geq 2^s$. Since our assumption $k \leq \sqrt{n}$ yields $s \geq r$, the desired lower bound $t \geq 2^r/K \geq 2^{\Omega(\sqrt{k})}/K$ follows. ◀

▶ **Corollary 6.8.** *Let $f = \mathrm{CLIQUE}(n,k)$ for $k \leq \sqrt{n}$. Then every DeMorgan circuit of average negation width $w = o(\sqrt{k}/\log k)$ computing f must have $2^{\Omega(\sqrt{k})}$ gates.*

Proof. Lemma 6.7 implies that, for every $K \geq 1$, every monotone circuit K-approximating f requires at least $t = 2^{\Omega(\sqrt{k})}/K$ gates. The length of prime implicants of f is $m = \binom{k}{2}$. So, by taking $K := 8m^{2w} = 2^{o(\sqrt{k})}$, Theorem 1.8 yields the desired lower bound on the size t of any DeMorgan circuit of average negation width w computing f. ◀

The aforementioned result [12, Theorem 3.4] holds also for monotone circuits with *unbounded* fanin AND and OR gates. The reduction lemma (Lemma 3.1) also holds for DeMorgan circuits with unbounded fanin AND and OR gates. So, Corollary 6.8 holds for DeMorgan circuits with unbounded fanin gates.

7 Final remarks

The measure of the *orientation weight* of a circuit over $\{\vee, \wedge, \neg\}$, considered by Koroth and Sarma [16], is the minimum number w such that, for every gate u, the function f_u computed at u is monotone in at least $n - w$ variables. In circuits of nonzero orientation w, negations are allowed to be applied to inner gates (not only to input variables). On the other hand, the (functional) use of such NOT gates is severely restricted: the function computed at each NOT gate in such a circuit cannot depend on more than $2w$ variables.

To see this, let $g = \neg h$ be the function computed at some NOT gate, and h the function computed at its input. Let X be the set of variables on which g depends. We know that neither g nor h can be non-monotone in more than w variables. If g is monotone in a variable $x_i \in X$, then h is non-monotone in x_i. So, g cannot be monotone in more than w variables of X. Since, due to the orientation width restriction, the function g itself cannot be non-monotone in more than w variables, the desired upper bound $|X| \leq 2w$ follows.

Our relaxation (the negation width, see Definition 1.2) is of a more "syntactic" nature than that of the orientation weight in Koroth and Sarma [16], but is also of a similar spirit. Instead of requiring that the produced extensions of prime implicants can only use negated variables from one *fixed* subset of $\leq w$ negated variables (as in [21, 9]), we now allow the extensions to use *different* subsets of $\leq w$ negated variables for different prime implicants. But, in contrast to [16], we have no restrictions on functions computed at *intermediate* gates: only terms produced at the end do matter. And only *nonzero* terms do matter: produced zero terms do not contribute to the negation width at all. The question of how (if at all) the orientation width of DeMorgan circuits is related to their negation width remains open.

Finally, let us note that our reduction to monotone circuits works also for *switching-and-rectifier networks*. Recall that such a network (known also as a *nondeterministic branching program*) is a directed acyclic graph with one distinguished source node s of zero indegree, and one distinguished target node t of zero outdegree. Every edge is either labeled by a literal or is unlabeled. Every s-t path defines a (not necessarily nonzero) term: just take the AND of all labels of edges along this path. So, the set of terms produced by the network is now just the set of terms defined by the s-t paths. The function f computed by the network is the OR of terms defined by all s-t paths. A network is monotone if it has no negated variables as labels. In a *switching network* (or *contact scheme*) the underlying graph is undirected.

The negation width of the network can be analogously defined as the minimal number w such that for every prime implicant p of f there is an s-t path defining an extension of p by at most w negated variables. This means that for every input $a \in f^{-1}(1)$ with a minimal number of 1s, there must be an s-t path along which all 1-positions and at most w 0-positions of a are tested. It is easy to see that the reduction lemma (Lemma 3.1) can be immediately adapted (by just replacing the term "sub-circuit" by "sub-network") to hold also for switching networks as well as for switching-and-rectifier networks with the same blow-up parameter K.

Potechin [20] has proved an interesting tradeoff between monotone switching networks and monotone switching-and-rectifier networks computing the s-t connectivity function STCON(n) on *directed* n-vertex graphs: every monotone switching network computing this function must have at least $n^{\Omega(\log n)}$ nodes. On the other hand, although this was not mentioned in [20], the well-known dynamic programming algorithm of Bellman and Ford gives a monotone switching-and-rectifier network for STCON(n) with only $O(n^2)$ nodes and $O(n^3)$ edges. Lemma 3.1 (adapted to switching networks) extends Potechin's lower bound to *non-monotone* switching networks of negation width $w = o(\log n)$: the blow-up parameter K is in this case at most $n^{o(\log n)}$.

References

1. N. Alon and R. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.
2. K. Amano and A. Maruoka. The Potential of the Approximation Method. *SIAM J. Comput.*, 33(2):433–447, 2004.
3. E. Blais, C.L. Canonne, I.C. Oliveira, R.A. Servedio, and L.Y. Tan. Learning Circuits with few Negations. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 40 of *LIPIcs*, pages 512–527, 2015.
4. R.P. Brent, D.J. Kuck, and K. Maruyama. The parallel evaluation of arithmetic expressions without divisions,. *IEEE Trans. Computers*, C-22:523–534, 1973.
5. Y. Crama and P.L. Hammer, editors. *Boolean Functions: Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Pess, 2011.
6. P.E. Dunne. Relationship between monotone and non-monotone network complexity. In M.S. Paterson, editor, *Boolean Function Complexity*, volume 169 of *London Math. Soc. Lect. Note Series*, pages 1–24. Cambridge University Press, 1992.
7. F. Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *Proc. of 39th Int. Symp. on Symbolic and Algebraic Computation*, pages 296–303, 2014.
8. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

9 S. Guo, T. Malkin, I.C. Oliveira, and A. Rosen. The Power of Negations in Cryptography. In *Proc. of 12th Theory of Cryptography Conference, TCC*, volume 9014 of *Lect. Notes in Comput. Sci.*, pages 36–65. Springer, 2015.

10 D. Harnik and R. Raz. Higher lower bounds on monotone size. In *Proc. 32nd Ann. ACM Symp. on Theory of Computing*, pages 378–387, 2000.

11 J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

12 S. Jukna. Combinatorics of monotone computations. *Combinatorica*, 19(1):65–85, 1999. Preliminary versions in: ECCC Report Nr. 26, 1996, and in Proc. of 12th Ann. IEEE Conf. on Comput. Complexity. 1997, pp. 223-238.

13 S. Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer-Verlag, 2012.

14 V.M. Khrapchenko. On a relation between the complexity and the depth of formulas. In *Methods of Discrete Analysis in Synthesis of Control Systems*, volume 32, pages 76–94. Institute of Mathematics. Novosibirsk, 1978. (In Russian).

15 M. Kochol. Efficient monotone circuits for threshold functions. *Inf. Process. Lett.*, 32:121–122, 1989.

16 S. Koroth and J. Sarma. Depth Lower Bounds against Circuits with Sparse Orientation. *Fundam. Inform.*, 152(2):123–144, 2017.

17 A. Lingas. Small Normalized Boolean Circuits for Semi-disjoint Bilinear Forms Require Logarithmic Conjunction-depth. In *Proc. of 33rd Comput. Complexity Conf.*, volume 102 of *LIPIcs*, pages 26:1–26:10, 2018. Extended version in: ECCC Report Nr. 108, 2018.

18 E.A. Okol'nishnikova. On the influence of one type of restrictions to the complexity of combinational circuits. *Diskrete Analysis*, 36:46–58, 1981. (In Russian).

19 T. Pitassi and R. Robere. Strongly exponential lower bounds for monotone computation. In *Proc. 49th Ann. ACM Symp. on Theory of Computing, STOC*, pages 1246–1255, 2017.

20 A. Potechin. Bounds on Monotone Switching Networks for Directed Connectivity. *J. ACM*, 64(4):29:1–29:48, 2017.

21 R. Raz and Wigderson. Probabilistic Communication Complexity of Boolean Relations. In *Proc. of 30th Ann. Symp. on Foundations of Computer Sci., FOCS*, pages 562–567, 1989.

22 R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. *JACM*, 39(3):736–744, 1992.

23 A.A. Razborov. Lower bounds for the monotone complexity of some boolean functions. *Soviet Math. Dokl.*, 31:354–357, 1985.

24 A.A. Razborov. Lower bounds on monotone complexity of the logical permanent. *Math. Notes of the Acad. of Sci. of the USSR*, 37(6):485–493, 1985.

25 A.A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990.

26 B. Rossman. Correlation Bounds Against Monotone NCˆ1. In *Proc. of 30th Comput. Complexity Conf.*, volume 33 of *LIPIcs*, pages 392—-411, 2015.

27 P.M. Spira. On time–hardware complexity tradeoffs for Boolean functions. In *Proc. of 4th Hawaii Symp. on System Sciences*, pages 525–527. Western Periodicals Company, North Hollywood, 1971.

28 V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

29 É. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 7(4):141–142, 1987.

30 I. Wegener. Relating monotone formula size and monotone depth of Boolean functions. *Infrom. Process. Letters*, 16:41–42, 1983.

31 I. Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.

32 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. of 44th Symp. on Theory of Comput., STOC*, pages 887–898, 2012.

A Motivating examples

We want explicit examples of monotone Boolean functions $f(X)$ of $|X| = n$ variables such that f requires large monotone circuits, but has small circuits when a moderate negation width $w = n^\epsilon$ for an arbitrarily small constant $\epsilon > 0$ is allowed. Such functions can be constructed using the following two simple observations: (1) negation width is always at most the total number of input variables, and (2) OR gates cannot increase the negation width.

A.1 Super-polynomial gaps

▶ **Example A.1** (Logical permanent)**.** The *logical permanent* function Per_m is a monotone Boolean function of m^2 variables which takes a Boolean $m \times m$ matrix Y as input, and outputs 1 if and only if Y contains m 1-entries no two of which lie in the same row or the same column. Let $0 < \epsilon < 1/2$ be an arbitrarily small constant, and assume for simplicity that both $m = n^\epsilon$ and $r = n^{1-\epsilon}$ are integers. Consider the monotone Boolean function $f(X)$ whose variables are arranged into an $n \times n$ matrix X. Split X into r^2 disjoint $m \times m$ submatrices. The function f accepts X if and only if $\mathrm{Per}_m(Y) = 1$ holds for at least one of these submatrices Y. The monotone circuit complexity of f is at least the monotone circuit complexity of Per_m which, as shown by Razborov [24], is $m^{\Omega(\log m)} = n^{\Omega(\log n)}$.

On the other hand, it is well known that Per_m can be computed by a DeMorgan circuit of size polynomial in m; see, for example, Hopcroft and Karp [11]. The negation width of such a circuit is clearly at most the number m^2 of its input variables. So, since at OR gates the negation width is not increased, we obtain a DeMorgan circuit for f of size $r^2 \cdot m^{O(1)} = n^{O(1)}$ and negation width $w \le m^2 = n^{2\epsilon}$.

▶ **Example A.2** (Tardos' function)**.** Let $0 < \epsilon < 1$ be an arbitrarily small constant, and assume for simplicity that both $m = n^\epsilon$ and $r = n^{1-\epsilon}$ are integers. As we already mentioned in the proof of Corollary 6.5, Tardos [29] exhibited a monotone Boolean function T_m of $m = \binom{v}{2}$ variables which can be computed by a DeMorgan circuit of polynomial in m size, but the monotone circuit complexity of T_m is exponential in $(v/\log v)^{1/3} = m^{\Omega(1)}$. Let f_n be a monotone Boolean function of $n = r \cdot m$ variables defined as the OR of r copies of T_m on disjoint m-element sets of variables. Then the monotone circuit complexity of f_n is also exponential in $m^{\Omega(1)} = n^{\Omega(1)}$, but the function f_n can be computed by a DeMorgan circuit of size $r \cdot n^{O(1)} = n^{O(1)}$ if the negation width $w = m$ $(= n^\epsilon)$ is allowed.

These two examples show that the size of monotone circuits (DeMorgan circuits of negation width $w = 0$) can be substantially (even super-polynomially) reduced by allowing moderate negation width $w = n^\epsilon$. Our next two examples (of the triangle function and threshold functions) show that non-trivial savings are also possible for monotone Boolean functions that *have* small (polynomial) monotone circuits.

A.2 The triangle function

Our goal is to show that for every constant $\epsilon > 0$, the triangle function $\mathrm{CLIQUE}(n, 3)$ can be computed by a DeMorgan circuit of negation width $w = n^\epsilon$ using a sub-cubic number $O(n^{3-\epsilon/4})$ of gates. The multi-output "cousin" of the triangle function is the *Boolean matrix multiplication* operator $\mathrm{BMM}(n) : \{0,1\}^{2n^2} \to \{0,1\}^{n^2}$. This operator takes two $n \times n$ Boolean matrices $X = (x_{i,j})$ and $Y = (y_{i,j})$ as inputs, and computes n^2 monotone Boolean functions $f_{i,j} = \bigvee_{k=1}^{n} x_{i,k} y_{k,j}$. Note that now, instead of just one output gate, every circuit computing $\mathrm{BMM}(n)$ has n^2 output gates. The negation width of such a (multi-output) circuit is just the maximum negation width of its sub-circuits computing the functions $f_{i,j}$.

Fast algebraic algorithms for arithmetic matrix multiplication [28, 7, 32] yield circuits over $\{\vee, \wedge, \neg\}$ for the $n \times n$ Boolean matrix product with $O(n^\omega)$ gates, where ω is the so-called matrix multiplication exponent; after the Strassen [28] breakthrough algorithm showed that $\omega < 2.807$, this exponent was pushed down by Vassilevska Williams [32] and Le Gall [7] to $\omega < 2.373$. This can be used to show that the circuit complexity of BMM(n) remains *sub-cubic* also when the negation width of circuits is lowered from the trivial $w = 2n$ (unrestricted circuits) to $w = n^\epsilon$ for an arbitrarily small constant $\epsilon > 0$.

▷ Claim A.3. For every $0 < \epsilon \leq 1$, the operator BMM(n) can be computed by a DeMorgan circuit of negation width $w = n^{2\epsilon}$ and size $O(n^{3-\epsilon/2})$.

Proof. We will use essentially the same argument as was used in [17, Proposition 12] to show an upper bound $O(n^{3-c})$ for an unspecified constant $c = c_\epsilon > 0$.

Set $m := \frac{1}{2}n^\epsilon$, and assume (for the sake of simplicity) that both m and $r := n/m$ are integers. Partition each of the given $n \times n$ matrices X and Y into r^2 disjoint $m \times m$ submatrices. The product of each pair of such submatrices can be computed by a DeMorgan circuit of size $O(m^\omega)$; the negation width of each of these circuits is trivially at most $2m^2 = n^{2\epsilon}$. So, it is enough to compute r^3 products of $m \times m$ submatrices, and to use additional rn^2 OR gates to compute all n^2 entries of the product matrix $X \cdot Y$. Since the negation width can only increase at AND gates, the negation width of the resulting circuit remains the same, that is, remains at most $w = n^{2\epsilon}$. Since $r = n/m$ with $m = \frac{1}{2}n^\epsilon$, and since $3 - \omega \geq 1/2$, the size of the resulting circuit is at most a constant times $r^3 m^\omega + rn^2 = n^3/m^{3-\omega} + n^3/m \leq 2n^3/\sqrt{m} \leq 3n^{3-\epsilon/2}$, as desired. ◁

▷ Claim A.4. For every $0 < \epsilon \leq 1$, the triangle function $f = \text{CLIQUE}(n, 3)$ can be computed by a DeMorgan circuit of negation width $w = n^{2\epsilon}$ and size $O(n^{3-\epsilon/2})$.

Proof. By Claim A.3, all n^2 entries $Y = (y_{i,j})$ of the Boolean matrix product $Y = X \cdot X$ of the adjacency $n \times n$ matrix of a given graph G can be simultaneously computed by a DeMorgan circuit of negation width $w = n^{2\epsilon}$ and size $O(n^{3-\epsilon/2})$. So, the function $f = \bigvee_{i<j} y_{i,j} \wedge x_{i,j}$ can be computed by taking a componentwise AND of Y and X, and computing the OR of all entries of the resulting matrix. ◁

A.3 Threshold functions

Recall that the threshold-k function Th_k^n accepts a Boolean input of length n if and only if it contains at least k ones. The smallest known monotone circuits for Th_k^n have size of order $n \log k$ (see, for example, [15]). On the other hand, we will now show that Th_k^n can be computed by a DeMorgan circuit of *linear* size $O(n)$ if negation width $w = k^3$ is allowed.

▷ Claim A.5. If $w = k^3$, then $\text{C}_w(\text{Th}_k^n) = O(n)$.

Proof. For the sake of simplicity of argumentation, assume that the number of variables n is divisible by the parameter $s \geq k$ (to be chosen latter). Divide the sequence X of $|X| = n$ Boolean variables into $m := n/s$ consecutive segments $X_1, \ldots, X_m$ of length s, and let $Q_l^j = \text{Th}_l^s(X_j)$ be the threshold-l function on the s variables in the jth segment.

It is well known (see, for example, [31, Sect. 3.4]) that all functions $\text{Th}_1^n, \text{Th}_2^n, \ldots, \text{Th}_n^n$ can be simultaneously computed by a (non-monotone) DeMorgan circuit of size $O(n)$. So, for every $j = 1, \ldots, m$, all the functions $Q_0^j, Q_1^j, \ldots, Q_k^j$ can be simultaneously computed by a DeMorgan circuit of size $O(s)$. It follows that all functions Q_l^j for $j = 1, \ldots, m$ and $l = 1, \ldots, k$ can be simultaneously computed by a DeMorgan circuit of size at most a constant

times $s \cdot (n/s) = n$. We now use a simple dynamic program to compute all the Boolean functions P_l^j such that $P_l^j = 1$ if and only if there are at least l ones in the first j segments.

As basis functions we take $P_0^j = Q_0^j = 1$ (constant 1 functions) for all $j = 1, \ldots, m$, $P_l^1 = Q_l^1$ for all $l = 1, \ldots, k$, and construct a DeMorgan circuit C using the recurrences

$$P_l^j = \bigvee_{r=0}^{l} P_{l-r}^{j-1} \wedge Q_r^j \,. \tag{6}$$

It is easy to see that the whole input sequence contains at least k ones iff $P_k^m = 1$. For the jth segment, we account $O(k^2)$ additional gates implementing the recurrences for P_l^j. Hence, the size of the DeMorgan circuit C computing P_k^m is at most a constant times $mk^2 = (n/s)k^2$.

To upper-bound the negation width of the resulting circuit, just expand the recursion (6). We then see that P_k^m is computed as the OR of ANDs $Q_{r_1}^1(X_1) \wedge Q_{r_2}^2(X_2) \wedge \cdots \wedge Q_{r_m}^m(X_m)$ over all sequences $r_1, \ldots, r_m$ of nonnegative integers satisfying $r_1 + \cdots + r_m = k$; recall that $Q_0^j = 1$ for all j. Since at most k of the r_js in each such sequence can be nonzero, at most k of the functions $Q_{r_j}^j$ can be not constant 1 functions. So, every term produced by the circuit C is of the form $q = \bigwedge_{j \in J} q_j$ for some subset $J \subseteq [m]$ of size $|J| \leq k$, where each q_j is a (not necessarily nonzero) term containing variables or their negations only from the jth segment X_j. So, if q is a nonzero term, then it can have at most $\sum_{j \in J} |X_j| \leq ks$ distinct literals and, hence also at most ks distinct negated variables. In particular, this means that all nonzero terms produced by the circuit C including the extensions of prime implicants of the computed by C function P_k^m, have at most ks distinct negated variables.

So, the constructed circuit C for the threshold-k function Th_k^n has negation width $w \leq ks$ and size of order $(n/s)k^2$. It remains to take the segment-length $s = k^2$. This gives us a circuit of linear size $O(n)$ and negation width at most k^3, as desired. ◁

Depth First Search in the Semi-streaming Model

Shahbaz Khan
Faculty of Computer Science, University of Vienna, Austria
shahbaz.khan@univie.ac.at

Shashank K. Mehta
Dept. of Computer Science and Engineering, Indian Institute of Technology Kanpur, India
skmehta@cse.iitk.ac.in

Abstract

Depth first search (DFS) tree is a fundamental data structure for solving various graph problems. The classical algorithm for building a DFS tree requires $O(m+n)$ time for a given undirected graph G having n vertices and m edges. In the streaming model, an algorithm is allowed several passes (preferably single) over the input graph having a restriction on the size of local space used.

Now, a DFS tree of a graph can be trivially computed using a single pass if $O(m)$ space is allowed. In the semi-streaming model allowing $O(n)$ space, it can be computed in $O(n)$ passes over the input stream, where each pass adds one vertex to the DFS tree. However, it remains an open problem to compute a DFS tree using $o(n)$ passes using $o(m)$ space even in any relaxed streaming environment.

We present the first semi-streaming algorithms that compute a DFS tree of an undirected graph in $o(n)$ passes using $o(m)$ space. We first describe an extremely simple algorithm that requires at most $\lceil n/k \rceil$ passes to compute a DFS tree using $O(nk)$ space, where k is any positive integer. For example using $k = \sqrt{n}$, we can compute a DFS tree in $\sqrt{n}$ passes using $O(n\sqrt{n})$ space. We then improve this algorithm by using more involved techniques to reduce the number of passes to $\lceil h/k \rceil$ under similar space constraints, where h is the height of the computed DFS tree. In particular, this algorithm improves the bounds for the case where the computed DFS tree is *shallow* (having $o(n)$ height). Moreover, this algorithm is presented in form of a *framework* that allows the *flexibility* of using any algorithm to maintain a DFS tree of a stored sparser subgraph as a *black box*, which may be of an independent interest. Both these algorithms essentially demonstrate the existence of a trade-off between the space and number of passes required for computing a DFS tree. Furthermore, we evaluate these algorithms experimentally which reveals their exceptional performance in practice. For both random and real graphs, they require merely a *few* passes even when allowed just $O(n)$ space.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Data structures design and analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Depth First Search, DFS, Semi-Streaming, Streaming, Algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.42

Related Version Full version of the papers is available at `https://arxiv.org/abs/1901.03689`.

Funding *Shahbaz Khan*: This research work was supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

1 Introduction

Depth first search (DFS) is a well known graph traversal technique. Right from the seminal work of Tarjan [35], DFS traversal has played an important role in the design of efficient algorithms for many fundamental graph problems, namely, bi-connected components, strongly connected components, topological sorting [38], dominators in directed graph [36], etc. Even

in undirected graphs, DFS traversal have various applications including computing connected components, cycle detection, edge and vertex connectivity [13] (via articulation points and bridges), bipartite matching [23], planarity testing [24] etc. In this paper, we address the problem of computing a DFS tree in the semi-streaming environment.

The streaming model [2, 17, 19] is a popular model for computation on large data sets wherein a lot of algorithms have been developed [18, 22, 19, 25] to address significant problems in this model. The model requires the entire input data to be accessed as a stream, typically in a single pass over the input, allowing very small amount of storage ($poly \log$ in input size). A streaming algorithm must judiciously choose the data to be saved in the small space, so that the computation can be completed successfully. In the context of graph problems, this model is adopted in the following fashion. For a given graph $G = (V, E)$ having n vertices, an input stream sends the graph edges in E using an arbitrary order only once, and the allowed size of local storage is $O(poly \log n)$. The algorithm iteratively asks for the next edge and performs some computation. After the stream is over, the final computation is performed and the result is reported. At no time during the entire process should the total size of stored data exceed $O(poly \log n)$.

In general only statistical properties of the graph are computable under this model, making it impractical for use in more complicated graph problems [15, 20]. A prominent exception for the above claim is the problem of counting triangles (3-cycles) in a graph [5]. Consequently, several relaxed models have been proposed with a goal to solve more complex graph problems. One such model is called *semi-streaming model* [32, 16] which relaxes the storage size to $O(n\ poly \log n)$. Several significant problems have been studied under this model (surveys in [33, 41, 31]). Moreover, even though it is preferred to allow only a single pass over the input stream, several hardness results [22, 10, 16, 9, 21] have reported the limitations of using a single pass (or even $O(1)$ passes). This has led to the development of various multi-pass algorithms [16, 15, 30, 1, 27, 26] in this model. Further, several streaming algorithms maintaining approximate distances [15, 6, 11] are also known to require $O(n^{1+\epsilon})$ space (for some constant $\epsilon > 0$) relaxing the requirement of $O(n\ poly \log n)$ space.

Now, a DFS tree of a graph can be computed in a single pass if $O(m)$ space is allowed. If the space is restricted to $O(n)$, it can be trivially computed using $O(n)$ passes over the input stream, where each pass adds one vertex to the tree. This can also be easily improved to $O(h)$ passes, where h is the height of the computed DFS tree. Despite most applications of DFS trees in undirected graphs being efficiently solved in the semi-streaming environment [40, 16, 15, 3, 4, 14, 29], due to its fundamental nature DFS is considered a long standing open problem [14, 33, 34] even for undirected graphs. Moreover, computing a DFS tree in $O(poly \log n)$ passes is considered hard [14]. To the best of our knowledge, it remains an open problem to compute a DFS tree using $o(n)$ passes even in any relaxed streaming environment.

In our results, we borrow some key ideas from recent sequential algorithms [8, 7] for maintaining dynamic DFS of undirected graphs. Recently, similar ideas were also used by Khan [28] who presented a semi-streaming algorithm that uses using $O(n)$ space for maintaining dynamic DFS of an undirected graph, requiring $O(\log^2 n)$ passes per update.

1.1 Our Results

We present the first semi-streaming algorithms to compute a DFS tree on an undirected graph in $o(n)$ passes. Our first result can be described using the following theorem.

▶ **Theorem 1.** *Given an undirected graph $G = (V, E)$, the DFS tree of the graph can be computed by a semi-streaming algorithm in at most n/k passes using $O(nk)$ space, requiring $O(m\alpha(m, n))$ time per pass.*

As described earlier, a simple algorithm can compute the DFS tree in $O(h)$ passes, where h is the height of the DFS tree. Thus, for the graphs having a DFS tree with height $h = o(n)$ (see full paper for details), we improve our result for such graphs in the following theorem.

▶ **Theorem 2.** *Given an undirected graph G, a DFS tree of G can be computed by a semi-streaming algorithm using $\lceil h/k \rceil$ passes using $O(nk)$ space requiring amortized $O(m + nk)$ time per pass for any integer $k \leq h$, where h is the height of the computed DFS tree.*[1]

Since typically the space allowed in the semi-streaming model is $O(n \; poly \log n)$, the improvement in upper bounds of the problem by our results is considerably small (upto $poly \log n$ factors). Recently, Elkin [12] presented the first $o(n)$ pass algorithm for computing Shortest Paths Trees. Using $O(nk)$ local space, it computes the shortest path tree from a given source in $O(n/k)$ passes for unweighted graphs, and in $O(n \log n/k)$ passes for weighted graphs. The significance of such results, despite improving the upper bounds by only small factors, is substantial because they address fundamental problems. The lack of any progress for such fundamental problems despite several decades of research on streaming algorithms further highlights the significance of such results. Moreover, allowing $O(n^{1+\epsilon})$ space (as in [15, 6, 11]) such results improves the upper bound significantly by $O(n^{\epsilon})$ factors. Furthermore, they demonstrate the existence of a trade-off between the space and number of passes required for computing such fundamental structures.

Our final algorithm is presented in form of a *framework*, which can use any algorithm for maintaining a DFS tree of a stored sparser subgraph, provided that it satisfies the property of *monotonic fall*. Such a *framework* allows more flexibility and is hopefully much easier to extend to better algorithms for computing a DFS tree or other problems requiring a computation of DFS tree. Hence we believe our *framework* would be of independent interest.

We also augment our theoretical analysis with the experimental evaluation of our proposed algorithms. For both random and real graphs, the algorithms require merely a *few* passes even when the allowed space is just $O(n)$. The exceptional performance and surprising observations of our experiments on random graphs might also be of independent interest.

1.2 Overview

We now briefly describe the outline of our paper. In Section 2 we establish the terminology and notations used in the remainder of the paper. In order to present the main ideas behind our approach in a simple and comprehensible manner, we present the algorithm in four stages. *Firstly* in Section 3, we describe the basic algorithm to build a DFS tree in n passes, which adds a new vertex to the DFS tree in every pass over the input stream. *Secondly* in Section 3.1, we improve this algorithm to compute a DFS tree in h passes, where h is the height of the final DFS tree. This algorithm essentially computes all the vertices in the next level of the currently built DFS tree simultaneously, building the DFS tree by one level in each pass over the input stream. Thus, in the i^{th} pass every vertex on the i^{th} level of the DFS tree is computed. *Thirdly* in Section 4, we describe an advanced algorithm which uses

[1] Note that there can be many DFS trees of a graph having varying heights, say h_{min} to h_{max}. Our algorithm does not guarantee the computation of DFS tree having minimum height h_{min}, rather it simply computes a *valid* DFS tree with height h, where $h_{min} \leq h \leq h_{max}$.

$O(nk)$ space to add a path of length at least k to the DFS tree in every pass over the input stream. Thus, the complete DFS tree can be computed in $\lceil n/k \rceil$ passes. *Finally*, in Section 5, we improve the algorithm to simultaneously add all the subtrees constituting the next k levels of the final DFS tree starting from the leaves of the current tree T. Thus, k levels are added to the DFS tree in each pass over the input stream, computing the DFS tree in $\lceil h/k \rceil$ passes. As described earlier, our final algorithm is presented in form of a *framework* which uses as a black box, any algorithm to maintain a DFS tree of a stored sparser subgraph, satisfying certain properties. In the interest of completeness, one such algorithm is described in the full paper. *Lastly* in Section 6, we present the results of the experimental evaluation of these algorithms. The details of this evaluation are deferred to the full version of the paper.

In our advanced algorithms, we employ two interesting properties of a DFS tree, namely, the *components* property [7] and the *min-height* property. These simple properties of any DFS tree prove crucial in building the DFS efficiently in the streaming environment.

2 Preliminaries

Let $G = (V, E)$ be an undirected connected graph having n vertices and m edges. The DFS traversal of G starting from any vertex $r \in V$ produces a spanning tree rooted at r called a DFS tree, in $O(m + n)$ time. For any rooted spanning tree of G, a non-tree edge of the graph is called a *back edge* if one of its endpoints is an ancestor of the other in the tree, else it is called a *cross edge*. A necessary and sufficient condition for any rooted spanning tree to be a DFS tree is that every non-tree edge is a back edge.

In order to handle disconnected graphs, we add a dummy vertex r to the graph and connect it to all vertices. Our algorithm computes a DFS tree rooted at r in this augmented graph, where each child subtree of r is a DFS tree of a connected component in the DFS forest of the original graph. The following notations will be used throughout the paper.

- T : The DFS tree of G incrementally computed by our algorithm.
- $par(w)$: Parent of w in T.
- $T(x)$: The subtree of T rooted at vertex x.
- $root(T')$: Root of a subtree T' of T, i.e., $root\big(T(x)\big) = x$.
- $level(v)$: Level of vertex v in T, where $level(root(T)) = 0$ and $level(v) = level(par(v))+1$.

In this paper we will discuss algorithms to compute a DFS tree T for the input graph G in the semi-streaming model. In all the cases T will be built iteratively starting from an empty tree. At any time during the algorithm, we shall refer to the vertices that are not a part of the DFS tree T as *unvisited* and denote them by V', i.e., $V' = V \setminus T$. Similarly, we refer to the subgraph induced by the *unvisited* vertices, $G' = G(V')$, as the *unvisited graph*. Unless stated otherwise, we shall refer to a connected component of the unvisited graph G' as simply a *component*. For any component C, the set of edges and vertices in the component will be denoted by E_C and V_C. Further, each component C maintains a spanning tree of the component that shall be referred as T_C. We refer to a path p in a DFS tree T as an *ancestor-descendant* path if one of its endpoints is an ancestor of the other in T. Since the DFS tree grows downwards from the root, a vertex u is said to be *higher* than vertex v if $level(u) < level(v)$. Similarly, among two edges incident on an ancestor-descendant path p, an edge (x, y) is *higher* than edge (u, v) if $y, v \in p$ and $level(y) < level(v)$.

We shall now describe two invariants such that any algorithm computing DFS tree incrementally satisfying these invariants at every stage of the algorithm, ensures the absence of cross edges in T and hence the correctness of the final DFS tree T.

Invariants:

$\mathcal{I}_1$: All non-tree edges among vertices in T are back edges, and

$\mathcal{I}_2$: For any component C of the unvisited graph, all the edges from C to the partially built DFS tree T are incident on a single *ancestor-descendant* path of T.

We shall also use the *components property* by Baswana et al. [7], described as follows.

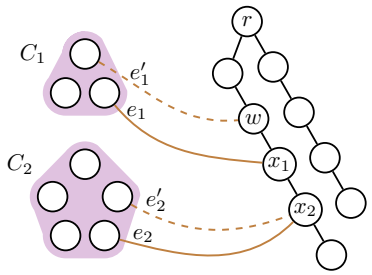


Figure 1 Edges e'_1 and e'_2 can be ignored during the DFS traversal (reproduced from [7]).

▶ **Lemma 3** (Components Property [7]). *Consider a partially completed DFS traversal where T is the partially built DFS tree. Let the connected components of G' be $C_1, .., C_k$. Consider any two edges e_i and e'_i from C_i that are incident respectively on a vertex x_i and some ancestor (not necessarily proper) w of x_i in T. Then it is sufficient to consider only e_i during the DFS traversal, i.e., the edge e'_i can be safely ignored.*

Ignoring e'_i during the DFS traversal, as stated in the components property, is justified because e'_i will appear as a back edge in the resulting DFS tree (refer to Figure 1). For each component C_i, the edge e_i can be found using a single pass over all the graph edges.

3 Simple Algorithms

We shall first briefly describe the trivial algorithm to compute a DFS tree of a (directed) graph using n passes. Since we are limited to have only $O(n \, poly \log n)$ space, we cannot store the adjacency list of the vertices in the graph. Recall that in the standard DFS algorithm [35], after visiting a vertex v, we choose any unvisited neighbour of v and visit it. If no neighbour of v is unvisited, the traversal retreats back to the parent of v and look for its unvisited neighbour, and so on.

In the streaming model, we can use the same algorithm. However, we do not store the adjacency list of a vertex. To find the unvisited neighbour of each vertex, we perform a complete pass over the edges in E. The algorithm only stores the partially built DFS tree and the status of each vertex (whether it is visited/added to T). Thus, for each vertex v (except r) one pass is performed to add v to T and another is performed before retreating to the parent of v. Hence, it takes $2(n-1)$ passes to complete the algorithm since T is initialized with the root r. Since, this procedure essentially simulates the standard DFS algorithm [35], it clearly satisfies the invariants $\mathcal{I}_1$ and $\mathcal{I}_2$.

This procedure can be easily transformed to require only $n-1$ passes by avoiding an extra pass for retreating from each vertex v. In each pass we find an edge e (from the stream) from the unvisited vertices, V', to the lowest vertex on the ancestor-descendant path connecting r and v, i.e., closest to v. Hence e would be an edge from the lowest (maximum level) ancestor of v (not necessarily proper) having at least one unvisited neighbour. Recall that if v does

not have an unvisited neighbour we move to processing its parent, and so on until we find an ancestor having an unvisited neighbour. We can thus directly add the edge e to T. Hence, retreating from a vertex would not require an additional pass and the overall procedure can be completed in $n-1$ passes, each pass adding a new vertex to T. Moreover, this also requires $O(1)$ processing time per edge and extra $O(n)$ time at the end of the pass, to find the relevant ancestor. Refer to the full paper for the pseudocode of the procedure. Thus, we get the following result.

▶ **Theorem 4.** *Given a directed/undirected graph G, a DFS tree of G can be computed by a semi-streaming algorithm in n passes using $O(n)$ space, using $O(m)$ time per pass.*

3.1 Improved algorithm

We shall now describe how this simple algorithm can be improved to compute a DFS tree of an undirected graph in h passes, where h is the height of the computed DFS tree. The main idea behind this approach is that each component of the unvisited graph G' will constitute a separate subtree of the final DFS tree. Hence each such subtree can be computed independent of each other in parallel (this idea was also used by [28]).

Using one pass over edges in E, the components of the unvisited graph G' can be found by using Union-Find algorithm [37, 39] on the edges E' of G'. Now, using the *components* property we know that it is sufficient to add the lowest edge from each component to the DFS tree T. At the end of the pass, for each component C we find the edge (x_C, y_C) incident from the lowest vertex $x_C \in T$ to some vertex $y_C \in V_C$ and add it to T. Note that in the next pass, for each component of $C \setminus \{y_C\}$ the lowest edge connecting it to T would necessarily be incident on y_C as C was connected. Hence, instead of lowest edge incident on T, we store e_y from $y \in V'$ only if e_y is incident on some leaf of T. Refer to the full paper for the pseudocode of the algorithm.

To prove the correctness of the algorithm, we shall prove using induction that the invariants $\mathcal{I}_1$ and $\mathcal{I}_2$ hold over the passes performed on E. Since T is initialized as an isolated vertex r, both invariants trivially hold. Now, let the invariants hold at the beginning of a pass. Using $\mathcal{I}_2$, each component C can have edges to a single ancestor-descendant path from r to x_C. Thus, adding the edge (x_C, y_C) for each component C, would not violate $\mathcal{I}_1$ at the end of the pass, given that $\mathcal{I}_1$ holds at the beginning of the pass. Additionally, from each component C we add a single vertex y_C as a child of x_C to T. Hence for any component of $C \setminus \{y_C\}$, the edges to T can only be to ancestors of y_C (using $\mathcal{I}_2$ of previous pass), and an edge necessarily to y_C, satisfying $\mathcal{I}_2$ at the end of the pass. Hence, using induction both $\mathcal{I}_1$ and $\mathcal{I}_2$ are satisfied proving the correctness of our algorithm.

Further, since each component C in any i^{th} pass necessarily has an edge to a leaf x_C of T, the new vertex y_C is added to the i^{th} level of T. This also implies that every vertex at i^{th} level of the final DFS tree is added during the i^{th} pass. Hence, after h passes we get a DFS tree of the whole graph as h is the height of the computed DFS tree.

Now, the total time[2] required to compute the connected components is $O(m\alpha(m,n))$. And computing an edge from each unvisited vertex to a leaf in T requires $O(1)$ time using $O(n)$ space. Thus, we have the following theorem.

[2] The Union-Find algorithm [37, 39] requires $O(m\alpha(m,n))$ time, where $\alpha(m,n)$ is the inverse Ackermann function.

▶ **Theorem 5.** *Given an undirected graph G, a DFS tree of G can be computed by a semi-streaming algorithm in h passes using $O(n)$ space, where h is the height of the computed DFS tree, using $O(m\alpha(m,n))$ time per pass.*

4 Computing DFS in sublinear number of passes

Since a DFS tree may have $O(n)$ height, we cannot hope to compute a DFS tree in sublinear number of passes using the previously described simple algorithms. The main difference between the advanced approaches and the simple algorithms is that, in each pass instead of adding a single vertex (say y) to the DFS tree, we shall be adding an entire path (starting from y) to the DFS tree. The DFS traversal gives the flexibility to chose the next vertex to be visited as long as the DFS property is satisfied, i.e., invariants $\mathcal{I}_1$ and $\mathcal{I}_2$ are maintained.

Hence in each pass we do the following for every component C in G'. Instead of finding a single edge (x_C, y_C) (see Section 3.1), we find a path P starting from y_C in C and attach this entire path P to T (instead of only y_C). Suppose this splits the component C into components $C_1, C_2, \ldots$ of $C \setminus P$. Now, each C_i would have an edge to at least one vertex on P (instead of necessarily the leaf x_C in Section 3.1) since C was a connected component. Hence in this algorithm for each C_i, we find the vertex y_i which is the lowest vertex of T (or P) to which an edge from C_i is incident. Observe that y_i is unique since all the neighbours of C_i in T are along one path from the root to a leaf. Using the *components* property, the selection of y_i as the parent of the root of the subtree to be computed for C_i ensures that invariant $\mathcal{I}_2$ continues to hold. Thus, in each pass from every component of the unvisited graph, we shall extract a path and add it to the DFS tree T.

This approach thus allows T to grow by more than one vertex in each pass which is essential for completing the tree in $o(n)$ passes. If in each pass we add a path of length at least k from each component of G', then the tree will grow by at least k vertices in each pass, requiring overall $\lceil n/k \rceil$ passes to completely build the DFS tree. We shall now present an important property of any DFS tree of an undirected graph, which ensures that in each pass we can find a path of length at least $k \geq m/n$.

▶ **Lemma 6** (Min-Height Property)**.** *Given a connected undirected graph G having m edges, any DFS tree of G from any root vertex necessarily has a height $h \geq m/n$.*

Proof. We know that each non-tree edge in a DFS tree of an undirected graph is a *back edge*. We shall associate each edge to its lower endpoint. Thus, in a DFS tree each vertex will be associated to a tree edge to its parent and back edges only to its ancestors. Now, each vertex can have only h ancestors as the height of the DFS tree is h, Hence each vertex has only h edges associated to it resulting in less than nh edges, i.e. $m \leq nh$ or $h \geq m/n$. Note that it is important for the graph to be connected otherwise from some root the corresponding component and hence its DFS tree can be much smaller. ◀

4.1 Algorithm

We shall now describe our algorithm to compute a DFS tree of the input graph in $o(n)$ passes. Let the maximum space allowed for computation in the semi-streaming model be $O(nk)$. The algorithm is a recursive procedure that computes a DFS tree of a component C from a root r_C. For each component C we maintain a spanning tree T_C of C. Initially we can perform a single pass over E to compute a spanning tree of the connected graph G (recall the assumption in Section 2) using the Union-Find algorithm. For the remaining components, its spanning tree would already have been computed and passed as an input to the algorithm.

We initiate a pass over the edge in E and store the first $|V_C| \cdot k$ edges (if possible) from the component C in the memory as the subgraph E'_C. Before proceeding with the remaining stream, we use any algorithm for computing a DFS tree T'_C rooted at r_C in the subgraph containing edges from T_C and E'_C. Note that adding T_C to E'_C is important to ensure that subgraph induced by $T_C \cup E'_C$ is connected. In case the pass was completed before E'_C exceeded storing $|V_C| \cdot k$ edges, T'_C is indeed a DFS tree of C and we directly add it to T. Otherwise, we find the longest path P from T'_C starting from r_C, i.e., path from r_C to the farthest leaf. The path P is then added to T.

Now, we need to compute the connected components of $C \setminus P$ and the new corresponding root for each such component. We use the Union-Find algorithm to compute these components, say $C_1, ..., C_f$, and compute the lowest edge e_i from each C_i on the path P. Clearly, there exist such an edge as C was connected. In order to find these components and edges, we need to consider all the edges in E_C, which can be done by first considering E'_C and then each edge from C in the remainder of input stream of the pass. Refer to the full paper for the pseudocode of the algorithm.

Using the components property, choosing the new root y_i corresponding to the lowest edge e_i ensures that the invariant $\mathcal{I}_2$ and hence $\mathcal{I}_1$ is satisfied. Now, in case $|E_C| < |V_C| \cdot k$, the entire DFS tree of C is constructed and added to T in a single pass. Otherwise, in each pass we add the longest path P from T'_C to the final DFS tree T. Since $|E'_C| = |V_C| \cdot k$ and $E'_C \cup T_C$ is a single connected component, the *min-height property* ensures that the height of any such T'_C (and hence P) is at least k. Since in each pass, except the last, we add at least k new vertices to T, this algorithm terminates in at most $\lceil n/k \rceil$ passes. Now, the total time required to find the components of the unvisited graph is again $O(m\alpha(m, n))$. The remaining operations clearly require $O(|E_C|)$ time for a component C, requiring overall $O(m)$ time. Thus, we get the following theorem.

▶ **Theorem 1.** *Given an undirected graph G, a DFS tree of G can be computed by a semi-streaming algorithm in at most $\lceil n/k \rceil$ passes using $O(nk)$ space, requiring $O(m\alpha(m, n))$ time per pass.*

▶ Remark 7. Since, the algorithm adds an ancestor-descendant path for each component of G', it might seem that the analysis of the algorithm is not tight for computing DFS trees with $o(n)$ height. However, there exist a sequence of input edges where the algorithm indeed takes $\Theta(n/k)$ passes for computing a DFS tree with height $o(n)$. The details of the sequence are described in the full version of the paper.

5 Final algorithm

We shall now further improve the algorithm so that the required number of passes reduces to $\lceil h/k \rceil$ while it continues to use $O(nk)$ space, where h is the height of the computed DFS tree and k is any positive integer. To understand the main intuition behind our approach, let us recall the previously described algorithms. We first described a simple algorithm (in Section 3) in which every pass over the input stream adds one new vertex as the child of some leaf of T, which was improved (in Section 3.1) to simultaneously adding all vertices which are children of the leaves of T in the final DFS tree. We then presented another algorithm (in Section 4) in which every pass over the input stream adds one *ancestor-descendant* path of length k or more, from each component of G' to T. We shall now improve it by adding all the subtrees constituting the next k levels of the final DFS tree starting from the leaves of the current tree T (or fewer than k levels if the corresponding component of G' is exhausted).

Now, consider any component C of G'. Let $r_C \in C$ be a vertex having an edge e to a leaf of the partially built DFS tree T. The computation of T can be completed by computing a DFS tree of C from the root r_C, which can be directly attached to T using e. However, computing the entire DFS tree of C may not be possible in a single pass over the input stream, due to the limited storage space available. Thus, using $O(n \cdot k)$ space we compute a special spanning tree T_C for each component C of G' in parallel, such that the top k levels of T_C is same as the top k levels of *some* DFS tree of C. As a result, in the i^{th} pass all vertices on the levels $(i-1) \cdot k + 1$ to $i \cdot k$ of the final DFS tree are added to T. This essentially adds a tree T'_C representing the top k levels of T_C for each component C of G'. This ensures that our algorithm will terminate in $\lceil h/k \rceil$ passes, where h is the height of the final DFS tree. Further, this special tree T_C also ensures an *additional* property, i.e., there is a one to one correspondence between the set of trees of $T_C \setminus T'_C$ and the components of $C \setminus T'_C$. In fact, each tree of $T_C \setminus T'_C$ is a spanning tree of the corresponding component. This property directly provides the spanning trees of the components of G' in the next pass.

Special spanning tree T_C

We shall now describe the properties of this special tree T_C (and hence T'_C) which is computed in a single pass over the input stream. For T'_C to be added to the DFS tree T of the graph, a necessary and sufficient condition is that T'_C satisfies the invariants $\mathcal{I}_1$ and $\mathcal{I}_2$ at the end of the pass. To achieve this we maintain T_C to be a spanning tree of C, such that these invariants are maintained by the corresponding T'_C throughout the pass as the edges are processed. Let S_C be the set of edges already visited during the current pass, which have both endpoints in C. In order to satisfy $\mathcal{I}_1$, no edge in S_C should be a cross edge in T'_C, i.e., no edge having both endpoints in the top k levels of T_C is a cross edge. In order to satisfy $\mathcal{I}_2$, no edge in S_C from any component $C' \in C \setminus T'_C$ to $C \setminus C'$ should be a cross edge in T_C. Hence, using the *additional* property of T_C, each edge from a tree τ in $T_C \setminus T'_C$ to $T_C \setminus \tau$ is necessarily a back edge. This is captured by the two conditions of invariant $\mathcal{I}_T$ given below. Hence $\mathcal{I}_T$ should hold after processing each edge in the pass. Observe that any spanning tree, T_C, trivially satisfies $\mathcal{I}_T$ at the beginning of the pass as $S_C = \emptyset$.

Invariant $\mathcal{I}_T$:
T_C is a spanning tree of C with the top k levels being T'_C such that:
$\mathcal{I}_{T_1}$: All non-tree edges of S_C having both endpoints in T'_C, are back edges.
$\mathcal{I}_{T_2}$: For each tree τ in $T_C \setminus T'_C$, all the edges of S_C from τ to $T_C \setminus \tau$ are back edges.

Thus, $\mathcal{I}_T$ is the local invariant maintained by T_C during the pass, so that the global invariants $\mathcal{I}_1$ and $\mathcal{I}_2$ are maintained throughout the algorithm. Now, in order to compute T_C (and hence T'_C) satisfying the above invariant, we store a subset of S_C along with T_C. Let H_C denote the (spanning) subgraph of G formed by T_C along with these additional edges. Note that all the edges of S_C cannot be stored in H_C due to space limitation of $O(nk)$. Since each pass starts with the spanning tree T_C of C and $S_C = \emptyset$, initially $H_C = T_C$. As the successive edges of the stream are processed, H_C is updated if the input edge belongs to the component C. We now formally describe H_C and its properties.

Spanning subgraph H_C

As described earlier, at the beginning of a pass for every component C of G', $H_C = T_C$. Now, the role of H_C is to facilitate the maintenance of the invariant $\mathcal{I}_T$. In order to satisfy $\mathcal{I}_{T_1}$ and $\mathcal{I}_{T_2}$, we store in H_C all the edges in S_C that are incident on at least one vertex of

T'_C. Therefore, H_C is the spanning tree T_C along with every edge in S_C which has at least one endpoint in T'_C. Thus, H_C satisfies the following invariant throughout the algorithm.

Invariant $\mathcal{I}_H$:
H_C comprises of T_C and all edges from S_C that are incident on at least one vertex of T'_C.

We shall now describe a few properties of H_C and then in the following section show that maintaining $\mathcal{I}_H$ for H_C is indeed sufficient to maintain the invariant $\mathcal{I}_T$ as the stream is processed. The following properties of H_C are crucial to establish the correctness of our procedure to maintain T_C and H_C and establish a bound on total space required by H_C.

▶ **Lemma 8.** *T_C is a valid DFS tree of H_C.*

Proof. In order to prove this claim it is sufficient to prove that all the non-tree edges stored in H_C are back edges in T_C, i.e., the endpoints of every such edge share an ancestor-descendant relationship. Now, invariant $\mathcal{I}_{T_1}$ ensures that any edge in S_C having both endpoints in T'_C is a back edge. And invariant $\mathcal{I}_{T_2}$ ensures that any edge between a vertex in T'_C and $T_C \setminus T'_C$ is a back edge. Hence, all the non-tree edges incident on T'_C (and hence all non-tree edges in H_C) are back edges, proving our lemma. ◀

▶ **Lemma 9.** *The total number of edges in H_C, for all the components C of G', is $O(nk)$.*

Proof. The size of H_C can be analysed using invariant $\mathcal{I}_H$ as follows. The number of tree edges in T_C (and hence in H_C) is $O(|V_C|)$. The non-tree edges stored by H_C have at least one endpoint in T'_C. Using Lemma 8 we know that all these edges are back edges. To bound the number of such edges let us associate each non-tree edge to its lower endpoint. Hence each vertex will be associated to at most k non-tree edges to its k ancestors in T'_C (recall that T'_C is the top k levels of T_C). Thus, H_C stores $O(|V_C|)$ tree edges and $O(|V_C| \cdot k)$ non-tree edges, i.e., total $O(|V_C| \cdot k)$ edges. Since $\sum_{C \in G'} |V_C| \leq n$, the total number of edges in H_C is $O(nk)$. ◀

5.1 Processing of Edges

We now describe how T_C and H_C are maintained while processing the edges of the input stream such that $\mathcal{I}_T$ and $\mathcal{I}_H$ are satisfied. Since our algorithm maintains the invariants $\mathcal{I}_1$ and $\mathcal{I}_2$ (because of $\mathcal{I}_T$), we know that any edge whose both endpoints are not in some component C of G', is either a back edge or already a tree edge in T. Thus, we shall only discuss the processing of an edge (x, y) having both endpoints in C (now added to S_C), where $level(x) \leq level(y)$.

1. If $x \in T'_C$ then the edge is added to H_C to ensure $\mathcal{I}_H$. In addition, if (x, y) is a cross edge in T_C it violates either $\mathcal{I}_{T_1}$ (if $y \in T'_C$) or $\mathcal{I}_{T_2}$ (if $y \notin T'_C$). Thus, T_C is required to be restructured to ensure that $\mathcal{I}_T$ is satisfied.
2. If $x \notin T'_C$ and if x and y belong to different trees in $T_C \setminus T'_C$, then it violates $\mathcal{I}_{T_2}$. Again in such a case, T_C is required to be restructured to ensure that $\mathcal{I}_T$ is satisfied.

Note that after restructuring T_C we need to update H_C such that $\mathcal{I}_H$ is satisfied. Consequently any non-tree edge in H_C that was incident on a vertex in original T'_C, has to be removed from H_C if none of its endpoints are in T'_C after restructuring T_C, i.e., one or both of its endpoints have moved out of T'_C. But the problem arises if a vertex moves into T'_C during restructuring. There might have been edges incident on such a vertex in S_C and which were not stored in H_C. In this case we need these edges in H_C to satisfy $\mathcal{I}_H$, which is

not possible without visiting S_C again. This problem can be avoided if our restructuring procedure ensures that no new vertex enters T'_C. This can be ensured if the restructuring procedure follows the property of *monotonic fall*, i.e., the level of a vertex is never decreased by the procedure. Let e be the new edge of component C in the input stream. We shall show that in order to preserve the invariants $\mathcal{I}_T$ and $\mathcal{I}_H$ it is sufficient that the restructuring procedure maintains the property of *monotonic fall* and ensures that the restructured T_C is a DFS tree of $H_C + e$.

▶ **Lemma 10.** *On insertion of an edge e, any restructuring procedure which updates T_C to be a valid DFS tree of $H_C + e$ ensuring* monotonic fall*, satisfies the invariants $\mathcal{I}_T$ and $\mathcal{I}_H$.*

Proof. The property of *monotonic fall* ensures that the vertex set of new T'_C is a subset of the vertex set of the previous T'_C. Using $\mathcal{I}_H$ we know that any edge of S_C which is not present in H_C must have both its endpoints outside T'_C. Hence, *monotonic fall* guarantees that $\mathcal{I}_H$ continues to hold with respect to the new T'_C for the edges in $S_C \setminus \{e\}$. Additionally, we save e in the new H_C if at least one of its endpoints belong to the new T'_C, ensuring that $\mathcal{I}_H$ holds for the entire S_C.

Since the restructuring procedure ensures that the updated T_C is a DFS tree of H_C, the invariant $\mathcal{I}_{T_1}$ trivially holds as a result of $\mathcal{I}_H$. In order to prove the invariant $\mathcal{I}_{T_2}$, consider any edge $e' \in S_C$ from a tree $\tau \in T_C \setminus T'_C$ to $T_C \setminus \tau$. Clearly, it will satisfy $\mathcal{I}_{T_2}$ if $e' \in H_C$, as T_C is a DFS tree of $H_C + e$. In case $e' \notin H_C$, it must be internal to some tree τ' in the original $T_C \setminus T'_C$ (using $\mathcal{I}_{T_2}$ in the original T_C). We shall now show that such an edge will remain internal to some tree in the updated $T_C \setminus T'_C$ as well, thereby not violating $\mathcal{I}_{T_2}$. Clearly the endpoints of e' cannot be in the updated T'_C due to the property of *monotonic fall*.

Assume that the endpoints of e' belong to different trees of updated $T_C \setminus T'_C$. Now, consider the edges $e_1, ..., e_t$ on the tree path in τ' connecting the endpoints of e'. Since the entire tree path is in τ', the endpoints of each e_i are not in original T'_C, ensuring that they are also not in the updated T'_C (by *monotonic fall*). Since the endpoints of e' (and hence the endpoints of the path $e_1, ..., e_t$) are in different trees in updated $T_C \setminus T'_C$, there must exist some e_i which also has endpoints belonging to different trees of updated $T_C \setminus T'_C$. This makes e_i a cross edge of the updated T_C. Since e_i is a tree edge of original T_C, it belongs to H_C and hence e_i being a cross edge implies that the updated T_C is not a DFS tree of $H_C + e$, which is a contradiction. Hence e' has both its endpoints in the same tree of the updated $T_c \setminus T'_C$, ensuring that $\mathcal{I}_{T_2}$ holds after the restructuring procedure. ◀

Hence, any procedure to restructure a DFS tree T_C of the subgraph H_C on insertion of a *cross edge* e, that upholds the property of *monotonic fall* and returns a new T_C which is a DFS tree of $H_C + e$, can be used as a *black box* in our algorithm. One such algorithm is the incremental DFS algorithm by Baswana and Khan [8], which precisely fulfils our requirement. They proved the total update time of the algorithm to be $O(n^2)$. They also showed that any algorithm maintaining incremental DFS abiding *monotonic fall* would require $\Omega(n^2)$ time even for sparse graphs, if it explicitly maintains the DFS tree. If the height of the DFS tree is known to be h, these bounds reduces to $O(nh + n_e)$ and $\Omega(nh + n_e)$ respectively, where n_e is the number of edges processed by the algorithm. Refer to the full paper for a brief description of the algorithm.

5.2 Algorithm

We now describe the details of our final algorithm which uses an incremental DFS algorithm [8] for restructuring the DFS tree when a cross edge is inserted. Similar to the algorithm in Section 4, for each component C of G', a rooted spanning tree T_C of the component is required as an input to the procedure having the root r_C.

Initially $T = \emptyset$ and $G' = G$ has a single component C, as G is connected (recall the assumption in Section 2). Hence for the first pass, we compute a spanning tree T_C of G using the Union-Find algorithm. Subsequently in each pass we directly get a spanning tree $T_{C'}$ for each component C' of the new G', which is the corresponding tree in $T_C \setminus T'_C$, where C is the component containing C' in the previous pass. Also, observe that the use of these trees as the new T_C ensures that the level of no vertex ever rises in the context of the entire tree T. This implies that the level of any vertex starting with the initial spanning tree T_G never rises, i.e., the entire algorithm satisfies the property of *monotonic fall*. We will use this fact crucially in the analysis of the time complexity.

As described earlier, we process the edges of the stream by updating the T_C and H_C maintaining $\mathcal{I}_T$ and $\mathcal{I}_H$ respectively. In case the edge is internal to some tree in $T_C \setminus T'_C$ (i.e., have both endpoint in the same tree in $T_C \setminus T'_C$), we simply ignore the edge. Otherwise, we add it to H_C to satisfy $\mathcal{I}_H$. Further, the incremental DFS algorithm [8] maintains T_C to be a DFS tree of H_C, which restructures T_C if the processed edge is added to H_C and is a cross edge in T_C. Now, in case T_C is updated we also update the subgraph H_C, by removing the extra non-tree edges having both endpoints in $T_C \setminus T'_C$. After the pass is completed, we attach T'_C (the top k levels of T_C) to T. Now, $\mathcal{I}_{T_2}$ ensures that each tree of $T_C \setminus T'_C$ forms the (rooted) spanning tree of the components of the new G', and hence can be used for the next pass. Refer to the full paper for the pseudocode of the algorithm.

5.3 Correctness and Analysis

The correctness of our algorithm follows from Lemma 10, which ensures that invariants $\mathcal{I}_H$ and $\mathcal{I}_T$ (and hence $\mathcal{I}_1$ and $\mathcal{I}_2$) are maintained as a result of using the incremental DFS algorithm which ensures *monotonic fall* of vertices. The total space used by our algorithm and the restructuring procedure is dominated by the cumulative size of H_C for all components C of G', which is $O(nk)$ using Lemma 9. Now, in every pass of the algorithm, a DFS tree for each component C of height k is attached to T. These trees collectively constitute the next k levels of the final DFS tree T. Therefore, the entire tree T is computed in $\lceil h/k \rceil$ passes.

Let us now analyse the time complexity of our algorithm. In the first pass $O(m\alpha(m,n))$ time is required to compute the spanning tree T_C using the Union-Find algorithm. Also, in each pass $O(m)$ time is required to process the input stream. Further, in order to update H_C we are required to delete edges having both endpoints out of T'_C. Hence, whenever a vertex falls below the k^{th} level, the edges incident on it are checked for deletion from H_C (if the other endpoint is also not in T'_C). Total time required for this is $O(\sum_{v \in V} deg(v)) = O(m)$ per pass. In the full paper we describe the details of an incremental DFS algorithm which maintains the DFS tree in total $O(nh + n_e)$ time, where $n_e = O(mh/k)$, for processing the entire input stream in each pass.

Finally, we need to efficiently answer the *query* whether an edge is internal to some tree in $T_C \setminus T'_C$. For this we maintain for each vertex x its ancestor at level k as $rep[x]$, i.e., $rep[x]$ is the root of the tree in $T_C \setminus T'_C$ that contains x. If $level(x) < k$, then $rep[x] = x$. For an edge (x, y) comparing the $rep[x]$ and $rep[y]$ efficiently answers the required query in $O(1)$ time. However, whenever T_C is updated we need to update $rep[v]$ for each vertex v in the

modified part of T_C, requiring $O(1)$ time per vertex in the modified part of T_C. We shall bound the total work done to update $rep[x]$ of such a vertex x throughout the algorithm to $O(nh)$ as follows.

Consider the potential function $\Phi = \sum_{v \in V} level(v)$. Whenever some part of T_C is updated, each vertex x in the modified T_C necessarily incurs a fall in its level (due to *monotonic fall*). Thus, the cost of updating $rep[x]$ throughout the algorithm is proportional to the number of times x descends in the tree, hence increases the value of Φ by at least one unit. Hence, updating $rep[x]$ for all x in the modified part of T_C can be accounted by the corresponding increase in the value of Φ. Clearly, the maximum value of Φ is $O(nh)$, since the level of each vertex is always less than h, where h is the height of the computed DFS tree. Thus, the total work done to update $rep[x]$ for all $x \in V$ is $O(nh)$. This proves our main theorem described in Section 1.1 which is stated as follows.

▶ **Theorem 2.** *Given an undirected graph G, a DFS tree of G can be computed by a semi-streaming algorithm using $\lceil h/k \rceil$ passes using $O(nk)$ space requiring amortized $O(m + nk)$ time per pass for any integer $k \leq h$, where h is the height of the computed DFS tree.*

▶ Remark 11. Note that the time complexity of our algorithm is indeed tight for our framework. Since our algorithm requires $\lceil h/k \rceil$ passes and any restructuring procedure following *monotonic fall* requires $\Omega(nh + n_e)$ time, each pass would require $\Omega(m + nk)$ time.

6 Experimental Evaluation

Most streaming algorithms deal with only $O(n)$ space, for which our advanced algorithms improve over the simple algorithms theoretically by just constant factors. However, their empirical performance demonstrates their significance in the real world applications. The evaluation of our algorithms on random and real graphs shows that in practice these algorithms require merely a *few* passes even when allowed to store just $5n$ edges. The results of our analysis can be summarized as follows (for details refer to the full paper).

The two advanced algorithms *kPath* (algorithmin Section 4) and *kLev* (algorithm in Section 5 with an additional heuristic) perform much better than the rest even when $O(n)$ space is allowed. For both random and real graphs, *kPath* performs slightly worse as the density of the graph increases. On the other hand *kLev* performs slightly better only in random graphs with the increasing density. The effect of the space parameter is very large on *kPath* from $k = 1$ to small constants, requiring very *few* passes even for $k = 5$ and $k = 10$. However, *kLev* seems to work very well even for $k = 1$ and has a negligible effect of increasing the value of k. Overall, the results suggest using *kPath* if nk space is allowed for k being a small constant such as 5 or 10. However, if the space restrictions are extremely tight it is better to use *kLev*.

7 Conclusion

We presented the first $o(n)$ pass semi-streaming algorithm for computing a DFS tree for an undirected graph, breaking the long standing presumed barrier of n passes. In our streaming model we assume that $O(nk)$ local space is available for computation, where k is any natural number. Our algorithm computes a DFS tree in $\lceil n/k \rceil$ passes. We improve our algorithm to require only $\lceil h/k \rceil$ passes without any additional space requirement, where h is the height of the final tree. This improvement becomes significant for graphs having shallow DFS trees. Moreover, our algorithm is described as a *framework* using a restructuring algorithm as a

black box. This allows more flexibility to extend our algorithm for solving other problems requiring a computation of DFS tree in the streaming environment.

Recently, in a major breakthrough Elkin [12] presented the first $o(n)$ pass algorithm for computing Shortest Paths Tree from a single source. Using $O(nk)$ local space, it computes the shortest path tree from a given source in $O(n/k)$ passes for unweighted graphs and in $O(n \log n/k)$ passes for weighted graphs.

Despite the fact that these breakthroughs provide only minor improvements (typically $poly \log n$ factors), they are significant steps to pave a path in better understanding of such fundamental problems in the streaming environment. These simple improvements come after decades of the emergence of streaming algorithms for graph problems, where such problems were considered implicitly hard in the semi-streaming environment. We thus believe that our result is a significant improvement over the known algorithm for computing a DFS tree in the streaming environment, and it can be a useful step in more involved algorithms that require the computation of a DFS tree.

Moreover, the experimental evaluation of our algorithms revealed exceptional performance of the advanced algorithms *kPath* and *kLev* (greatly affected by the *additional heuristic*). Thus, it would be interesting to further study these algorithms theoretically which seem to work extremely well in practice.

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013.

2 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.

3 Giorgio Ausiello, Donatella Firmani, and Luigi Laura. Datastream computation of graph biconnectivity: Articulation Points, Bridges, and Biconnected Components. In *Theoretical Computer Science, 11th Italian Conference, ICTCS 2009, Cremona, Italy, September 28-30, 2009, Proceedings.*, pages 26–29, 2009.

4 Giorgio Ausiello, Donatella Firmani, and Luigi Laura. Real-time monitoring of undirected networks: Articulation points, bridges, and connected and biconnected components. *Networks*, 59(3):275–288, 2012. doi:10.1002/net.21450.

5 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 623–632, 2002.

6 Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

7 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in Undirected Graphs: breaking the O(m) barrier. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 730–739, 2016.

8 Surender Baswana and Shahbaz Khan. Incremental Algorithm for Maintaining a DFS Tree for Undirected Graphs. *Algorithmica*, 79(2):466–483, 2017.

9 Glencora Borradaile, Claire Mathieu, and Theresa Migler. Lower bounds for testing digraph connectivity with one-pass streaming algorithms. *CoRR*, abs/1404.1323, 2014.

10 Adam L. Buchsbaum, Raffaele Giancarlo, and Jeffery Westbrook. On finding common neighborhoods in massive graphs. *Theor. Comput. Sci.*, 299(1-3):707–718, 2003.

11 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011.

12 Michael Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 757–770, 2017.

13 Shimon Even and Robert Endre Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975.

14 Martin Farach-Colton, Tsan-sheng Hsu, Meng Li, and Meng-Tsung Tsai. Finding Articulation Points of Large Graphs in Linear Time. In *Algorithms and Data Structures, WADS*, pages 363–372, Cham, 2015. Springer International Publishing.

15 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph Distances in the Streaming Model: The Value of Space. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 745–754, 2005.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On Graph Problems in a Semi-streaming Model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005.

17 Joan Feigenbaum, Sampath Kannan, Martin J. Strauss, and Mahesh Viswanathan. An Approximate L1-Difference Algorithm for Massive Data Streams. *SIAM J. Comput.*, 32(1):131–151, January 2003.

18 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

19 Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and Histograms. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 471–475, 2001.

20 Venkatesan Guruswami and Krzysztof Onak. Superlinear Lower Bounds for Multipass Graph Processing. *Algorithmica*, 76(3):654–683, November 2016.

21 Venkatesan Guruswami and Krzysztof Onak. Superlinear Lower Bounds for Multipass Graph Processing. *Algorithmica*, 76(3):654–683, 2016.

22 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, pages 107–118, 1998.

23 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

24 John E. Hopcroft and Robert Endre Tarjan. Efficient Planarity Testing. *J. ACM*, 21(4):549–568, 1974.

25 Piotr Indyk. Stable Distributions, Pseudorandom Generators, Embeddings, and Data Stream Computation. *J. ACM*, 53(3):307–323, May 2006.

26 Sagar Kale and Sumedh Tirodkar. Maximum Matching in Two, Three, and a Few More Passes Over Graph Streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 15:1–15:21, 2017.

27 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013.

28 Shahbaz Khan. Near Optimal Parallel Algorithms for Dynamic DFS in Undirected Graphs. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 283–292, 2017.

29 Lasse Kliemann. Engineering a Bipartite Matching Algorithm in the Semi-Streaming Model. In *Algorithm Engineering - Selected Results and Surveys*, pages 352–378. Springer International Publishing, 2016.

30 Andrew McGregor. Finding Graph Matchings in Data Streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, pages 170–181, 2005.

31 Andrew McGregor. Graph Stream Algorithms: A Survey. *SIGMOD Rec.*, 43(1):9–20, May 2014.

32 Shan Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

33 Thomas C. O'Connell. A Survey of Graph Algorithms Under Extended Streaming Models of Computation. In *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*, pages 455–476, 2009.

34 Jan Matthias Ruhl. Efficient Algorithms for New Computational Models. *PhD Thesis*, Department of Computer Science, MIT, Cambridge, MA, 2003.

35 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

36 Robert Endre Tarjan. Finding Dominators in Directed Graphs. *SIAM J. Comput.*, 3(1):62–89, 1974.

37 Robert Endre Tarjan. Efficiency of a Good But Not Linear Set Union Algorithm. *J. ACM*, 22(2):215–225, April 1975.

38 Robert Endre Tarjan. Edge-Disjoint Spanning Trees and Depth-First Search. *Acta Inf.*, 6:171–185, 1976.

39 Robert Endre Tarjan and Jan van Leeuwen. Worst-case Analysis of Set Union Algorithms. *J. ACM*, 31(2):245–281, 1984.

40 Jeffery Westbrook and Robert Endre Tarjan. Maintaining Bridge-Connected and Biconnected Components On-Line. *Algorithmica*, 7(5&6):433–464, 1992.

41 Jian Zhang. A Survey on Streaming Algorithms for Massive Graphs. In *Managing and Mining Graph Data*, pages 393–420. Springer US, 2010.

On Finite Monoids over Nonnegative Integer Matrices and Short Killing Words

Stefan Kiefer
University of Oxford, UK

Corto Mascle
ENS Paris-Saclay, France

Abstract

Let n be a natural number and $\mathcal{M}$ a set of $n \times n$-matrices over the nonnegative integers such that $\mathcal{M}$ generates a finite multiplicative monoid. We show that if the zero matrix 0 is a product of matrices in $\mathcal{M}$, then there are $M_1, \ldots, M_{n^5} \in \mathcal{M}$ with $M_1 \cdots M_{n^5} = 0$. This result has applications in automata theory and the theory of codes. Specifically, if $X \subset \Sigma^*$ is a finite incomplete code, then there exists a word $w \in \Sigma^*$ of length polynomial in $\sum_{x \in X} |x|$ such that w is not a factor of any word in X^*. This proves a weak version of Restivo's conjecture.

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

Keywords and phrases matrix semigroups, unambiguous automata, codes, Restivo's conjecture

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.43

Funding *Stefan Kiefer*: Work supported by a Royal Society University Research Fellowship.

1 Introduction

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. In this paper we show the following theorem:

▶ **Theorem 1.** *Let $n \in \mathbb{N}$ and $\mathcal{M} \subseteq \mathbb{N}^{n \times n}$ be a finite set of nonnegative integer matrices. Denote by $\overline{\mathcal{M}}$ the monoid generated by $\mathcal{M}$ under matrix multiplication. If $\overline{\mathcal{M}}$ is finite then there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq \frac{1}{16}n^5 + \frac{15}{16}n^4$ such that the matrix product $M_1 \cdots M_\ell$ has minimum rank in $\overline{\mathcal{M}}$. Further, $M_1, \ldots, M_\ell$ can be computed in time polynomial in the description size of $\mathcal{M}$.*

The mortality problem. Theorem 1 is related to the *mortality* problem for matrices: given a finite set $\mathcal{M}$ of matrices, can the zero matrix (which is defined to have rank 0) be expressed as a finite product of matrices in $\mathcal{M}$? Paterson [14] showed that the mortality problem is undecidable for 3×3 integer matrices, i.e., $\mathcal{M} \subset \mathbb{Z}^{3\times 3}$. It remains undecidable for $\mathcal{M} \subset \mathbb{Z}^{3\times 3}$ with $|\mathcal{M}| = 7$ and for $\mathcal{M} \subset \mathbb{Z}^{21\times 21}$ with $|\mathcal{M}| = 2$, see [8]. Mortality for 2×2 integer matrices is NP-hard [1] and not known to be decidable, see [15] for recent work on the 2×2 case.

The mortality problem for *nonnegative* matrices is much easier, as for each matrix entry it only matters whether it is zero or nonzero, so one can assume $\mathcal{M} \subseteq \{0, 1\}^{n\times n}$. This version is naturally phrased in terms of automata. Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a *nondeterministic finite automaton (NFA)* over a finite alphabet Σ, a finite set Q of states, and with transition function $\delta : Q \times \Sigma \to 2^Q$ (initial and final states do not play a role here). A word $w \in \Sigma^*$ is called *killing word* for $\mathcal{A}$ if w does not label any path in $\mathcal{A}$. Associate to $\mathcal{A}$ the monoid morphism $M_\mathcal{A} : \Sigma^* \to \mathbb{N}^{Q\times Q}$ where for all $a \in \Sigma$ we define $M_\mathcal{A}(a)(p, q) = 1$ if $\delta(p, a) \ni q$ and 0 otherwise. Then, for any word $w \in \Sigma^*$ we have that $M_\mathcal{A}(w)(p, q)$ is the number of w-labelled paths from p to q. It follows that the mortality problem for nonnegative matrices is equivalent to the problem whether an NFA has a killing word. The problem is PSPACE-complete [12], and there are examples where the shortest killing word has exponential length in the number of states of the automaton [6, 12]. This implies that the assumption in Theorem 1 that the

generated monoid $\overline{\mathcal{M}}$ be finite cannot be dropped. Whether $\overline{\mathcal{M}}$ is finite can be checked in polynomial time [11], see also [21] and the references therein. If $\overline{\mathcal{M}}$ is finite then the mortality problem for nonnegative integer matrices is solvable in polynomial time:

▶ **Proposition 2.** *Let $\mathcal{M} \subseteq \mathbb{N}^{n\times n}$ be a finite set of nonnegative integer matrices, generating a finite monoid $\overline{\mathcal{M}}$. One can decide in polynomial time if $0 \in \overline{\mathcal{M}}$.*

Short killing words for unambiguous finite automata. In the central proofs of this paper, the finiteness assumption can be further strengthened so that it corresponds to unambiguousness of NFAs. More precisely, an NFA $\mathcal{A} = (\Sigma, Q, \delta)$ is called an *unambiguous finite automaton (UFA)* if for all states p, q all paths from p to q are labelled by different words, i.e., for each word $w \in \Sigma^*$ there is at most one w-labelled path from p to q. Call a monoid $\overline{\mathcal{M}} \subseteq \mathbb{N}^{n\times n}$ an *unambiguous monoid of relations* if $\overline{\mathcal{M}} \subseteq \{0,1\}^{n\times n}$. For any UFA $\mathcal{A}$ the image $M_{\mathcal{A}}(\Sigma^*)$ of the monoid morphism $M_{\mathcal{A}}$ is an unambiguous monoid of relations, and any unambiguous monoid of relations can be viewed in this way.

Proposition 2 provides a polynomial-time procedure for checking whether a UFA has a killing word. Define ρ as the spectral radius of the rational matrix $\frac{1}{|\Sigma|}\sum_{a\in\Sigma} M(a)$. One can show that $\mathcal{A}$ has a killing word if $\rho < 1$, and otherwise $\rho = 1$. Proposition 2 then follows from the fact that one can compare ρ with 1 in polynomial time. Thus the spectral radius tells whether there *exists* a killing word, but does not *provide* a killing word. Neither does this method imply a polynomial bound on the length of a minimal killing word, let alone a polynomial-time algorithm for computing a killing word. Theorem 1, which is proved purely combinatorially, fills this gap: if there is a killing word, then one can compute a killing word of length $O(|Q|^5)$ in polynomial time. NP-hardness results for approximating the length of a shortest killing word were proved in [17], even for the case $|\Sigma| = 2$ and for *partial DFAs*, which are UFAs with $|\delta(p, a)| \leq 1$ for all $p \in Q$ and all $a \in \Sigma$.

Short minimum-rank words. Define the *rank* of a UFA $\mathcal{A} = (\Sigma, Q, \delta)$ as the minimum rank of the matrices $M_{\mathcal{A}}(w)$ for $w \in \Sigma^*$. A word w such that the rank of $M_{\mathcal{A}}(w)$ attains that minimum is called a *minimum-rank* word. Minimum-rank words have been very well studied for deterministic finite automata (DFAs). DFAs are UFAs with $|\delta(p, a)| = 1$ for all $p \in Q$ and all $a \in \Sigma$. In DFAs of rank 1, minimum-rank words are called *synchronizing* because $\delta(Q, w)$ is a singleton when w is a minimum-rank word. It is the famous Černý conjecture that whenever a DFA has a synchronizing word then it has a synchronizing word of length at most $(n-1)^2$ where $n := |Q|$. There are DFAs whose shortest synchronizing words have that length, but the best known upper bound is cubic in n, see [20] for a survey on the Černý conjecture.

In 1986 Berstel and Perrin generalized the Černý conjecture from DFAs to UFAs by conjecturing [2] that in any UFA a shortest minimum-rank word has length $O(n^2)$. They remarked that no polynomial upper bound was known. Then Carpi [4] showed the following:

▶ **Theorem 3** (Carpi [4])**.** *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a UFA of rank $r \geq 1$ such that the state transition graph of $\mathcal{A}$ is strongly connected. Let $n := |Q| \geq 1$. Then $\mathcal{A}$ has a minimum-rank word of length at most $\frac{1}{2}rn(n-1)^2 + (2r-1)(n-1)$.*

This implies an $O(n^4)$ bound for the case where $r \geq 1$. Carpi left open the case $r = 0$, i.e., when a killing word exists. The main technical contribution of our paper concerns the case $r = 0$. Combined with Carpi's Theorem 3 we then obtain Theorem 1. Theorem 1 provides, to the best of the authors' knowledge, the first polynomial bound, $O(n^5)$, on the length of shortest minimum-rank words for UFAs.

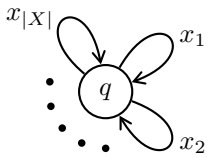


Figure 1 Given a finite language $X \subseteq \Sigma^*$, the flower automaton $\mathcal{A}_X$ has one "petal" for each word $x \in X$. Thus $\delta(q, w) \ni q$ holds if and only if $w \in X^*$. If X is a code then $\mathcal{A}_X$ is unambiguous.

Restivo's conjecture. Let $X \subseteq \Sigma^*$ be a finite set of words over a finite alphabet Σ, and define $k := \max_{x \in X} |x|$. A word $v \in \Sigma^*$ is called *uncompletable* in X if there are no words $u, w \in \Sigma^*$ such that $uvw \in X^*$, i.e., v is not a factor of any word in X^*. In 1981 Restivo [16] conjectured that if there exists an uncompletable word then there is an uncompletable word of length at most $2k^2$. This strong form of Restivo's conjecture has been refuted, with a lower bound of $5k^2 - O(k)$, see [7]. A recent article [10] describes a sophisticated computer-assisted search for sets X with long shortest uncompletable words. While these experiments do not formally disprove a quadratic upper bound in k, they seem to hint at an exponential behaviour in k. See also [5] for recent work and open problems related to Restivo's conjecture.

A set $X \subseteq \Sigma^*$ is called a *code* if every word $w \in X^*$ has at most one decomposition $w = x_1 \cdots x_\ell$ with $x_1, \ldots, x_\ell \in X$. See [3] for a comprehensive reference on codes. For a finite code $X \subseteq \Sigma^*$ define $m := \sum_{x \in X} |x|$. Given X one can construct a *flower automaton* [3, Chapter 4.2], which is a UFA $\mathcal{A}_X = (\Sigma, Q, \delta)$ with $m - |X| + 1$ states, see Figure 1. In this UFA any word is killing if and only if it is uncompletable in X. Hence Theorem 1 implies an $O(m^5)$ bound on the length of the shortest uncompletable word in a finite code. This proves a weak (note that m^5 may be much larger than k^2) version of Restivo's conjecture for finite codes.

Is any product a short product? It was shown in [21] that if $\overline{\mathcal{M}} \subseteq \mathbb{N}^{n \times n}$ is finite then for *every* matrix $M \in \overline{\mathcal{M}}$ there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq \lceil e^2 n! \rceil - 2$ such that $M = M_1 \cdots M_\ell$. It was also shown in [21] that such a bound on ℓ cannot be smaller than 2^{n-2}. In view of Theorem 1 one may ask if a polynomial bound on ℓ exists for *low-rank* matrices M. The answer is no, even for unambiguous monoids of relations and even when M has rank 1 and when 1 is the minimum rank in $\overline{\mathcal{M}}$:

▶ **Theorem 4.** *There is no polynomial p such that the following holds:*

Let $n \in \mathbb{N}$, let $\mathcal{M} \subseteq \{0,1\}^{n \times n}$ generate an unambiguous monoid of relations $\overline{\mathcal{M}} \subseteq \{0,1\}^{n \times n}$. Let $M \in \overline{\mathcal{M}}$ have rank 1, and let 1 be the minimum rank in $\overline{\mathcal{M}}$. Then there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq p(n)$ such that $M = M_1 \cdots M_\ell$.

Thus, while Theorem 1 guarantees that *some* minimum-rank matrix in the monoid is a short product, this is not the case for every minimum-rank matrix in the monoid.

By how much can the $O(n^5)$ upper bound be improved? A *synchronizing 0-automaton* is a DFA $\mathcal{A} = (\Sigma, Q, \delta)$ that has a state $0 \in Q$ and a word $w \in \Sigma^*$ such that $\delta(Q, wx) = \{0\}$ holds for all $x \in \Sigma^*$. The shortest such synchronizing words w are exactly the shortest killing words in the partial DFA obtained from $\mathcal{A}$ by omitting all transitions into the state 0. There exist synchronizing 0-automata with n states where the shortest synchronizing word has length $n(n-1)/2$, and an $\frac{n^2}{4} - 4$ lower bound exists even for synchronizing 0-automata

with $|\Sigma| = 2$ [13]. This implies that the $O(n^5)$ upper bound from Theorem 1 cannot be improved to $o(n^2)$, not even in the case that a killing word exists. One might generalize the Černý conjecture by claiming Theorem 1 with an upper bound of $(n-1)^2$ (note that such a conjecture would concern minimum-rank words, not minimum nonzero-rank words). To the best of the authors' knowledge, this vast generalization of the Černý conjecture has not yet been refuted.

Organization of the paper. In the remaining three sections we prove Proposition 2, Theorem 1, and Theorem 4, respectively.

2 Proof of Proposition 2

Let $\mathcal{M} \subseteq \mathbb{N}^{n\times n}$ be a finite set of nonnegative integer matrices, generating a finite monoid $\overline{\mathcal{M}}$. For notational convenience, throughout the paper, we associate to $\mathcal{M}$ a bijection $M : \Sigma \to \mathcal{M}$ and extend it to the monoid morphism $M : \Sigma^* \to \overline{\mathcal{M}}$. Thus we may write $M(\Sigma^*)$ for $\overline{\mathcal{M}}$.

Towards a proof of Proposition 2, define the rational nonnegative matrix $A \in \mathbb{Q}^{n\times n}$ by $A := \frac{1}{|\Sigma|}\sum_{a\in\Sigma} M(a)$. Observe that for $k \in \mathbb{N}$ we have $A^k = \frac{1}{|\Sigma^k|}\sum_{w\in\Sigma^k} M(w)$, i.e., A^k is the average of the $M(w)$, where w ranges over all words of length k. Define $\rho \geq 0$ as the spectral radius of A.

▶ **Lemma 5.** *We have $\rho \leq 1$.*

Proof. Since $M(\Sigma^*)$ is finite, it is bounded. Hence $(A^k)_{k\in\mathbb{N}}$ is bounded. By the Perron-Frobenius theorem, A has a nonnegative left eigenvector $u \in \mathbb{R}^n$ with $uA = \rho u$. So $uA^k = \rho^k u$. It follows $\rho \leq 1$. ◀

▶ **Lemma 6.** *We have $\rho < 1$ if and only if there is $w \in \Sigma^*$ with $M(w) = 0$.*

Proof. Suppose $\rho < 1$. Then $\lim_{k\to\infty} A^k = 0$, and so there is $k \in \mathbb{N}$ such that the sum of all entries of A^k is less than 1. It follows that there is $w \in \Sigma^k$ such that the sum of all entries of $M(w)$ is less than 1. Since $M(w) \in \mathbb{N}^{n\times n}$ it follows $M(w) = 0$.

Conversely, suppose there is $w_0 \in \Sigma^*$ with $M(w_0) = 0$. Since $M(\Sigma^*)$ is finite, there is $B \in \mathbb{N}$ such that all entries of all matrices in $M(\Sigma^*)$ are at most B. For any $k \in \mathbb{N}$ define $W(k) := \Sigma^k \setminus (\Sigma^* w_0 \Sigma^*)$, i.e., $W(k)$ is the set of length-k words that do not contain w_0 as a factor. Note that $M(w) = 0$ holds for all $w \in \Sigma^k \setminus W(k)$. It follows that any entry of A^k is at most $\frac{|W(k)|}{|\Sigma^k|} \cdot B$. On the other hand, for any $m \in \mathbb{N}$, if a word of length $m|w_0|$ is picked uniformly at random, then the probability of picking a word in $W(m|w_0|)$ is at most

$$\left(1 - \frac{1}{|\Sigma^{|w_0|}|}\right)^m.$$

It follows that $\lim_{k\to\infty} \frac{|W(k)|}{|\Sigma^k|} = 0$. Hence $\lim_{k\to\infty} A^k = 0$ and so $\rho < 1$. ◀

With these lemmas at hand, we can prove Proposition 2:

▶ **Proposition 2.** *Let $\mathcal{M} \subseteq \mathbb{N}^{n\times n}$ be a finite set of nonnegative integer matrices, generating a finite monoid $\overline{\mathcal{M}}$. One can decide in polynomial time if $0 \in \overline{\mathcal{M}}$.*

Proof. By Lemma 6, it suffices to check whether $\rho < 1$.

If $\rho < 1$ then the linear system $xA = x$ does not have a nonzero solution. Conversely, if $\rho \geq 1$ then, by Lemma 5, we have $\rho = 1$ and thus, by the Perron-Frobenius theorem, the linear system $xA = x$ has a (real) nonzero solution.

Hence it suffices to check if $xA = x$ has a nonzero solution. This can be done in polynomial time. ◀

As remarked in section 1, this algorithm does not exhibit a word w with $M(w) = 0$, even when it proves the existence of such w.

3 Proof of Theorem 1

As before, let $M : \Sigma^* \to \mathbb{N}^{n\times n}$ be a monoid morphism with finite image $M(\Sigma^*)$. Call M *strongly connected* if for all $i, j \in \{1, \dots, n\}$ there is $w \in \Sigma^*$ with $M(w)(i, j) \geq 1$. In subsection 3.1 we consider the case where M is strongly connected. In subsection 3.2 we consider the general case.

3.1 Strongly Connected

In this section we consider the case that M is strongly connected and prove the following proposition, which extends Carpi's Theorem 3:

▶ **Proposition 7.** *Let $M : \Sigma^* \to \mathbb{N}^{n\times n}$ be strongly connected with finite $M(\Sigma^*)$. Given $M : \Sigma \to \mathbb{N}^{n\times n}$, one can compute in polynomial time a word $w \in \Sigma^*$ with $|w| \leq \frac{1}{16}n^5 + \frac{15}{16}n^4$ such that $M(w)$ has minimum rank in $M(\Sigma^*)$.*

In the strongly connected case, $M(\Sigma^*)$ does not have numbers larger than 1:

▶ **Lemma 8.** *If M is strongly connected, then $M(\Sigma^*) \subseteq \{0, 1\}^{n\times n}$.*

Proof. Let M be strongly connected. Suppose $M(v)(i, j) \geq 2$ for some $v \in \Sigma^*$. Since M is strongly connected, there is $w \in \Sigma^*$ with $M(w)(j, i) \geq 1$. Hence $M(vw)(i, i) \geq 2$. It follows that $M((vw)^k)(i, i) \geq 2^k$ for all $k \in \mathbb{N}$, contradicting the finiteness of $M(\Sigma^*)$. ◀

Lemma 8 allows us to view the strongly connected case in terms of UFAs. Define a UFA $\mathcal{A} = (\Sigma, Q, \delta)$ with $Q = \{1, \dots, n\}$ and $\delta(p, a) \ni q$ if and only if $M(a)(p, q) = 1$. For the rest of the subsection we will mostly consider Q as an arbitrary finite set of n states. We extend $\delta : Q \times \Sigma \to 2^Q$ in the usual way to $\delta : 2^Q \times \Sigma^* \to 2^Q$ by setting $\delta(P, a) := \bigcup_{q\in P} \delta(q, a)$ and $\delta(P, \varepsilon) := P$ and $\delta(P, wa) := \delta(\delta(P, w), a)$, where $P \subseteq Q$ and $a \in \Sigma$ and ε is the empty word and $w \in \Sigma^*$. When there is no confusion, we may write pw for $\delta(p, w)$ and wq for $\{p \in Q : pw \ni q\}$. We extend this to $Pw := \bigcup_{p\in P} pw$ and $wP := \bigcup_{p\in P} wp$. We say a state p is *reached by* a word w when $wp \neq \emptyset$, and a state p *survives* a word w when $pw \neq \emptyset$. Note that Qw is the set of states that are reached by w, and wQ is the set of states that survive w. Let $q_1 \neq q_2$ be two different states. Then q_1, q_2 are called *coreachable* when there is $w \in \Sigma^*$ with $wq_1 \cap wq_2 \neq \emptyset$ (i.e., there is $p \in Q$ with $pw \supseteq \{q_1, q_2\}$), and they are called *mergeable* when there is $w \in \Sigma^*$ with $q_1w \cap q_2w \neq \emptyset$. For any $q \in Q$ we define $C(q)$ as the set of states coreachable with q. Also, define $c := \max\{|qw| : q \in Q,\ w \in \Sigma^*\}$ and $m := \max\{|wq| : w \in \Sigma^*,\ q \in Q\}$. The following lemma says that one can compute short witnesses for coreachability:

▶ **Lemma 9.** *If states $q \neq q'$ are coreachable, then one can compute in polynomial time $w_{q,q'} \in \Sigma^*$ with $|w_{q,q'}| \leq \frac{1}{2}(n + 2)(n - 1)$ such that $qw_{q,q'} \supseteq \{q, q'\}$.*

Proof. Let $q \neq q'$ be coreachable states. Then there are $p \in Q$ and $v \in \Sigma^*$ with $pv \supseteq \{q, q'\}$. Since M is strongly connected, there is $u \in \Sigma^*$ with $qu \ni p$, hence $quv \supseteq \{q, q'\}$. Define an edge-labelled directed graph $G = (V, E)$ with vertex set $V = \{\{r, s\} : r, s \in Q\}$ and edge set

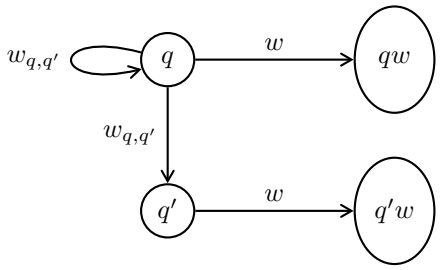
$E = \{(R, a, S) \in V \times \Sigma \times V : Ra \supseteq S\}$. Since $quv \supseteq \{q, q'\}$, the graph G has a path, labelled with uv, from $\{q\}$ to $\{q, q'\}$. The shortest path from $\{q\}$ to $\{q, q'\}$ has at most $|V| - 1$ edges and is thus labelled with a word $w \in \Sigma^*$ with $|w| \leq |V| - 1 = \frac{1}{2}n(n+1) - 1 = \frac{1}{2}(n+2)(n-1)$. For this w we have $qw \supseteq \{q, q'\}$. ◀

▶ **Lemma 10.** *For each $q \in Q$ one can compute in polynomial time a word $w_q \in \Sigma^*$ with $|w_q| \leq \frac{1}{2}(c-1)(n+2)(n-1)$ such that no state $q' \neq q$ survives w_q and is coreachable with q.*

Proof. Let $q \in Q$. Consider the following algorithm:

1. $w := \varepsilon$ (the empty word)
2. while there is $q' \in C(q)$ such that q' survives w:
$w := w_{q,q'}w$ (with $w_{q,q'}$ from Lemma 9)
3. return $w_q := w$

The following picture visualizes aspects of this algorithm:



We argue that the computed word w_q has the required properties. First we show that the set qw increases in each iteration of the algorithm. Indeed, let w and $w_{q,q'}w$ be the words computed by two subsequent iterations. Since $qw_{q,q'} \supseteq \{q, q'\}$, we have $qw_{q,q'}w \supseteq qw \cup q'w$. The set $q'w$ is nonempty, as q' survives w. As can be read off from the picture above, the sets qw and $q'w$ are disjoint, as otherwise there would be two distinct paths from q to a state in $qw \cap q'w$, both labelled with $w_{q,q'}w$, contradicting unambiguousness. It follows that $qw_{q,q'}w \supsetneq qw$. Hence the algorithm must terminate.

Since in each iteration the set qw increases by at least one element (starting from $\{q\}$), there are at most $c - 1$ iterations. Hence $|w_q| \leq \frac{1}{2}(c-1)(n+2)(n-1)$. There is no state $q' \neq q$ that survives w_q and is coreachable with q, as otherwise the algorithm would not have terminated. ◀

▶ **Lemma 11.** *One can compute in polynomial time words $z, y \in \Sigma^*$ such that:*

- *$|z| \leq \frac{1}{4}(c-1)(n+2)n(n-1)$ and there are no two coreachable states that both survive z;*
- *$|y| \leq \frac{1}{4}(m-1)(n+2)n(n-1)$ and there are no two mergeable states that are both reached by y.*

Proof. As the two statements are dual, we prove only the first one. Consider the following algorithm:

1. $w := \varepsilon$ (the empty word)
2. while there are coreachable p, p' that both survive w:
$q :=$ arbitrary state from pw
$w := ww_q$ (with w_q from Lemma 10)
3. return $z := w$

We show that the set

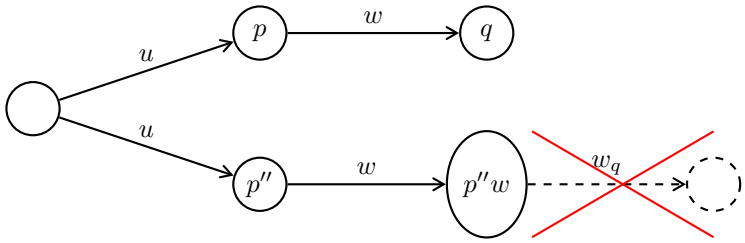
$$B := \{p \in Q : \exists\, p'' \in C(p) \text{ such that both } p, p'' \text{ survive } w\}$$

loses at least two states in each iteration. First observe that

$$B' := \{p \in Q : \exists p'' \in C(p) \text{ such that both } p, p'' \text{ survive } ww_q\}$$

is clearly a subset of B.

Let $p \in B$ be the state from line 2 of the algorithm, and let $q \in pw$ be the state from the body of the loop. We claim that no $p'' \in C(p)$ survives ww_q. Indeed, let $p'' \in C(p)$. The following picture visualizes the situation:



By unambiguousness and since $q \in pw$, we have $q \notin p''w$. By the definition of w_q and since all states in $p''w$ are coreachable with q, we have $p''ww_q = \emptyset$, which proves the claim.

By the claim, we have $p \notin B'$. Let $p' \in B$ be the state p' from line 2 of the algorithm. We have $p' \in C(p)$. By the claim, p' does not survive ww_q. Hence $p' \notin B'$.

So we have shown that the algorithm removes at least two states from B in every iteration. Thus it terminates after at most $\frac{n}{2}$ iterations. Using the length bound from Lemma 10 we get $|z| \leq \frac{1}{4}(c-1)(n+2)n(n-1)$. There are no coreachable q, q' that both survive z, as otherwise the algorithm would not have terminated. ◀

For the following development, let $q_1, \ldots, q_k$ be the states that are reached by y and survive z (with y, z from Lemma 11), see Figure 2.

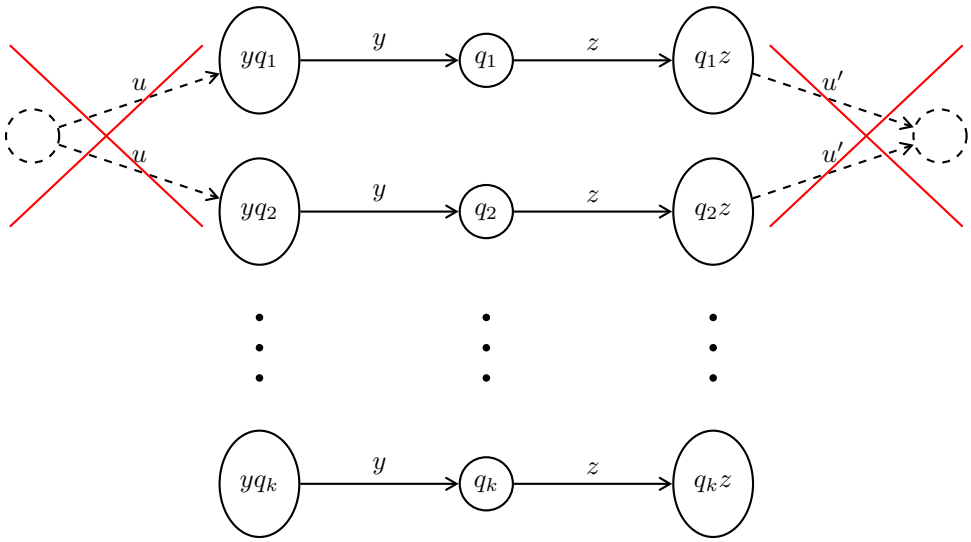


Figure 2 The states $q_1, \ldots, q_k$ are neither coreachable nor mergeable.

▶ **Lemma 12.** *Let $1 \leq i < j \leq k$. Then q_i, q_j are neither coreachable nor mergeable.*

Proof. Immediate from the properties of y, z (Lemma 11). ◀

The following lemma restricts sets of the form $q_i zxyz$ for $i \in \{1,\dots,k\}$ and $x \in \Sigma^*$:

▶ **Lemma 13.** *Let $i \in \{1,\dots,k\}$ and $x \in \Sigma^*$. Then there is $j \in \{1,\dots,k\}$ such that $q_i zxyz \subseteq q_j z$.*

Proof. If $q_i zxyz = \emptyset$ then choose j arbitrarily. Otherwise, let $q \in q_i zxyz$. Then q is reached by yz, so there is j with $q_i zxy \ni q_j$ and $q_j z \ni q$. We show that $q_i zxyz \subseteq q_j z$. To this end, let $q' \in q_i zxyz$. Then q' is reached by yz, so there is j' with $q_i zxy \ni q_{j'}$ and $q_{j'} z \ni q'$. Since $q_i zxy \supseteq \{q_j, q_{j'}\}$ and $q_j, q_{j'}$ are not coreachable (by Lemma 12), we have $j' = j$. Hence $q_j z = q_{j'} z \ni q'$. ◀

Provided that there is a killing word (which can be checked via Proposition 2 in polynomial time), the following lemma asserts that for each $i \in \{1,\dots,k\}$ one can efficiently compute a short word x_i such that no state in $q_i z$ survives $x_i yz$. The proof hinges on a linear-algebra based technique for checking equivalence of automata that are weighted over a field. This technique goes back to Schützenberger [18] and has often been rediscovered, see, e.g., [19].

▶ **Lemma 14.** *Suppose there exists $w_0 \in \Sigma^*$ with $M(w_0) = 0$ (this word w_0 may not be given). For each $i \in \{1,\dots,k\}$ one can compute in polynomial time a word $x_i \in \Sigma^*$ with $|x_i| \le n$ such that $q_i z x_i yz = \emptyset$.*

Proof. Let $i \in \{1,\dots,k\}$. Since $y\{q_1,\dots,q_k\}$ are the only states to survive yz, it suffices to compute $x \in \Sigma^*$ with $|x| \le n$ such that $q_i zx \cap y\{q_1,\dots,q_k\} = \emptyset$.

Define $e \in \{0,1\}^Q$ as the row vector with $e(q) = 1$ if and only if $q \in q_i z$. Define $f \in \{0,1\}^Q$ as the row vector with $f(q) = 1$ if and only if $q \in y\{q_1,\dots,q_k\}$. First we show that for any $x \in \Sigma^*$ we have $eM(x)f^\top \le 1$, where the superscript $\top$ denotes transpose. Towards a contradiction suppose $eM(x)f^\top \ge 2$. Then there are two distinct x-labelled paths from $q_i z$ to $y\{q_1,\dots,q_k\}$. It follows that there are two distinct zxy-labelled paths from q_i to $\{q_1,\dots,q_k\}$. By unambiguousness, these paths end in two distinct states $q_j, q_{j'}$. But then $q_j, q_{j'}$ are coreachable, contradicting Lemma 12. Hence we have shown that $eM(x)f^\top \le 1$ holds for all $x \in \Sigma^*$.

Define the (row) vector space

$$V \;:=\; \big\langle \begin{pmatrix} eM(x) & 1 \end{pmatrix} : x \in \Sigma^* \big\rangle \;\subseteq\; \mathbb{R}^{n+1}\,,$$

i.e., V is spanned by the vectors $\begin{pmatrix} eM(x) & 1 \end{pmatrix}$ for $x \in \Sigma^*$. The vector space V can be equivalently characterized as the smallest vector space that contains $\begin{pmatrix} e & 1 \end{pmatrix}$ and is closed under multiplication with $\begin{pmatrix} M(a) & 0 \\ 0 & 1 \end{pmatrix}$ for all $a \in \Sigma$. Hence the following algorithm computes a set $B \subseteq \Sigma^*$ such that $\big\{\begin{pmatrix} eM(x) & 1 \end{pmatrix} : x \in B\big\}$ is a basis of V:

1. $B := \{\varepsilon\}$ (where ε is the empty word)
2. while there are $u \in B$ and $a \in \Sigma$ such that $\begin{pmatrix} eM(ua) & 1 \end{pmatrix} \notin \big\langle \begin{pmatrix} eM(x) & 1 \end{pmatrix} : x \in B \big\rangle$:
   $B := B \cup \{ua\}$
3. return B

Observe that the algorithm performs at most n iterations of the loop body, as every iteration increases the dimension of the space $\big\langle \begin{pmatrix} eM(x) & 1 \end{pmatrix} : x \in B \big\rangle$ by 1, but the dimension cannot grow larger than $n+1$. Hence $|x| \le n$ holds for all $x \in B$. Since $eM(w_0)f^\top = 0 \ne 1$, the space V is not orthogonal to $\begin{pmatrix} f & -1 \end{pmatrix}$. So there exists $x \in B$ such that $eM(x)f^\top \ne 1$. Since $eM(x)f^\top \le 1$, we have $eM(x)f^\top = 0$. Hence $q_i zx \cap y\{q_1,\dots,q_k\} = \emptyset$. ◀

Now we can prove the following lemma, which is our main technical contribution:

▶ **Lemma 15.** *Suppose there is $w_0 \in \Sigma^*$ with $M(w_0) = 0$ (this word w_0 may not be given). One can compute in polynomial time a word $w \in \Sigma^*$ with $M(w) = 0$ and $|w| \leq \frac{1}{16}n^5 + \frac{15}{16}n^4$.*

Proof. For any $1 \leq j < j' \leq k$ the sets $q_j z$ and $q_{j'} z$ are disjoint by Lemma 12 and nonempty. Hence any $P' \subseteq Q$ has at most one set $P \subseteq \{q_1, \dots, q_k\}$ with $Pz = P'$, which we call the *generator* of P'. Note that all sets of the form $Q'yz$ where $Q' \subseteq Q$ have a generator. For any $i \in \{1, \dots, k\}$, let x_i be the word from Lemma 14, i.e., $q_i z x_i y z = \emptyset$. By Lemma 13, for any $j \in \{1, \dots, k\}$ the generator of $q_j z x_i y z$ has at most one element. Thus, if $q_i \in P \subseteq \{q_1, \dots, q_k\}$, then the generator, P, of Pz has strictly more elements than the generator of Pzx_iyz.

Consider the following algorithm:

1. $w := yz$
2. while $Qw \neq \emptyset$:
   $q_i :=$ arbitrary element of the generator of Qw
   $w := wx_iyz$
3. return w

It follows from the argument above that the size of the generator of Qw decreases in every iteration of the loop. Hence the algorithm terminates after at most k iterations and computes a word w such that $Qw = \emptyset$ and, using Lemmas 11 and 14,

$$|w| \;\leq\; |yz| + k(n + |yz|) \;\leq\; n^2 + (k+1)(|y| + |z|) \;\leq\; n^2 + \frac{1}{4}(k+1)(c+m-2)(n+2)n(n-1)\,.$$

Let $q, q' \in Q$ and $u, u' \in \Sigma^*$ such that $c = |qu|$ and $m = |u'q'|$. Clearly, $qu \cup u'q' \cup \{q_1, \dots, q_k\} \subseteq Q$, and it follows from the inclusion-exclusion principle:

$$c + m + k \;\leq\; n + |qu \cap u'q'| + |qu \cap \{q_1, \dots, q_k\}| + |\{q_1, \dots, q_k\} \cap u'q'|$$

The sets qu and $u'q'$ overlap in at most one state by unambiguousness. The sets qu and $\{q_1, \dots, q_k\}$ overlap in at most one state by Lemma 12, and similarly for $\{q_1, \dots, q_k\}$ and $u'q'$. It follows $c+m+k \leq n+3$, thus $(k+1)+(c+m-2) \leq n+2$, hence $(k+1)(c+m-2) \leq \frac{1}{4}(n+2)^2$. With the bound on the length of w above we conclude that $|w| \leq n^2 + \frac{1}{16}(n+2)^3 n(n-1)$, which is bounded by $\frac{1}{16}n^5 + \frac{15}{16}n^4$ for $n \geq 1$. ◀

We combine Lemma 15 and Carpi's Theorem 3 to prove Proposition 7:

▶ **Proposition 7.** *Let $M : \Sigma^* \to \mathbb{N}^{n \times n}$ be strongly connected with finite $M(\Sigma^*)$. Given $M : \Sigma \to \mathbb{N}^{n \times n}$, one can compute in polynomial time a word $w \in \Sigma^*$ with $|w| \leq \frac{1}{16}n^5 + \frac{15}{16}n^4$ such that $M(w)$ has minimum rank in $M(\Sigma^*)$.*

Proof. One can check in polynomial time whether there is $w_0 \in \Sigma^*$ with $M(w_0) = 0$, see Proposition 2. If yes, then the minimum rank is 0, and Lemma 15 gives the result.

Otherwise, the minimum rank r is between 1 and n, and hence $n \geq 1$. Theorem 3 asserts the existence of a word w such that $M(w)$ has rank r and $|w| \leq \frac{1}{2}n^4 - n^3 + \frac{5}{2}n^2 - 3n + 1$, which is bounded by $\frac{1}{16}n^5 + \frac{15}{16}n^4$ for $n \geq 1$. An inspection of Carpi's proof [4] shows that his proof is constructive and can be transformed into an algorithm that computes w in polynomial time. ◀

3.2 Not Necessarily Strongly Connected

We prove Theorem 1:

▶ **Theorem 1.** *Let $n \in \mathbb{N}$ and $\mathcal{M} \subseteq \mathbb{N}^{n\times n}$ be a finite set of nonnegative integer matrices. Denote by $\overline{\mathcal{M}}$ the monoid generated by $\mathcal{M}$ under matrix multiplication. If $\overline{\mathcal{M}}$ is finite then there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq \frac{1}{16}n^5 + \frac{15}{16}n^4$ such that the matrix product $M_1 \cdots M_\ell$ has minimum rank in $\overline{\mathcal{M}}$. Further, $M_1, \ldots, M_\ell$ can be computed in time polynomial in the description size of $\mathcal{M}$.*

In terms of the previous notions in the proof we can rephrase Theorem 1 as follows:

▶ **Theorem 1** (rephrased). *Let $M : \Sigma^* \to \mathbb{N}^{n\times n}$ be a monoid morphism whose image $M(\Sigma^*)$ is finite. Given $M : \Sigma \to \mathbb{N}^{n\times n}$, one can compute in polynomial time a word $w \in \Sigma^*$ with $|w| \leq \frac{1}{16}n^5 + \frac{15}{16}n^4$ such that $M(w)$ has minimum rank in $M(\Sigma^*)$.*

Proof. For any matrix A denote by $rk(A)$ its rank. For $i, j \in \{1, \ldots, n\}$ write $i \to j$ if there is $u \in \Sigma^*$ such that $M(u)(i, j) > 0$, and write $i \leftrightarrow j$ if $i \to j$ and $j \to i$. The relation $\leftrightarrow$ is an equivalence relation. Denote by $C_1, \ldots, C_h \subseteq \{1, \ldots, n\}$ its equivalence classes ($h \leq n$). We can assume that whenever $i \in C_k$ and $j \in C_\ell$ and $i \to j$, then $k \leq \ell$. Hence, without loss of generality, $M(u)$ for any $u \in \Sigma^*$ has the following block-upper triangular form:

$$M(u) \;=\; \begin{pmatrix} M_{11}(u) & M_{12}(u) & \cdots & M_{1h}(u) \\ 0 & M_{22}(u) & \cdots & M_{2h}(u) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & M_{hh}(u) \end{pmatrix},$$

where $M_{ii}(u) \in \mathbb{N}^{|C_i|\times|C_i|}$ for all $i \in \{1, \ldots, h\}$. For $i \in \{1, \ldots, h\}$ define $r_i := \min_{u\in\Sigma^*} rk(M_{ii}(u))$. For any $u \in \Sigma^*$ we have $rk(M(u)) \geq \sum_{i=1}^{h} rk(M_{ii}(u))$ (see, e.g., [9, Chapter 0.9.4]). It follows that the minimum rank among the matrices in $M(\Sigma^*)$ is at least $\sum_{i=1}^{h} r_i$.

Let $w_1, \ldots, w_h \in \Sigma^*$ be the words from Proposition 7 for $M_{11}, \ldots, M_{hh}$, respectively, so that $rk(M_{ii}(w_i)) = r_i$ holds for all $i \in \{1, \ldots, h\}$. Define $w := w_1 \cdots w_h$. Then we have:

$$|w| \;\leq\; \sum_{i=1}^{h} |w_i| \;\leq\; \sum_{i=1}^{h} \frac{1}{16}|C_i|^5 + \frac{15}{16}|C_i|^4 \;\leq\; \frac{1}{16}n^5 + \frac{15}{16}n^4$$

It remains to show that $rk(M(w)) \leq \sum_{i=1}^{h} r_i$. It suffices to prove that $rk(M_k(w_1 \cdots w_k)) \leq \sum_{i=1}^{k} r_i$ holds for all $k \in \{1, \ldots, h\}$, where $M_k(u)$ for any $u \in \Sigma^*$ is the principal submatrix obtained by restricting $M(u)$ to the rows and columns corresponding to $\bigcup_{i=1}^{k} C_i$. We proceed by induction on k. For the base case, $k = 1$, we have $rk(M_1(w_1)) = rk(M_{11}(w_1)) = r_1$. For the induction step, let $1 < k \leq h$. Then there are matrices A_1, A_2, B_1, B_2 such that:

$$\begin{aligned} M_k(w_1 \cdots w_k) \;&=\; M_k(w_1 \cdots w_{k-1}) M_k(w_k) \\ &=\; \begin{pmatrix} M_{k-1}(w_1 \cdots w_{k-1}) & A_1 \\ 0 & A_2 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \\ 0 & M_{kk}(w_k) \end{pmatrix} \\ &=\; \begin{pmatrix} M_{k-1}(w_1 \cdots w_{k-1}) \\ 0 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \end{pmatrix} + \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \begin{pmatrix} 0 & M_{kk}(w_k) \end{pmatrix} \end{aligned} \tag{1}$$

By the induction hypothesis, we have $rk(M_{k-1}(w_1 \cdots w_{k-1})) \leq \sum_{i=1}^{k-1} r_i$. Further, we have $rk(M_{kk}(w_k)) = r_k$. So the ranks of the two summands in (1) are at most $\sum_{i=1}^{k-1} r_i$ and r_k, respectively. Since for any matrices A, B it holds $rk(A + B) \leq rk(A) + rk(B)$, we conclude that $rk(M_k(w_1 \cdots w_k)) \leq \sum_{i=1}^{k} r_i$, completing the induction proof. ◀

4 Proof of Theorem 4

In terms of the previous notions we can rephrase Theorem 4 as follows:

▶ **Theorem 4** (rephrased). *There is no polynomial p such that the following holds:*

Let $M : \Sigma^* \to \{0,1\}^{Q\times Q}$ be a monoid morphism. Let $w_0 \in \Sigma^*$ be such that $M(w_0)$ has rank 1, and let 1 be the minimum rank in $M(\Sigma^*)$. Then there is $w \in \Sigma^*$ with $|w| \leq p(|Q|)$ such that $M(w_0) = M(w)$.

Proof. Denote by p_i the ith prime number (so $p_1 = 2$). Let $m \geq 1$. Define:

$$\begin{aligned}\Sigma &:= \{a, b_1, \dots, b_m\}\\ Q_i &:= \{(i,0),(i,1),\dots,(i,p_i-1)\} \quad \text{for every } i \in \{1,\dots,m\}\\ Q &:= \{0\} \cup \bigcup_{i=1}^{m} Q_i\end{aligned}$$

Further, define a monoid morphism $M : \Sigma^* \to \mathbb{N}^{Q\times Q}$ by setting for all $i \in \{1,\dots,m\}$

$$\begin{aligned}M(a)(0,(i,0)) &:= 1\\ M(a)((i,j),(i,j+1 \bmod p_i)) &:= 1 \quad \text{for all } j \in \{0,\dots,p_i-1\}\\ M(b_i)(0,0) &:= 1\\ M(b_i)((i,j),0) &:= 1 \quad \text{for all } j \in \{0,\dots,p_i-1\}\end{aligned}$$

and setting all other entries of $M(a), M(b_1), \dots, M(b_m)$ to 0, see Figure 3. We have $M(\Sigma^*) \subseteq \{0,1\}^{Q\times Q}$, i.e., $M(\Sigma^*)$ is an unambiguous monoid of relations. For all $q \in Q$ and all $q' \in Q \setminus \{0\}$ we have $M(b_1)(q,q') = 0$, i.e., $M(b_1)$ has rank 1. For all $w \in \Sigma^*$ there is $q \in Q$ with $M(w)(0,q) = 1$, i.e., 1 is the minimum rank in $M(\Sigma^*)$. A shortest word $w_0 \in \Sigma^*$ such that $M(w_0)$ has rank 1 and $M(w_0)(0,(i,p_i-1)) = 1$ holds for all $i \in \{1,\dots,m\}$ is the word $w_0 = b_1 a^P$ where $P = \prod_{i=1}^{m} p_i \geq 2^m$. On the other hand, we have $|Q| = 1 + \sum_{i=1}^{m} p_i \in O(m^2 \log m)$ by the prime number theorem.

Hence there is no polynomial p such that $P \leq p(|Q|)$ holds for all m. ◀

References

1 P.C. Bell, M. Hirvensalo, and I. Potapov. Mortality for 2 ×2 Matrices Is NP-Hard. In *Proceedings of Mathematical Foundations of Computer Science (MFCS)*, pages 148–159. Springer, 2012.

2 J. Berstel and D. Perrin. Trends in the theory of codes. *Bulletin of the EATCS*, 29:84–95, 1986.

3 J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata*. Cambridge University Press, 2009.

4 A. Carpi. On synchronizing unambiguous automata. *Theoretical Computer Science*, 60(3):285–296, 1988.

5 A. Carpi and F. D'Alessandro. On incomplete and synchronizing finite sets. *Theoretical Computer Science*, 664:67–77, 2017.

6 P. Goralčík, Z. Hedrlín, V. Koubek, and J. Ryšlinková. A game of composing binary relations. *R.A.I.R.O. Informatique théorique*, 16(4):365–369, 1982.

7 V.V. Gusev and E.V. Pribavkina. On Non-complete Sets and Restivo's Conjecture. In *Proceedings of Developments in Language Theory (DLT)*, pages 239–250. Springer, 2011.

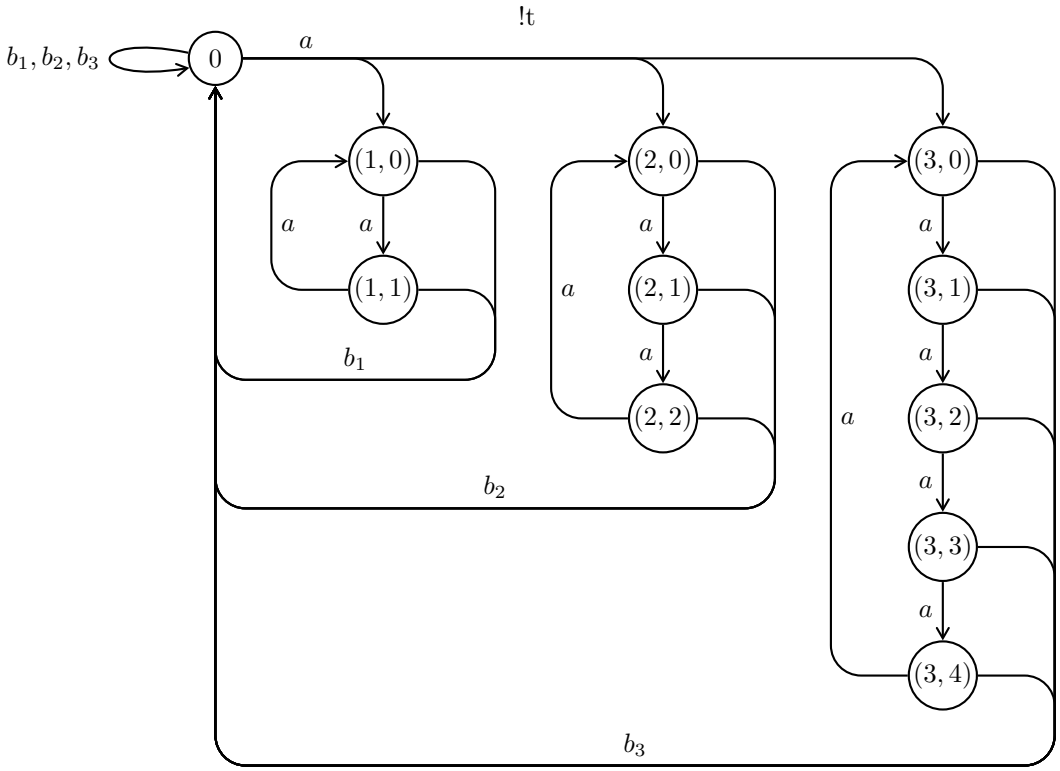


Figure 3 Automaton representation of M for $m = 3$.

8 V. Halava, T. Harju, and M. Hirvensalo. Undecidability Bounds for Integer Matrices Using Claus Instances. *International Journal of Foundations of Computer Science*, 18(5):931–948, 2007.

9 R.A. Horn and C.R. Johnson. *Matrix analysis*. Cambridge University Press, 2nd edition, 2013.

10 S. Julia, A. Malapert, and J. Provillard. A Synergic Approach to the Minimal Uncompletable Words Problem. *Journal of Automata, Languages and Combinatorics*, 22(4):271–286, 2017.

11 R.M. Jungers, V. Protasov, and V.D. Blondel. Efficient algorithms for deciding the type of growth of products of integer matrices. *Linear Algebra and its Applications*, 428(10):2296–2311, 2008.

12 J.-Y. Kao, N. Rampersad, and J. Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47):5010–5021, 2009.

13 P.V. Martugin. A series of slowly synchronizing automata with a zero state over a small alphabet. *Information and Computation*, 206(9-10):1197–1203, 2008.

14 M.S. Paterson. Unsolvability in 3×3 Matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

15 I. Potapov and P. Semukhin. Decidability of the Membership Problem for 2×2 integer matrices. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 170–186. SIAM, 2017.

16 A. Restivo. Some remarks on complete subsets of a free monoid. In *Quaderni de "La Ricerca Scientifica". Non-Commutative Structures in Algebra and Geometric Combinatorics*, volume 109, pages 19–25. Consiglio Nazionale Delle Ricerche, 1981.

17 A. Ryzhikov and M. Szykuła. Finding Short Synchronizing Words for Prefix Codes. In *Proceedings of Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *LIPIcs*, pages 21:1–21:14, 2018.

18 M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

19 W. Tzeng. A Polynomial-Time Algorithm for the Equivalence of Probabilistic Automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.

20 M.V. Volkov. Synchronizing Automata and the Černý Conjecture. In *Proceedings of Language and Automata Theory and Applications (LATA)*, pages 11–27. Springer, 2008.

21 A. Weber and H. Seidl. On finitely generated monoids of matrices with entries in ℕ. *Informatique Théorique et Applications*, 25(1):19–38, 1991.

Tight Complexity Lower Bounds for Integer Linear Programming with Few Constraints

Dušan Knop
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic
knop.dusan@fit.cvut.cz

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Marcin Wrochna
Institute of Informatics, University of Warsaw, Poland
University of Oxford, UK
m.wrochna@mimuw.edu.pl

Abstract

We consider the standard ILP Feasibility problem: given an integer linear program of the form $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$, where A is an integer matrix with k rows and ℓ columns, $\mathbf{x}$ is a vector of ℓ variables, and $\mathbf{b}$ is a vector of k integers, we ask whether there exists $\mathbf{x} \in \mathbb{N}^\ell$ that satisfies $A\mathbf{x} = \mathbf{b}$. Each row of A specifies one linear *constraint* on $\mathbf{x}$; our goal is to study the complexity of ILP Feasibility when both k, the number of constraints, and $\|A\|_\infty$, the largest absolute value of an entry in A, are small.

Papadimitriou [29] was the first to give a fixed-parameter algorithm for ILP Feasibility under parameterization by the number of constraints that runs in time $((\|A\|_\infty + \|\mathbf{b}\|_\infty) \cdot k)^{\mathcal{O}(k^2)}$. This was very recently improved by Eisenbrand and Weismantel [9], who used the Steinitz lemma to design an algorithm with running time $(k\|A\|_\infty)^{\mathcal{O}(k)} \cdot \|\mathbf{b}\|_\infty^2$, which was subsequently improved by Jansen and Rohwedder [17] to $\mathcal{O}(k\|A\|_\infty)^k \cdot \log \|\mathbf{b}\|_\infty$. We prove that for $\{0,1\}$-matrices A, the running time of the algorithm of Eisenbrand and Weismantel is probably optimal: an algorithm with running time $2^{o(k \log k)} \cdot (\ell + \|\mathbf{b}\|_\infty)^{o(k)}$ would contradict the Exponential Time Hypothesis (ETH). This improves previous non-tight lower bounds of Fomin et al. [10].

We then consider integer linear programs that may have many constraints, but they need to be structured in a "shallow" way. Precisely, we consider the parameter *dual treedepth* of the matrix A, denoted $\mathrm{td}_D(A)$, which is the treedepth of the graph over the rows of A, where two rows are adjacent if in some column they simultaneously contain a non-zero entry. It was recently shown by Koutecký et al. [24] that ILP Feasibility can be solved in time $\|A\|_\infty^{2^{\mathcal{O}(\mathrm{td}_D(A))}} \cdot (k + \ell + \log \|\mathbf{b}\|_\infty)^{\mathcal{O}(1)}$. We present a streamlined proof of this fact and prove that, again, this running time is probably optimal: even assuming that all entries of A and $\mathbf{b}$ are in $\{-1, 0, 1\}$, the existence of an algorithm with running time $2^{2^{o(\mathrm{td}_D(A))}} \cdot (k + \ell)^{\mathcal{O}(1)}$ would contradict the ETH.

2012 ACM Subject Classification Theory of computation → Integer programming; Theory of computation → Fixed parameter tractability

Keywords and phrases integer linear programming, fixed-parameter tractability, ETH

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.44

Related Version The full version of this paper is available as an arXiv preprint [23], `http://arxiv.org/abs/1811.01296`.

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 44; pp. 44:1–44:15

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

Funding This work is a part of projects CUTACOMBS, PowAlgDO (M. Wrochna) and TOTAL (M. Pilipczuk) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements No. 714704, No 714532, and No. 677651). Dušan Knop is supported by DFG, project "MaMu", NI 369/19. Marcin Wrochna is supported by Foundation for Polish Science (FNP) via the START stipend.

1 Introduction

Integer linear programming (ILP) is a powerful technique used in countless algorithmic results of theoretical importance, as well as applied routinely in thousands of instances of practical computational problems every day. Despite the problem being NP-hard in general, practical ILP solvers excel in solving real-life instances with thousands of variables and constraints. This can be partly explained by applying a variety of subroutines, often based on heuristic approaches, that identify and exploit structure in the input in order to apply the best suited algorithmic strategies. A theoretical explanation of this phenomenon would of course be hard to formulate, but one approach is to use the paradigm of *parameterized complexity*. Namely, the idea is to design algorithms that perform efficiently when certain relevant structural parameters of the input have moderate values.

In this direction, probably the most significant is the classic result of Lenstra [25], who proved that ILP Optimization is *fixed-parameter tractable* when parameterized by the number of variables ℓ. That is, it can be solved in time $f(\ell) \cdot |I|^{\mathcal{O}(1)}$, where f is some function and $|I|$ is the total bitsize of the input; we shall use the previous notation throughout the whole manuscript. Subsequent work in this direction [11, 18] improved the dependence of the running time on ℓ to $f(\ell) \leqslant 2^{\mathcal{O}(\ell \log \ell)}$.

In this work we turn to a different structural aspect and study ILPs that have few *constraints*, as opposed to few *variables* as in the setting considered by Lenstra. Formally, we consider the parameterization by k, the number of *constraints* (rows of the input matrix A), and $\|A\|_\infty$, the maximum absolute value over all entries in A. The situation when the number of constraints is significantly smaller than the number of variables appears naturally in many relevant settings. For instance, to encode Subset Sum as an instance of ILP Feasibility it suffices to introduce a $\{0,1\}$-variable x_i for every input number s_i, and then set only one constraint: $\sum_{i=1}^{n} s_i x_i = t$, where t is the target value. Note that the fact that Subset Sum is NP-hard for the binary encoding of the input and polynomial-time solvable for the unary encoding, explains why $\|A\|_\infty$ is also a relevant parameter for the complexity of the problem. Integer linear programs with few constraints and many variables arise most often in the study of knapsack-like and scheduling problems via the concept of so-called *configuration ILPs*, in the context of approximation and parameterized algorithms.

Parameterization by the number of constraints. Probably the first to study the complexity of integer linear programming with few constraints was Papadimitriou [29], who already in 1981 observed the following. Consider an ILP of the standard form $\{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geqslant 0\}$, where A is an integer matrix with k rows (constraints) and ℓ columns (variables), $\mathbf{x}$ is a vector of integer variables, and $\mathbf{b}$ is a vector of integers. Papadimitriou proved that assuming such an ILP is feasible, it admits a solution with all variables bounded by $B = \ell \cdot ((\|A\|_\infty + \|\mathbf{b}\|_\infty) \cdot k)^{2k+1}$, which in turn can be found in time $\mathcal{O}((\ell B)^{k+1} \cdot |I|)$ using simple dynamic programming. Noting that by removing duplicate columns one can assume that $\ell \leqslant (2\|A\|_\infty + 1)^k$, this yields an algorithm with running time $((\|A\|_\infty + \|\mathbf{b}\|_\infty) \cdot k)^{\mathcal{O}(k^2)}$. The approach can be lifted

to give an algorithm with a similar running time bound also for the ILP Optimization problem, where instead of finding any feasible solution $\mathbf{x}$, we look for one that maximizes the value $\mathbf{w}^\intercal\mathbf{x}$ for a given optimization goal vector $\mathbf{w}$.

The result of Papadimitriou was recently improved by Eisenbrand and Weismantel [9], who used the Steinitz Lemma to give an amazingly elegant algorithm solving the ILP Optimization problem (and thus also the ILP Feasibility problem) for a given instance $\{\max \mathbf{w}^\intercal\mathbf{x}\colon A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ with k constraints in time $(k\|A\|_\infty)^{\mathcal{O}(k)} \cdot \|b\|_\infty^2$. This running time has been subsequently refined by Jansen and Rohwedder [17] to $\mathcal{O}(k\|A\|_\infty)^{2k} \cdot \log \|\mathbf{b}\|_\infty$ in the case of ILP Optimization, and to $\mathcal{O}(k\|A\|_\infty)^k \cdot \log \|\mathbf{b}\|_\infty$ in the case of ILP Feasibility.

From the point of view of fine-grained parameterized complexity, this raises the question of whether the parametric factor $\mathcal{O}(k\|A\|_\infty)^k$ is the best possible. Jansen and Rohwedder [17] studied this question under the assumption that k is a fixed constant and $\|A\|_\infty$ is the relevant parameter. They proved that assuming the Strong Exponential Time Hypothesis (SETH), for every fixed k there is no algorithm with running time $(k \cdot (\|A\|_\infty + \|\mathbf{b}\|_\infty))^{k-\delta} \cdot |I|^{\mathcal{O}(1)}$, for any $\delta > 0$. Note that as k is considered a fixed constant, this essentially shows that the degree of $\|A\|_\infty$ needs to be at least k, but does not exclude algorithms with running time of the form $\|A\|_\infty^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$, or $2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ when all entries in the input matrix A are in $\{-1, 0, 1\}$. On the other hand, the algorithms of [9, 17] provide only an upper bound of $2^{\mathcal{O}(k \log k)} \cdot |I|^{\mathcal{O}(1)}$ in the latter setting. As observed by Fomin et al. [10], a trivial encoding of 3SAT as an ILP shows a lower bound of $2^{o(k)} \cdot |I|^{\mathcal{O}(1)}$ for instances with A having entries only in $\{0, 1\}$, $\mathbf{b}$ having entries only in $\{0, 1, 2, 3\}$, and $\ell = \mathcal{O}(k)$. This still leaves a significant gap between the $2^{o(k)} \cdot |I|^{\mathcal{O}(1)}$ lower bound and the $2^{\mathcal{O}(k \log k)} \cdot |I|^{\mathcal{O}(1)}$ upper bound.

Parameterization by the dual treedepth. A related, recent line of research concerns ILPs that may have many constraints, but these constraints need to be somehow organized in a structured, "shallow" way. It started with a result of Hemmecke et al. [13], who gave a fixed-parameter tractable algorithm for solving the so-called *n-fold ILPs*. An n-fold ILP is an ILP where the constraint matrix is of the form

$$A = \begin{pmatrix} B & B & \dots & B \\ C & 0 & \cdots & 0 \\ 0 & C & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C \end{pmatrix},$$

and the considered parameters are the dimensions of matrices B and C, as well as $\|A\|_\infty$. The running time obtained by Hemmecke et al. is $\|A\|_\infty^{\mathcal{O}(k^3)} \cdot |I|^{\mathcal{O}(1)}$ when all these dimensions are bounded by k. See [13] and the recent improvements of Eisenbrand et al. [8] for more refined running time bounds expressed in terms of particular dimensions.

The result of Hemmecke et al. [13] quickly led to multiple improvements in the best known upper bounds for several parameterized problems, where the technique of configuration ILPs is applicable [20, 21, 22]. Recently, the technique was also applied to improve the running times of several approximation schemes for scheduling problems [16]. Chen and Marx [6] introduced a more general concept of *tree-fold ILPs*, where the "star-like" structure of an n-fold ILP is generalized to any bounded-depth rooted tree, and they showed that it retains relevant fixed-parameter tractability results. This idea was followed on by Eisenbrand et al. [8] and by Koutecký et al. [24], whose further generalizations essentially boil down to considering a structural parameter called the *dual treedepth* of the input matrix A. This parameter, denoted $\mathrm{td}_D(A)$, is the smallest number h such that the rows of A can be organized into

a rooted forest of height h with the following property: whenever two rows have non-zero entries in the same column, one is the ancestor of the other in the forest. As shown explicitly by Koutecký et al. [24] and somewhat implicitly by Eisenbrand et al. [8], ILP Optimization can be solved in fixed-parameter time when parameterized by $\|A\|_\infty$ and $\mathrm{td}_D(A)$.

Our results. For the parameterization by the number of constraints k, we close the above mentioned complexity gap by proving the following optimality result.

▶ **Theorem 1.** *Assuming ETH, there is no algorithm that would solve any* ILP feasibility *instance* $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ *with* $A \in \{0,1\}^{k\times\ell}$, $\mathbf{b} \in \mathbb{N}^k$, *and* $\ell, \|\mathbf{b}\|_\infty = \mathcal{O}(k \log k)$ *in time* $2^{o(k \log k)}$.

This shows that the algorithms of [9, 17] have the essentially optimal running time of $2^{\mathcal{O}(k \log k)} \cdot |I|^{\mathcal{O}(1)}$ also in the regime where $\|A\|_\infty$ is a constant and the number of constraints k is the relevant parameter.

The main ingredient of the proof of Theorem 1 is a certain quaint combinatorial construction – *detecting matrices* introduced by Lindström [26] – that provides a general way for compressing a system $A\mathbf{x} = \mathbf{b}$ with k equalities and bounded targets $\|\mathbf{b}\|_\infty \leqslant d$ into $\mathcal{O}(k/\log_d k)$ equalities (with unbounded targets). Each new equality is a linear combination of the original ones; in fact, just taking $\mathcal{O}(k/\log_d k)$ sums of random subsets of the original equalities suffices, but we also provide a deterministic construction taking $\mathcal{O}(dk/\log_d k)$ such subsets. By composing such a compression procedure for $d = 4$ with a standard reduction from (3,4)SAT – a variant of 3SAT where every variable occurs at most 4 times – to ILP Feasibility, we obtain a reduction that given an instance of (3,4)SAT with n variables and m clauses, produces an equivalent instance of ILP Feasibility with $k = \mathcal{O}((n+m)/\log(n+m))$ constraints. Since $2^{o(k \log k)} = 2^{o(n+m)}$, we would obtain a $2^{o(n+m)}$-time algorithm for (3,4)SAT, which is known to contradict ETH. We note that detecting matrices were recently used by two of the authors in the context of different lower bounds based on ETH [3].

For the parameterization by the dual treedepth, we first streamline the presentation of the approach of Koutecký et al. [24] and clarify that the parametric factor in the running time is doubly-exponential in the treedepth. The key ingredient here is the upper bound on ℓ_1-norms of the elements of the *Graver basis* of the input matrix A, expressed in terms of $\|A\|_\infty$ and $\mathrm{td}_D(A)$. Using standard textbook bounds for Graver bases and the recursive definition of treedepth, we prove that these ℓ_1-norms can be bounded by $(2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}$. This, combined with the machinery developed by Koutecký et al. [24], implies the following.

▶ **Theorem 2.** *There is an algorithm that solves any given* ILP Optimization *instance* $I = \{\max \mathbf{w}^\intercal \mathbf{x} \colon A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}\}$ *in time* $\|A\|_\infty^{2^{\mathcal{O}(\mathrm{td}_D(A))}} \cdot |I|^{\mathcal{O}(1)}$.

We remark that the running time as outlined above also follows from a fine analysis of the reasoning presented in [24], but the intermediate step of using tree-fold ILPs in [24] makes tracking parametric dependencies harder to follow.

We next show that the running time provided by Theorem 2 is probably optimal. Namely, we have the following lower bound.

▶ **Theorem 3.** *Assuming ETH, there is no algorithm that would solve any* ILP Feasibility *instance* $I = \{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$, *where all entries of* A *and* $\mathbf{b}$ *are in* $\{-1, 0, 1\}$, *in time* $2^{2^{o(\mathrm{td}_D(A))}} \cdot |I|^{\mathcal{O}(1)}$.

To prove Theorem 3 we reduce from the Subset Sum problem. The key idea is that we are able to "encode" any positive integer s using an ILP with dual treedepth $\mathcal{O}(\log \log s)$.

2 Parameterization by the number of constraints

2.1 Detecting matrices

Our main tool is the usage of so-called *detecting matrices*, first studied by Lindström [26]. They can be explained via the following coin-weighing puzzle: given m coins with weights in $\{0, 1, \ldots, d-1\}$, we want the deduce the weight of each coin with as few weighings as possible. We have a spring scale, so in one weighing we can exactly determine the sum of weights of any subset of the coins. While the naive strategy – weigh coins one by one – yields m weighings, it is actually possible to find a solution using $\mathcal{O}(m/\log_d m)$ weighings. This number is asymptotically optimal, as each weighing provides $\Theta(\log m)$ bits of information, so fewer weighings would not be enough to distinguish all d^m possible weight functions.

Probably the easiest way to construct such a strategy is using the probabilistic method. It turns out that querying $\mathcal{O}(m/\log_d m)$ random subsets of coins with high probability provides enough information to determine the weight of each coin. This is because a random subset distinguishes any of the $\mathcal{O}(d^m \cdot d^m)$ non-equal pairs of weight functions with probability at least $\frac{1}{2}$, but pairs of weight functions that are close to each other are few, while pairs of weight functions that are far from each other have a significantly better probability than $\frac{1}{2}$ of being distinguished. Note that thus we construct a *non-adaptive* strategy: the subsets of coins to be weighed can be determined and fixed at the very start. We refer the reader to e.g. [12, Corollary 2] for full details, and we remark that the last two authors recently used detecting matrices in the context of algorithmic lower bounds for the Multicoloring problem [3].

Viewing each tuple of coin weights as a vector $\mathbf{v} \in \{0, \ldots, d-1\}^m$, each weighing returns the value $\mathbf{a}^\intercal \mathbf{v}$ for the characteristic vector $\mathbf{a} \in \{0,1\}^m$ of some subset of coins. Thus k weighings give the vectors of values $M\mathbf{v}$ for some $\{0,1\}$-matrix M with k rows and m columns. An equivalent formulation is then to ask for a $\{0,1\}$-matrix M with m columns, such that knowing the vector $M\mathbf{v}$ uniquely determines any $\mathbf{v} \in \{0, \ldots, d-1\}^m$. Such an M is called a *d-detecting matrix* and we seek to minimize the number of rows/weighings k it can have. Lindström gave a deterministic construction and proved the bound on k to be tight. See also Bshouty [4] for a more direct and general construction using Fourier analysis.

▶ **Theorem 4** ([26]). *For all $d, m \in \mathbb{N}$, there is a $\{0,1\}$-matrix M with m columns and $k \leqslant \frac{2m \log d}{\log m}(1+o(1))$ rows such that for any $\mathbf{u}, \mathbf{v} \in \{0, \ldots, d-1\}^m$, if $M\mathbf{u} = M\mathbf{v}$ then $\mathbf{u} = \mathbf{v}$. Moreover, such matrix M can be constructed in time polynomial in dm.*

In other words, this allows us to check m equalities between values in $\{0, \ldots, d-1\}$ (i.e., corresponding coordinates of vectors $\mathbf{u}$ and $\mathbf{v}$) using only $\mathcal{O}(m/\log_d m)$ comparisons of sums of certain subsets of these values (i.e., coordinates of vectors $M\mathbf{u}$ and $M\mathbf{v}$). For an ILP instance $A\mathbf{x} = \mathbf{b}$ with $\|\mathbf{b}\|_\infty \leqslant d$ and m constraints, we may use this idea to check the equality on each of the m coordinates of $A\mathbf{x}$ using only $\mathcal{O}(m/\log_d m)$ constraints. Indeed, the intuition is that if M is a d-detecting matrix, then we can rewrite $A\mathbf{x} = \mathbf{b}$ as $MA\mathbf{x} = M\mathbf{b}$ and check the latter – which involves $\mathcal{O}(m/\log_d m)$ $\{0,1\}$-combinations of the original constraints.

This is the core of our approach. However, there is one subtle caveat: in order to claim that the assertions $A\mathbf{x} = \mathbf{b}$ and $MA\mathbf{x} = M\mathbf{b}$ are equivalent, we would need to ensure that $\|A\mathbf{x}\|_\infty \leqslant d$ for an arbitrary vector $\mathbf{x} \in \mathbb{N}^n$. One solution is to use the fact that a uniformly random $\{0,1\}$-matrix has a stronger "detecting" property: it will, with high probability, distinguish all vectors of low ℓ_1-norm, as shown by Grebinski and Kucherov [12].

▶ **Lemma 5** ([12]). *For all $d, m \in \mathbb{N}$, there exists a $\{0,1\}$-matrix M with m columns and $k \leqslant \frac{4m\log(d+1)}{\log m}(1+o(1))$ rows such that for any $\mathbf{u}, \mathbf{v} \in \mathbb{N}^m$ satisfying $\|\mathbf{u}\|_1, \|\mathbf{v}\|_1 \leqslant dm$, if $M\mathbf{u} = M\mathbf{v}$ then $\mathbf{u} = \mathbf{v}$. Moreover, such matrix M can be computed in randomized polynomial time (in dm).*

Note that in Lemma 5, we do not actually have to assume bounds on one of the two vectors: it suffices to assume $\mathbf{u} \in \mathbb{N}^m$ and $\|\mathbf{v}\|_1 \leqslant dm$, because simply adding a single row full of ones to M guarantees $\|\mathbf{u}\|_1 = \|\mathbf{v}\|_1$. Therefore as long as A is non-negative and $\|\mathbf{b}\|_\infty \leqslant d$, it suffices to check $MA\mathbf{x} = M\mathbf{b}$. Unfortunately, to the best of our knowledge, no deterministic construction is known for Lemma 5. We remark that Bshouty gave a deterministic, but adaptive detecting strategy [4]; that is, in terms of coin weighing, consecutive queries on coins may depend on results of previous weighings.

Instead, we show that a different, recursive construction by Cantor and Mills [5] for 2-detecting matrices can be adapted so that no bounds (other than non-negativity) are assumed for one of the vectors, while the other must have all coefficients in $\{0, 1, \ldots, d-1\}$. The proof is deferred to the full version [23], which we mark with (♠).

▶ **Lemma 6** (♠). *For all $d, m \in \mathbb{N}$, there exists a $\{0,1\}$-matrix M with m columns and $k \leqslant \frac{md\log d}{\log m}(1+o(1))$ rows such that for any $\mathbf{u} \in \mathbb{N}^m$ and $\mathbf{v} \in \{0,1,\ldots,d-1\}^m$, if $M\mathbf{u} = M\mathbf{v}$ then $\mathbf{u} = \mathbf{v}$. Moreover, such matrix M can be computed in time polynomial in dm.*

We remark that the bounds in Theorem 4 and Lemma 5 were also shown to be tight. Lemma 6 gives matrices that are also d-detecting, in particular, hence the bound is tight for $d = 2$ (and tight up to an $\mathcal{O}(d)$ factor in general).

Note also that we can relax the non-negativity constraint to requiring that $\mathbf{u} \in \mathbb{Z}^m$ is any integer with all entries lower bounded by $-\lfloor \frac{d}{2} \rfloor$ and $\mathbf{v} \in \{-\lfloor \frac{d}{2} \rfloor, \ldots, \lfloor \frac{d}{2} \rfloor\}^m$. This is because $M\mathbf{u} = M\mathbf{v}$ is equivalent to $M(\mathbf{u}+\mathbf{c}) = M(\mathbf{v}+\mathbf{c})$ where $\mathbf{c}$ is the constant $\lfloor \frac{d}{2} \rfloor$ vector. This allows to use the same detecting matrix for such pairs of vectors as well. However, note that some lower bound on the coefficients of $\mathbf{u}$ is necessary, since even if we fix $\mathbf{v} = 0$, the matrix M has a non-trivial kernel, giving many non-zero vectors $\mathbf{u} \in \mathbb{Z}^m$ satisfying $M\mathbf{u} = M\mathbf{v}$.

2.2 Coefficient reduction

In further constructions, we will need a way to reduce coefficients in a given ILP Feasibility instance with a nonnegative constraint matrix A to $\{0,1\}$. We now prove that this can be done in a standard way by replacing each constraint with $\mathcal{O}(\log \|A\|_\infty)$ constraints that check the original equality bit by bit. Here and throughout this paper we use the convention that for a vector $\mathbf{x}$, by x_i we denote the i-th entry of $\mathbf{x}$.

▶ **Lemma 7** (Coefficient Reduction). *Consider an instance $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility*, where $\mathbf{b} \in \mathbb{N}^k$ and A is a nonnegative integer matrix with k rows and ℓ columns. In polynomial time, this instance can be reduced to an equivalent instance $\{A'\mathbf{x} = \mathbf{b}', \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *where A' is a $\{0,1\}$-matrix with $k' = \mathcal{O}(k \log \|A\|_\infty)$ rows and $\ell' = \ell + \mathcal{O}(k \log \|A\|_\infty)$ columns, and $\mathbf{b}' \in \mathbb{N}^{k'}$ is a vector with $\|\mathbf{b}'\|_\infty = \mathcal{O}(\|\mathbf{b}\|_\infty)$.*

Proof. Denote $\delta = \lceil \log(1 + \|A\|_\infty) \rceil = \mathcal{O}(\log \|A\|_\infty)$. Consider a single constraint $\mathbf{a}^\intercal \mathbf{x} = b$, where $\mathbf{a} \in \mathbb{N}^\ell$ is a row of A and $b \in \mathbb{N}$ is an entry of $\mathbf{b}$. Let $a_i[j]$ be the j-th bit of a_i, the i-th entry of vector $\mathbf{a}$; similarly for b. By choice of δ, $\|\mathbf{a}\|_\infty \leqslant 2^\delta - 1$, so each entry of $\mathbf{a}$ has up to δ binary digits. Now, for $\mathbf{x} \in \mathbb{Z}^n$, the constraint $\mathbf{a}^\intercal \mathbf{x} = b$ is equivalent to

$$\sum_{j=0}^{\delta-1} 2^j \left(\sum_{i=1}^{n} a_i[j] x_i \right) = b\,.$$

We rewrite this equation into δ equations, each responsible for verifying one bit. For this, we introduce $\delta - 1$ carry variables $y_0, y_1, \ldots, y_{\delta-2}$ and emulate the standard algorithm for adding binary numbers by writing equations

$$y_{j-1} + \sum_{i=1}^{n} a_i[j]x_i = b[j] + 2y_j \qquad \text{for } j = 0, \ldots, \delta - 1,$$

where y_{-1} and $y_{\delta-1}$ are replaced with 0 and $b[\delta-1]$ is replaced with the number whose binary digits are (from the least significant): $b[\delta-1], b[\delta], b[\delta+1], \ldots$ (we do this because b may have more than δ digits). To get rid of the variable y_j on the right-hand side, we let $B = 2^{\lceil \log b \rceil}$ and introduce two new variables y'_j, y''_j for each carry variable y_j, with constraints

$$y_j + y'_j = B \quad \text{and} \quad y_j + y''_j = B \quad \text{for } j = 0, \ldots, \delta - 2,$$

which is equivalent to $y'_j = y''_j = B - y_j$. Hence the previous equations can be replaced by

$$y_{j-1} + \sum_{i=1}^{n} a_i[j]x_i + y'_j + y''_j = b[j] + 2B \qquad \text{for } j = 0, \ldots, \delta - 1.$$

We thus replace each row of A with $2(\delta - 1) + \delta$ rows and $3(\delta - 1)$ auxiliary variables. ◀

2.3 Proof of Theorem 1

The Exponential Time Hypothesis states that for some $0 < c < 1$, 3SAT with n variables cannot be solved in time $\mathcal{O}^\star(2^{cn})$ (the $\mathcal{O}^\star$ notation hides polynomial factors). It was introduced by Impagliazzio, Paturi, and Zane [15] and developed by Impagliazzo and Paturi [14] to become a central conjecture for proving tight lower bounds for the complexity of various problems. While the original statement considers the parameterization by the number of variables, the *Sparsification Lemma* [14] allows us to assume that the number of clauses is linear in the number of variables, and hence we have the following.

▶ **Theorem 8** (see e.g. Theorem 14.4 in [7]). *Unless ETH fails, there is no algorithm for* 3SAT *that runs in time* $2^{o(n+m)}$*, where* n *and* m *denote the numbers of variables and clauses, respectively.*

We now proceed to the proof of Theorem 1. Our first step is to decrease the number of occurrences of each variable. The (3,4)SAT is the variant of 3SAT where each clause uses exactly 3 different variables and every variable occurs in at most 4 clauses. Tovey [30] gave a linear reduction from 3SAT to (3,4)SAT, i.e., an algorithm that, given an instance of 3SAT with n variables and m clauses, in linear time constructs an equivalent instance of (3,4)SAT with $\mathcal{O}(n + m)$ variables and clauses. In combination with Theorem 8 this yields:

▶ **Corollary 9.** *Unless ETH fails, there is no algorithm for* (3,4)SAT *that runs in time* $2^{o(n+m)}$*, where* n *and* m *denote the numbers of variables and clauses, respectively.*

We now reduce (3,4)SAT to ILP Feasibility. A $(3, 4)$SAT instance φ with n variables and m clauses can be encoded in a standard way as an ILP Feasibility instance with $\mathcal{O}(n + m)$ variables and constraints as follows. For each formula variable v we introduce two ILP variables x_v and $x_{\neg v}$ with a constraint $x_v + x_{\neg v} = 1$ (hence exactly one of them should be 1, the other 0). For each clause c we introduce two auxiliary slack variables y_c, z_c and two constraints: $y_c + z_c = 2$ and $x_{\ell_1} + x_{\ell_2} + x_{\ell_3} + y_c = 3$, where ℓ_1, ℓ_2, ℓ_3 are the three literals in c. Since y_c, z_c will not appear in any other constraints, the first constraint is equivalent to

ensuring that $y_c \leqslant 2$, so the second constraint is equivalent to $x_{\ell_1} + x_{\ell_2} + x_{\ell_3} \geqslant 1$. This way, one can reduce in polynomial time a $(3,4)$SAT instance φ with n variables and m clauses into an equivalent instance $\{\mathbf{x} \in \mathbb{Z}^\ell \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of ILP FEASIBILITY where:

- the constraint matrix A has $k := n + 2m$ rows and $\ell := 2n + 2m$ columns;
- each entry in A is zero or one;
- each row and column of A contains at most 4 non-zero entries; and
- the target vector $\mathbf{b}$ has all entries equal to 1, 2, or 3;

We now reduce the obtained instance to another ILP FEASIBILITY instance containing only $\mathcal{O}((n+m)/\log(n+m))$ constraints. Let M be the detecting matrix given by Lemma 6 for $d = 4$ and the required number of columns (m in the notation of the statement of Lemma 6) equal to the number or rows (constraints) in A, which is k. Then for any $\mathbf{x} \in \mathbb{N}^\ell$, we have $A\mathbf{x} \in \mathbb{N}^k$ (since A is non-negative) and $\mathbf{b} \in \{0, \ldots, d-1\}^k$, hence by Lemma 6 we have that $A\mathbf{x} = \mathbf{b}$ if and only if $MA\mathbf{x} = M\mathbf{b}$. We conclude that the ILP FEASIBILITY instance $\{\mathbf{x} \in \mathbb{Z}^\ell \mid A'\mathbf{x} = \mathbf{b}', \mathbf{x} \geqslant 0\}$ with $A' = MA$ and $\mathbf{b}' = M\mathbf{b}$ is equivalent to the previous instance $\{\mathbf{x} \in \mathbb{Z}^\ell \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$.

The new instance has the same number $\ell' = \ell = 2n + 2m$ of variables, but only $k' = \mathcal{O}(k/\log k) = \mathcal{O}((n+m)/\log(n+m))$ constraints. The entries of $\mathbf{b}' = M\mathbf{b}$ are non-negative and bounded by $k \cdot \|\mathbf{b}\|_\infty = \mathcal{O}(n+m)$. Similarly, the entries of $A' = MA$ are non-negative, and since every column of A has at most 4 non-zero entries, we get $\|A'\|_\infty \leqslant 4$.

To further reduce $\|A'\|_\infty$, we apply Lemma 7, replacing each row of A' by a constant number of $\{0,1\}$-rows and auxiliary variables. This way, we reduced in polynomial time a $(3,4)$SAT instance φ with n variables and m clauses into an equivalent ILP FEASIBILITY instance $\{\mathbf{x} \in \mathbb{Z}^{\ell''} \mid A''\mathbf{x} = \mathbf{b}'', \mathbf{x} \geqslant 0\}$, where A'' is a $\{0,1\}$-matrix with $\ell'' = \ell' + \mathcal{O}(k') = \mathcal{O}(n+m)$ columns and $k'' = \Theta(k') = \Theta((n+m)/\log(n+m))$ rows, while $\|\mathbf{b}''\|_\infty = \mathcal{O}(n+m)$. Hence $\ell'', \|\mathbf{b}''\|_\infty = \mathcal{O}(k'' \log k'')$.

We are now in position to finish the proof of Theorem 1. Suppose there is an algorithm for ILP FEASIBILITY that works in time $2^{o(k'' \log k'')}$ on instances with $A \in \{0,1\}^{k'' \times \ell''}$ and $\ell'', \|\mathbf{b}''\|_\infty = \mathcal{O}(k'' \log k'')$. Then applying the above reduction would solve $(3,4)$SAT instances with $N = n + m$ variables and clauses in time $2^{o((N/\log N) \cdot \log(N/\log N))} = 2^{o(N)}$, which contradicts ETH by Corollary 9. This concludes the proof of Theorem 1.

3 Parameterization by the dual treedepth

3.1 Preliminaries

Treedepth and dual treedepth. For a graph G, the *treedepth* of G, denoted $\mathrm{td}(G)$, can be defined recursively as follows:

$$\mathrm{td}(G) = \begin{cases} 1 & \text{if } G \text{ has one vertex;} \\ \max(\mathrm{td}(G_1), \ldots, \mathrm{td}(G_p)) & \text{if } G \text{ is disconnected and } G_1, \ldots, G_p \\ & \text{are its connected components;} \\ 1 + \min_{u \in V(G)} \mathrm{td}(G - u) & \text{if } G \text{ has more than one vertex} \\ & \text{and is connected.} \end{cases} \tag{1}$$

See e.g. [28]. Equivalently, treedepth is the smallest possible height of a rooted forest F on the same vertex set as G such that whenever uv is an edge in G, then u is an ancestor of v in F or vice versa.

Since we focus on constraints, we consider, for a matrix A, the *constraint graph* or *dual graph* $G_D(A)$, defined as the graph with rows of A as vertices where two rows are adjacent if and only if in some column they simultaneously contain a non-zero entry. The *dual treedepth* of A, denoted $\mathrm{td}_D(A)$, is the treedepth of $G_D(A)$.

The recursive definition (1) is elegantly reinterpreted in terms of row removals and partitioning into blocks as follows. A matrix A is *block-decomposable* if after permuting its rows and columns it can be presented in block-diagonal form, i.e., rows and columns can be partitioned into intervals $R_1, \ldots, R_p$ and $C_1, \ldots, C_p$, for some $p \geqslant 2$, such that non-zero entries appear only in blocks $B_1, \ldots, B_p$, where B_i is the block of entries at intersections of rows from R_i with columns from C_i. It is easy to see that A is block-decomposable if and only if $G_D(A)$ is disconnected, and the finest block decomposition of A corresponds to the partition of $G_D(A)$ into connected components. The blocks $B_1, \ldots, B_p$ in this finest partition are called the *block components* of A – they are not block-decomposable. Then the recursive definition of treedepth provided in (1) translates to the following definition of the dual treedepth of a matrix A:

$$\mathrm{td}_D(A) = \begin{cases} 1 & \text{if } A \text{ has one row;} \\ \max(\mathrm{td}_D(B_1), \ldots, \mathrm{td}_D(B_p)) & \text{if } A \text{ is block-decomposable and } B_1, \ldots, B_p \text{ are its block components;} \\ 1 + \min\limits_{\mathbf{a}^\intercal \colon \text{rows of } A} \mathrm{td}_D(A \backslash \mathbf{a}^\intercal) & \text{if } A \text{ has more than one row and is not block decomposable.} \end{cases} \tag{2}$$

Here $A \backslash \mathbf{a}^\intercal$ is the matrix obtained from A by removing the row $\mathbf{a}^\intercal$. Intuitively, dual treedepth formalizes the idea that a block-decomposable matrix is as hard as the hardest of its block components, and that adding a single row makes it a bit harder, but not uncontrollably so.

Graver bases. Two integer vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$ are *sign-compatible* if $a_i \cdot b_i \geqslant 0$ for all $i = 1, \ldots, n$. For $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$ we write $\mathbf{a} \sqsubseteq \mathbf{b}$ if $\mathbf{a}$ and $\mathbf{b}$ are sign-compatible and $|a_i| \leqslant |b_i|$ for all $i = 1, \ldots, n$. Then $\sqsubseteq$ is a partial order on $\mathbb{Z}^n$; we call it the *conformal order*. Note that $\sqsubseteq$ has a unique minimum element, which is the zero vector $\mathbf{0}$.

For a matrix A, the *Graver basis* of A, denoted $\mathcal{G}(A)$ is the set of conformally minimal vectors in $(\ker A \cap \mathbb{Z}^n) - \{\mathbf{0}\}$. It is easy to see by Dickson's lemma that $(\mathbb{Z}^n, \sqsubseteq)$ is a well quasi-ordering, hence there are no infinite antichains with respect to the conformal order. It follows that the Graver basis of every matrix is finite, though it can be quite large. For a matrix A and $p \in [1, \infty]$, we denote $g_p(A) = \max_{\mathbf{u} \in \mathcal{G}(A)} \|\mathbf{u}\|_p$.

3.2 Upper bound

We start with the upper bound for the dual treedepth parameterization, that is, Theorem 2. As explained in the introduction, this result easily follows from the work of Koutecký et al. [24] and the following lemma bounding $g_1(A)$ in terms of $\mathrm{td}_D(A)$ and $\|A\|_\infty$, for any integer matrix A.

▶ **Lemma 10.** *For any matrix A with integer entries, it holds that*

$$g_1(A) \leqslant (2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)} - 1}.$$

Before we prove Lemma 10, let us sketch how using the reasoning from Koutecký et al. [24] one can derive Theorem 2. Using the bound on the ℓ_1-norm of vectors in the Graver

basis of A, we can construct a Λ-*Graver-best oracle* for the considered ILP Optimization instance. This is an oracle that given any feasible solution $\mathbf{x}$, returns another feasible solution $\mathbf{x}'$ that differs from $\mathbf{x}$ only by an integer multiple not larger than Λ of a vector from the Graver basis of A, and among such solution achieves the best goal value of $\mathbf{w}^\intercal\mathbf{x}'$. Such a Λ-Graver-best oracle runs in time $(\|A\|_\infty \cdot g_1(A))^{\mathcal{O}(\mathrm{tw}_D(A))} \cdot |I|^{\mathcal{O}(1)}$, where $\mathrm{tw}_D(A)$ is the treewidth of the constraint graph $G_D(A)$, which is always upper bounded by $\mathrm{td}_D(A)+1$. See the proof of Lemma 25 and the beginning of the proof of Theorem 3 in [24]; the reasoning there is explained in the context of tree-fold ILPs, but it uses only boundedness of the dual treedepth of A. Once a Λ-Graver-best oracle is implemented, we can use it to implement a *Graver-best oracle* (Lemma 14 in [24]) within the same asymptotic running time, and finally use the main theorem – Theorem 1 in [24] – to obtain the algorithm promised in Theorem 2 above.

We now proceed to the proof of Lemma 10.

Proof of Lemma 10. We proceed by induction on the number of rows of A using the recursive definition (2). For the base case – when A has one row – we may use the following well-known bound.

▷ Claim 11 (Lemma 3.5.7 in [27]). If A is an integer matrix with one row, then

$$g_1(A) \leqslant 2\|A\|_\infty + 1.$$

We note that the original bound of $2\|A\|_\infty - 1$, stated in [27], works only for non-zero A.

We now move to the induction step, so suppose the considered matrix A has more than one row. We consider two cases: either A is block-decomposable, or it is not.

First suppose that A is block-decomposable. Let $B_1, \ldots, B_p$ be the block components of A, and let $R_1, \ldots, R_p$ and $C_1, \ldots, C_p$ be the corresponding partitions of rows and columns of A into segments, respectively. Observe that integer vectors $\mathbf{u}$ from $\ker A$ are exactly vectors of the form $(\,\mathbf{v}^{(1)} \mid \mathbf{v}^{(2)} \mid \ldots \mid \mathbf{v}^{(p)}\,)$, where each $\mathbf{v}^{(i)}$ is an integer vector of length $|C_i|$ that belongs to $\ker B_i$. It follows that $\mathcal{G}(A)$ consists of vectors of the following form: for some $i \in \{1, \ldots, p\}$ put a vector from $\mathcal{G}(B_i)$ on coordinates corresponding to the columns of C_i, and fill all the other entries with zeroes. Consequently, we have

$$g_1(A) \leqslant \max_{i=1,\ldots,p} g_1(B_i). \tag{3}$$

On the other hand, by (2) we have

$$\mathrm{td}_D(A) = \max_{i=1,\ldots,p} \mathrm{td}_D(B_i). \tag{4}$$

Since each matrix B_i has fewer rows than A, we may apply the induction assumption to matrices $B_1, \ldots, B_p$, thus inferring by (3) and (4) that

$$g_1(A) \leqslant \max_{i=1,\ldots,p} g_1(B_i) \leqslant \max_{i=1,\ldots,p} (2\|B_i\|_\infty + 1)^{2^{\mathrm{td}_D(B_i)}-1} \leqslant (2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}.$$

We are left with the case when A is not block-decomposable. For this, we use the following claim, which is essentially Lemma 3.7.6 and Corollary 3.7.7 in [27]. The statement there is slightly different, but the same proof in fact proves the following bound; for convenience, we repeat the argument in the full version.

▷ Claim 12 (♠). Let A be an integer matrix and let $\mathbf{a}^\intercal$ be a row of A. Then

$$g_1(A) \leqslant (2\|\mathbf{a}^\intercal\|_\infty + 1) \cdot g_1(A \backslash \mathbf{a}^\intercal) \cdot g_\infty(A \backslash \mathbf{a}^\intercal).$$

Suppose then that A is not block-decomposable. By (2), there exists a row $\mathbf{a}^\intercal$ of A such that $\mathrm{td}_D(A\backslash\mathbf{a}^\intercal) = \mathrm{td}_D(A) - 1$. Then, by Claim 12 and the inductive assumption, we have

$$\begin{aligned} g_1(A) &\leqslant (2\|\mathbf{a}^\intercal\|_\infty + 1) \cdot g_1(A\backslash\mathbf{a}^\intercal) \cdot g_\infty(A\backslash\mathbf{a}^\intercal) \leqslant (2\|A\|_\infty + 1) \cdot (g_1(A\backslash\mathbf{a}^\intercal))^2 \\ &\leqslant (2\|A\|_\infty + 1)^{1+2\cdot(2^{\mathrm{td}_D(A)-1}-1)} = (2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}. \end{aligned}$$

This concludes the proof. ◀

3.3 Lower bound

We now move to the proof of the lower bound, Theorem 3. We will reduce from the Subset Sum problem: given non-negative integers $s_1, \ldots, s_k, t$, encoded in binary, decide whether there is a subset of numbers $s_1, \ldots, s_k$ that sums up to t. The standard NP-hardness reduction from 3SAT to Subset Sum takes an instance of 3SAT with n variables and m clauses, and produces an instance $(s_1, \ldots, s_k, t)$ of Subset Sum with a linear number of numbers and each of them of linear bit-length, that is, $k \leqslant \mathcal{O}(n+m)$ and $0 \leqslant s_1, \ldots, s_k, t < 2^\delta$, for some $\delta \leqslant \mathcal{O}(n+m)$. See e.g. [1] for an even finer reduction, yielding lower bounds for Subset Sum under Strong ETH. By Theorem 8, this immediately implies an ETH-based lower bound for Subset Sum.

▶ **Lemma 13.** *Unless ETH fails, there is no algorithm for* Subset Sum *that would solve any input instance $(s_1, \ldots, s_k, t)$ in time $2^{o(k+\delta)}$, where δ is the smallest integer such that $s_1, \ldots, s_k, t < 2^\delta$.*

The idea for our reduction from Subset Sum to ILP Feasibility is as follows. Given an instance $(s_1, \ldots, s_k, t)$, we first *construct* numbers $s_1, \ldots, s_k$ using ILPs $P_1, \ldots, P_k$, where each P_i uses only constant-size coefficients and has dual treedepth $\mathcal{O}(\log \delta)$. The ILP P_i will have a designated variable z_i and two feasible solutions: one that sets z_i to 0 and one that sets it to s_i. Similarly we can construct an ILP Q that forces a designated variable w to be set to t. Having that, the whole input instance can be encoded using one additional constraint: $z_1 + \ldots + z_k - w = 0$. To construct each P_i, we first create δ variables $y_0, y_1, \ldots, y_{\delta-1}$ that are either all evaluated to 0 or all evaluated to $2^0, 2^1, \ldots, 2^{\delta-1}$, respectively; this involves constraints of the form $y_{j+1} = 2y_j$. Then the number s_i (or 0) can be obtained on a new variable z_i using a single constraint that assembles the binary encoding of s_i. The crucial observations is that the constraint graph $G_D(P_i)$ consists of a path on δ vertices and one additional vertex, and thus has treedepth $\mathcal{O}(\log \delta)$.

We start implementing this plan formally by giving the construction for a single number s.

▶ **Lemma 14.** *For all positive integers δ and s satisfying $0 \leqslant s < 2^\delta$, there exists an instance $P = \{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *with the following properties:*

- *A has all entries in $\{-1, 0, 1, 2\}$ and $\mathrm{td}_D(A) \leqslant \mathcal{O}(\log \delta)$;*
- *$\mathbf{b}$ is a vector with all entries in $\{0, 1\}$; and*
- *P has exactly two solutions $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, where $x_1^{(1)} = 0$ and $x_1^{(2)} = s$.*

Moreover, the instance P can be constructed in time polynomial in $\delta + \log s$.

Proof. We shall use $n+2$ variables, denoted for convenience by $y_0, y_1, \ldots, y_{\delta-1}, z, u$; these are arranged into the variable vector $\mathbf{x}$ of length $\delta + 2$ so that $x_1 = z$. Letting $b_0, b_1, \ldots, b_{\delta-1}$ be the consecutive digits of the number s in the binary encoding, the instance P then looks

as follows:

$$\begin{array}{ccccccccccccccc}
u & + & y_0 & & & & & & & & & & & = & 1 \\
 & & 2y_0 & - & y_1 & & & & & & & & & = & 0 \\
 & & & & 2y_1 & - & y_2 & & & & & & & = & 0 \\
 & & & & & & \ddots & & \ddots & & & & & \vdots \\
 & & & & & & & & 2y_{\delta-2} & - & y_{\delta-1} & & = & 0 \\
 & & b_0y_0 & + & b_1y_1 & + & \ldots & + & b_{\delta-2}y_{\delta-2} & + & b_{\delta-1}y_{\delta-1} & - & z & = & 0
\end{array}$$

Since $0 \leqslant u \leqslant 1$, it is easy to see that P has exactly two solutions in nonnegative integers:

- If one sets $u = 1$, then all the other variables need to be set to 0.
- If one sets $u = 0$, then y_i needs to be set to 2^i for all $i = 0, 1, \ldots, \delta - 1$, and then z needs to be set to s by the last equation.

It remains to analyze the dual treedepth of A. Observe that the constraint graph $G_D(A)$ consists of a path of length δ, plus one vertex corresponding to the last equation that may have an arbitrary neighborhood within the path. Since the path on δ vertices has treedepth $\lceil \log(\delta + 1) \rceil$, it follows that $G_D(A)$ has treedepth at most $1 + \lceil \log(\delta + 1) \rceil \leqslant \mathcal{O}(\log \delta)$. ◀

We note that in the above construction one may remove the variable u and replace the constraint $u + y_0 = 1$ with $y_0 = 1$, thus forcing only one solution: the one that sets the first variable to s. This will be used later.

We are ready to show the core part of the reduction.

▶ **Lemma 15.** *An instance $(s_1, \ldots, s_k, t)$ of* Subset Sum *with $0 \leqslant s_i, t < 2^\delta$ for $i = 1, \ldots, k$, can be reduced in polynomial time to an equivalent instance $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *where the entries of A are in $\{-1, 0, 1, 2\}$, entries of $\mathbf{b}$ are in $\{0, 1\}$, and $\mathrm{td}_D(A) \leqslant \mathcal{O}(\log \delta)$.*

Proof. For each $i \in \{1, \ldots, k\}$, apply Lemma 14 to construct a suitable instance $P_i = \{A_i\mathbf{x} = \mathbf{b}_i, \mathbf{x} \geqslant 0\}$ of ILP Feasibility for $s = s_i$. Also, apply Lemma 14 to construct a suitable instance $Q = \{C\mathbf{x} = \mathbf{d}, \mathbf{x} \geqslant 0\}$ of ILP Feasibility for $s = t$, and modify it as explained after the lemma's proof so that there is only one solution, setting the first variable to t. Let

$$A = \left(\begin{array}{c|cccc} \mathbf{c}^\intercal & & & & \\ \hline A_1 & & & & \\ & A_2 & & & \\ & & \ddots & & \\ & & & A_k & \\ & & & & C \end{array}\right)$$

where

$$\mathbf{c}^\intercal = (\,1\,0\,\ldots\,0 \mid 1\,0\,\ldots\,0 \mid \ldots \mid 1\,0\,\ldots\,0 \mid (-1)\,0\,\ldots\,0\,)$$

with consecutive blocks of lengths equal to the numbers of columns of $A_1, \ldots, A_k$, and C, respectively. Observe that

$$\mathrm{td}_D(A) \leqslant 1 + \max(\mathrm{td}_D(A_1), \ldots, \mathrm{td}_D(A_k), \mathrm{td}_D(C)) = \mathcal{O}(\log \delta).$$

Further, let

$$\mathbf{b}^\intercal = (\,0 \mid \mathbf{b}_1^\intercal \mid \ldots \mid \mathbf{b}_k^\intercal \mid \mathbf{d}^\intercal\,).$$

We now claim that the ILP $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ is feasible if and only if the input instance of Subset Sum has a solution. Indeed, if we denote by $z_1, \ldots, z_k, w$ the variables corresponding to the first columns of blocks $A_1, \ldots, A_k, C$, respectively, then by Lemma 14 within each block A_i there are two ways of evaluating variables corresponding to columns of A_i: one setting $z_i = 0$ and second setting $z_i = s_i$. However, there is only one way of evaluating the variables corresponding to columns of C, which sets $w = t$. The first row of A then constitutes the constraint $z_1 + \ldots + z_k - w = 0$, which can be satisfied by setting z_i-s and w as above if and only if some subset of the numbers $s_1, \ldots, s_k$ sums up to t. ◀

It remains to reduce the entries in A equal to 2, simply by duplicating variables.

▶ **Lemma 16 (♠).** *An instance* $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ *of* ILP Feasibility *where the entries of* A *are in* $\{-1, 0, 1, 2\}$ *and the entries of* $\mathbf{b}$ *are in* $\{0, 1\}$ *can be reduced in polynomial time to an equivalent instance* $\{A'\mathbf{x} = \mathbf{b}', \mathbf{x} \geqslant 0\}$ *of* ILP Feasibility *with all entries in* $\{-1, 0, 1\}$ *and* $\mathrm{td}_D(A') \leqslant \mathrm{td}_D(A) + 1$.

Theorem 3 now follows by observing that combining the reductions of Lemma 15 and Lemma 16 with a hypothetical algorithm for ILP Feasibility on $\{-1, 0, 1\}$-input with running time $2^{2^{o(\mathrm{td}_D(A))}} \cdot |I|^{\mathcal{O}(1)}$ would yield an algorithm for Subset Sum with running time $2^{o(k+\delta)}$, contradicting ETH by Lemma 13.

4 Conclusions

We conclude this work by stating two concrete open problems in the topic.

First, apart from considering the standard form $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$, Eisenbrand and Weismantel [9] also studied the more general setting of ILPs of the form $\{A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}\}$, where $\mathbf{l}$ and $\mathbf{u}$ are integer vectors. That is, instead of only requiring that every variable is nonnegative, we put an arbitrary lower and upper bound on the values it can take. Note that such lower and upper bounds can be easily emulated in the standard formulation using slack variables, but this would require adding more constraints to the matrix A; the key here is that we *do not* count these lower and upper bounds in the total number of constraints k. For this more general setting, Eisenbrand and Weismantel [9] gave an algorithm with running time $k^{\mathcal{O}(k^2)} \cdot \|A\|_\infty^{\mathcal{O}(k^2)} \cdot |I|^{\mathcal{O}(1)}$, which boils down to $2^{\mathcal{O}(k^2 \log k)} \cdot |I|^{\mathcal{O}(1)}$ when $\|A\|_\infty = \mathcal{O}(1)$. (A typo leading to a $2^{\mathcal{O}(k^2)} \cdot |I|^{\mathcal{O}(1)}$ bound has been fixed in later versions of the paper). Is this running time optimal or could the $2^{\mathcal{O}(k^2 \log k)}$ factor be improved?

Second, in this work we studied the parameter dual treedepth of the constraint matrix A, but of course one can also consider the *primal treedepth*. It can be defined as the treedepth of the graph over the columns (variables) of A, where two columns are adjacent if they have a non-zero entry in same row (the variables appear simultaneously in some constraint). It is known that ILP Feasibility and ILP Optimization are fixed-parameter tractable when parameterized by $\|A\|_\infty$ and $\mathrm{td}_P(A)$, that this, there is an algorithm with running time $f(\|A\|_\infty, \mathrm{td}_P(A)) \cdot |I|^{\mathcal{O}(1)}$, for some function f [24]. Again, the key ingredient here is an inequality on ℓ_∞-norms of the elements of the Graver basis of any integer matrix A: $g_\infty(A) \leqslant h(\|A\|_\infty, \mathrm{td}_P(A))$ for some function h. Unfortunately, the known proofs of this fact[1], see [2][2], use the theory of well quasi-orderings (in a highly non-trivial way) and consequently

[1] Very recently, Klein [19] provided an alternative constructive proof for 2-stage and multistage IPs.

[2] The work of Aschenbrenner and Hemmecke [2] considers the setting of *multi-stage stochastic programming*, which is related to primal treedepth in the same way as tree-fold ILPs are related to dual treedepth. The translation between MSSP and primal treedepth was formulated by Koutecký et al. [24].

give no direct bounds on the function h. A good upper bound on h would directly lead to a correspondingly efficient FPT algorithm for ILP Optimization parameterized by $\|A\|_\infty$ and $\mathrm{td}_P(A)$. However, we conjecture that the function h has to be non-elementary in $\mathrm{td}_P(A)$. If this was the case, an example could likely be used to prove a non-elementary lower bound under ETH for ILP Feasibility under that $\mathrm{td}_P(A)$ parameterization (with $\|A\| = \mathcal{O}(1)$).

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for Subset Sum and Bicriteria Path. In *SODA 2019*, pages 41–57. SIAM, 2019.

2 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness Theorems in Stochastic Integer Programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007.

3 Marthe Bonamy, Łukasz Kowalik, Michał Pilipczuk, Arkadiusz Socała, and Marcin Wrochna. Tight Lower Bounds for the Complexity of Multicoloring. In *ESA 2017*, volume 87 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

4 Nader H. Bshouty. Optimal Algorithms for the Coin Weighing Problem with a Spring Scale. In *COLT 2009*, 2009.

5 David G Cantor and WH Mills. Determination of a subset from certain combinatorial properties. *Canad. J. Math*, 18:42–48, 1966.

6 Lin Chen and Dániel Marx. Covering a tree with rooted subtrees — parameterized and approximation algorithms. In *SODA 2018*, pages 2801–2820. SIAM, 2018.

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

8 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster Algorithms for Integer Programs with Block Structure. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.

9 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for Integer Programming using the Steinitz Lemma. In *SODA 2018*, pages 808–816. SIAM, 2018.

10 Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. On the Optimality of Pseudo-polynomial Algorithms for Integer Programming. In *ESA 2018*, volume 112 of *LIPIcs*, pages 31:1–31:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.

11 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

12 Vladimir Grebinski and Gregory Kucherov. Optimal Reconstruction of Graphs under the Additive Model. *Algorithmica*, 28(1):104–124, 2000.

13 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-Fold integer programming in cubic time. *Math. Program.*, 137(1-2):325–341, 2013.

14 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

16 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the Configuration-IP — New PTAS Results for Scheduling with Setups Times. In *ITCS 2019*, volume 124 of *LIPIcs*, pages 44:1–44:19. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019.

17 Klaus Jansen and Lars Rohwedder. On Integer Programming and Convolution. In *ITCS 2019*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019.

18 Ravi Kannan. Minkowski's Convex Body Theorem and Integer Programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

19 Kim-Manuel Klein. About the Complexity of Two-Stage Stochastic IPs. *CoRR*, abs/1901.01135, 2019.

20 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *J. Scheduling*, 21(5):493–503, 2018.

21 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold Integer Programming and Applications. In *ESA 2017*, volume 87 of *LIPIcs*, pages 54:1–54:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

22 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and Bribing in Single-Exponential Time. In *STACS 2017*, volume 66 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

23 Dušan Knop, Michał Pilipczuk, and Marcin Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *CoRR*, abs/1811.01296, 2018. URL: `http://arxiv.org/abs/1811.01296`.

24 Martin Koutecký, Asaf Levin, and Shmuel Onn. A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.

25 Hendrik W. Lenstra. Integer Programming with a Fixed Number of Variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

26 Bernt Lindström. On a combinatorial problem in number theory. *Canad. Math. Bull*, 8(4):477–490, 1965.

27 Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*, volume 14 of *MOS-SIAM Series on Optimization*. SIAM, 2013.

28 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity — Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

29 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

30 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

The Set Cover Conjecture and Subgraph Isomorphism with a Tree Pattern

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel
robert.krauthgamer@weizmann.ac.il

Ohad Trabelsi
Weizmann Institute of Science, Rehovot, Israel
ohad.trabelsi@weizmann.ac.il

Abstract

In the Set Cover problem, the input is a ground set of n elements and a collection of m sets, and the goal is to find the smallest sub-collection of sets whose union is the entire ground set. The fastest algorithm known runs in time $O(mn2^n)$ [Fomin et al., WG 2004], and the Set Cover Conjecture (SeCoCo) [Cygan et al., TALG 2016] asserts that for every fixed $\varepsilon > 0$, no algorithm can solve Set Cover in time $2^{(1-\varepsilon)n}\mathsf{poly}(m)$, even if set sizes are bounded by $\Delta = \Delta(\varepsilon)$. We show strong connections between this problem and kTree, a special case of Subgraph Isomorphism where the input is an n-node graph G and a k-node tree T, and the goal is to determine whether G has a subgraph isomorphic to T.

First, we propose a weaker conjecture Log-SeCoCo, that allows input sets of size $\Delta = O(1/\varepsilon \cdot \log n)$, and show that an algorithm breaking Log-SeCoCo would imply a faster algorithm than the currently known $2^n\mathsf{poly}(n)$-time algorithm [Koutis and Williams, TALG 2016] for Directed nTree, which is kTree with $k = n$ and arbitrary directions to the edges of G and T. This would also improve the running time for Directed Hamiltonicity, for which no algorithm significantly faster than $2^n\mathsf{poly}(n)$ is known despite extensive research.

Second, we prove that if p-Partial Cover, a parameterized version of Set Cover that requires covering at least p elements, cannot be solved significantly faster than $2^n\mathsf{poly}(m)$ (an assumption even weaker than Log-SeCoCo) then kTree cannot be computed significantly faster than $2^k\mathsf{poly}(n)$, the running time of the Koutis and Williams' algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Discrete optimization; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Conditional lower bounds, Hardness in P, Set Cover Conjecture, Subgraph Isomorphism

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.45

Related Version This paper is based on two preliminary versions arXiv:1711.08041 and arXiv:1708.07591.

Funding *Robert Krauthgamer*: Work supported in part by the Israel Science Foundation grant #1086/18, ONR Award N00014-18-1-2364, a Minerva Foundation grant, and a Google Faculty Research Award. Part of this work was done while was visiting the Simons Institute for the Theory of Computing.

1 Introduction

Set Cover and Subgraph Isomorphism are two of the most well-researched problems in theoretical computer science. In this paper we show a strong connection between their time complexity. We first discuss each, and then show our results.

Set Cover

In the Set Cover problem, the input is a ground set $[n] = \{1, ..., n\}$ and a collection of m sets, and the goal is to find the smallest sub-collection of sets whose union is the entire ground set. An exhaustive search takes $O(n2^m)$ time, and a dynamic-programming algorithm has running time $O(mn2^n)$ [15], which is faster when $m > n$, a common assumption that we will make throughout. In spite of extensive effort, no algorithm that runs in time $O^*(2^{(1-\varepsilon)n})$ is known, although some improvements are known in special cases [22, 9, 30, 10]. Here and throughout, $O^*(\cdot)$ hides polynomial factors in the instance size, and unless stated otherwise, $\varepsilon > 0$ denotes a fixed constant (and similarly ε'). Thus, it was conjectured that the above running time is optimal [12], even if the input sets are small. To state this more formally, let Δ-Set Cover denote the Set Cover problem where all sets have size at most $\Delta > 0$.

▶ **Conjecture 1.1** (Set Cover Conjecture (SeCoCo) [12])**.** *For every fixed $\varepsilon > 0$ there is $\Delta(\varepsilon) > 0$, such that no algorithm (even randomized) solves Δ-Set Cover in time $O^*(2^{(1-\varepsilon)n})$.*

This conjecture clearly implies that for every $\Delta = \omega(1)$, no algorithm solves Δ-Set Cover in time $O^*(2^{(1-\varepsilon)n})$. Several conditional lower bounds were based on this conjecture (by reducing Set Cover to it) in the recent decade, including for Steiner Tree, Set Partitioning, and more [12, 11, 8, 24, 25]. The authors of [12] asked whether the problems they reduce Set Cover to can be reduced *back* to Set Cover, so that their running time complexity would stand and fall with SeCoCo. They believed it would be hard to do, since it would probably provide for those problems an alternative algorithm with running time that matches the currently fastest one, which is very complex and took decades to achieve for some (e.g., for Steiner Tree).

Connection to SETH

No formal connection is known to date between the SeCoCo conjecture and the Strong Exponential Time Hypothesis (SETH) of [18], which asserts that for every $\varepsilon > 0$ there exists $k(\varepsilon)$, such that kSAT on N variables and M clauses cannot be solved in time $O^*(2^{(1-\varepsilon)N})$. Cygan et al. [12] provided a partial answer by showing a SETH-based lower bound for a certain variant of Set Cover (that counts the number of solutions). It is known that the weaker assumption ETH implies a $2^{\Omega(n)}$ time lower bound for Set Cover, even if $\Delta = O(1)$, and that SAT can be solved in time $O^*(2^{(1-\varepsilon)N})$ if and only if Set Cover can be solved in time $O^*(2^{(1-\varepsilon')m})$, see [12]. Some researchers hesitate to rely on SeCoCo as a conjecture, and prefer other, more popular conjectures such as SETH. For example, a running time lower bound for Subset Sum was recently shown [1] based on SETH, even though a lower bound based on SeCoCo was already known [12].

We address the necessity of SeCoCo by proposing a weaker assumption, and showing an independent justification for it. Our conjecture deals with Δ-Set Cover for $\Delta = O(\log n)$, as follows.

▶ **Conjecture 1.2** (Logarithmic Set Cover Conjecture (Log-SeCoCo))**.** *For every fixed $\varepsilon > 0$, there is $\Delta(\varepsilon, n) = O(1/\varepsilon \cdot \log n)$ such that no algorithm (even randomized) solves Δ-Set Cover in time $O^*(2^{(1-\varepsilon)n})$.*

The fastest algorithm known for Δ-Set Cover runs in time $O^*(2^{n\lambda_\Delta})$ [22] for $\lambda_\Delta = (2\Delta - 2)/\sqrt{(2\Delta-1)^2 - 2\ln(2)} \leq 1 - 1/(2\Delta)$, where the inequality assumes $\Delta \geq 2$, hence this running time is slightly faster than for general Set Cover. All known hardness results that are based on SeCoCo can be based also on our conjecture, with appropriate adjustments related to the set sizes in Set Cover parameterized by the universe size plus the solution size [12] and in Parity of Set Covers [8].

Subgraph Isomorphism with a tree pattern

The Subgraph Isomorphism problem asks whether a host graph G contains a copy of a pattern graph H as a subgraph. It is well known to be NP-hard since it generalizes hard problems such as Maximum Clique and Hamiltonicity [21], but unlike many natural NP-hard problems, it requires $N^{\Omega(N)}$ time where $N = |V(G)| + |V(H)|$ is the total number of vertices, assuming the exponential time hypothesis (ETH) [13]. Hence, most past research addressed its special cases that are in P, including the case where the pattern graph is of constant size [28], or when both graphs are trees [2], biconnected outerplanar graphs [26], two-connected series-parallel graphs [27], and more [14, 29]. We will focus on a version called kTree, where the pattern is a tree T on k nodes. In the directed version of the problem, denoted Directed kTree, the edges of G and T are oriented, allowing also anti-parallel edges in G[1]. Throughout, unless accompanied with the word *directed*, kTree and nTree refer to their undirected versions. Directed kTree can only be harder than kTree - even when the directed tree T is an arborescence, as one can reduce the undirected version to it with essentially no loss[2]. A couple of different techniques were used in order to design algorithms for Directed kTree. The color-coding method, designed by Alon, Yuster, and Zwick [3], yields an algorithm with running time $O^*((2e)^k)$. Later, a new method utilized kMLD (stands for k Multilinear Monomial Detection – the problem of detecting multilinear monomials of degree k in polynomials presented as circuits) to design a Directed kTree algorithm with running time $O^*(2^k)$ [23].

Our Results

The first result connects our conjecture to the Directed nTree problem (see Figure 1), which is Directed kTree with $k = n$. This problem includes as a special case the well known Directed Hamiltonicity problem, which asks to determine whether a directed graph G contains a simple path (or cycle) that visits all the nodes (the Hamiltonian cycle and path problems are easily reducible to each other with only small overhead). Next, we show that an algorithm that breaks Log-SeCoCo implies a fast algorithm for Directed nTree.

▶ **Theorem 1.3.** *Suppose Log-SeCoCo fails, namely, there is $\varepsilon > 0$ such that for every $\Delta = O(1/\varepsilon \cdot \log n)$, Δ-Set Cover can be solved in time $O^*(2^{(1-\varepsilon)n})$. Then for some $\delta(\varepsilon) > 0$, Directed nTree on $\tilde{n}$ nodes can be solved in time $O^*(2^{(1-\delta)\tilde{n}})$. This holds even when in Δ-Set Cover, every optimal solution is of size $O(\varepsilon n/\log n)$ and consists of disjoint sets.*

In the special case of Directed Hamiltonicity, we actually reduce to rather constrained instances of Set Cover.

▶ **Theorem 1.4.** *Suppose Log-SeCoCo fails, namely, there is $\varepsilon > 0$ such that for every $\Delta = O(1/\varepsilon \cdot \log n)$, Δ-Set Cover can be solved in time $O^*(2^{(1-\varepsilon)n})$. Then for some $\delta(\varepsilon) > 0$, Directed Hamiltonicity on $\tilde{n}$ nodes can be solved in time $O^*(2^{(1-\delta)\tilde{n}})$. This holds even when in Δ-Set Cover, all sets are of the same size and every optimal solution is of size $O(\varepsilon n/\log n)$ and consists of disjoint sets.*

[1] T need not be an arborescence, only its underlying undirected graph is a tree.

[2] This could be done in the following way. Define the host graph G' to be G with edges in both directions, and direct the edges in T away from an arbitrary vertex $v \in T$ to create the directed tree T', which is thus an arborescence. Clearly, the directed instance is a yes-instance if and only if the undirected instance also is.

We can also show that even moderate improvements to the fastest known running time for Δ-Set Cover, namely, to the $O^*(2^{(1-1/2\Delta)n})$ time algorithm of [22], implies improvements for Directed nTree and for Directed Hamiltonicity (Section 4).

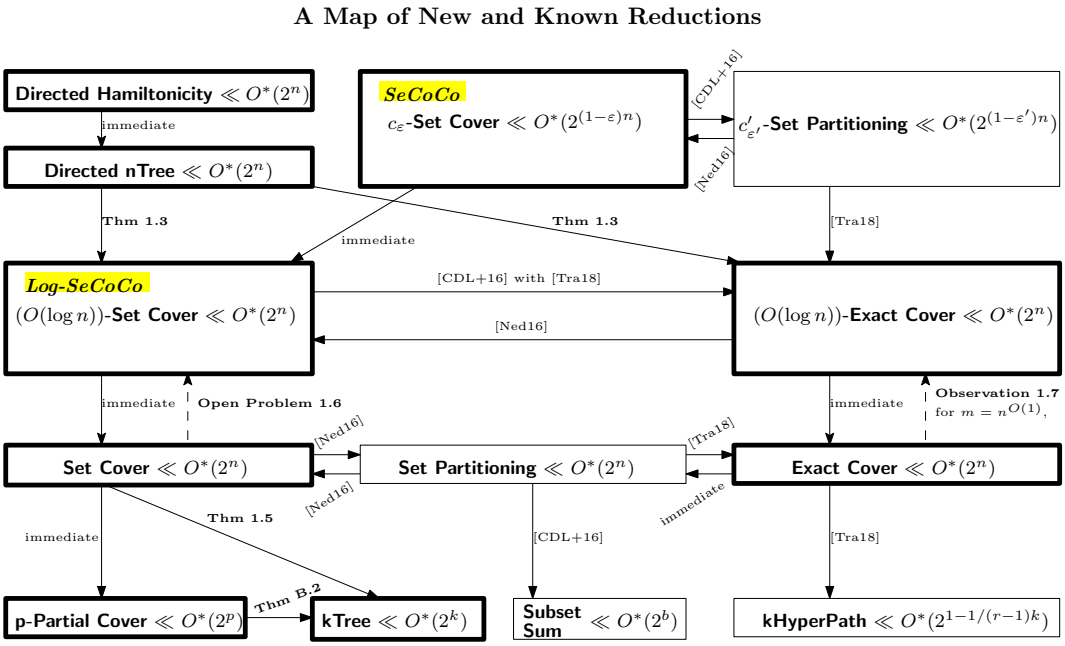


Figure 1 An arrow from a box with $A \ll O^*(2^{n_A})$ to $B \ll O^*(2^{n_B})$ represents a reduction from problem A to problem B, such that if B can be solved in time $O^*(2^{(1-\varepsilon)n_B})$ then A can be solved in time $O^*(2^{(1-\varepsilon'(\varepsilon))n_A})$. We denote by b the number of bits required to represent the integers in Subset Sum, and by r the uniformity parameter in kHyperPath. The problems we focus on are drawn in thick frames.

Our next result, whose proof appears in Section 3, shows that the $2^k\mathsf{poly}(n)$ running time of kTree by [23] is actually optimal (up to exponential improvements) even when considering the undirected version, assuming SeCoCo or even weaker hypotheses such as Log-SeCoCo.

▶ **Theorem 1.5.** *If for some fixed $\varepsilon > 0$, kTree can be solved in time $O^*((2-\varepsilon)^k)$, then for some $\delta(\varepsilon) > 0$, Set Cover on n elements and m sets can be solved in time $O^*((2-\delta)^n)$.*

In fact, our reduction also works from the more general p-Partial Cover problem, whose input is similar to the Set Cover problem but with an additional integer p, and the goal is to find the smallest sub-collection of sets whose union contains at least p elements (rather than all elements). For simplicity, we first present the reduction from Set Cover to kTree (Section 3), and then we show how to adjust it to be from p-Partial Cover (Subsection 3.1).

Discussion

Our first result (Theorem 1.3) supports the validity of Log-SeCoCo based on the Directed nTree problem, which we believe does not admit an $O^*(2^{(1-\varepsilon)n})$-time algorithm, for two reasons. First, this problem includes the well-known Directed Hamiltonicity problem, and in the last 50 years no algorithm significantly faster than $O^*(2^n)$-time was found for it, despite extensive efforts [4, 17, 5, 33] and in contrast to progress on its undirected version [6]. Second, for a generalization of nTree and kTree variants, namely, for Subgraph Isomorphism where the

pattern is an arbitrary graph of arbitrary size, a time lower bound $n^{\Omega(n)}$ is known assuming ETH [13], even when the host and pattern graphs have the same number of nodes. We see it as evidence that also Directed kTree does not become easier as the size k of the pattern graph increases all the way to $k = n$, which would imply that the conditional lower bound in Theorem 1.5 which shows that kTree cannot be solved in time $O^*(2^{(1-\varepsilon)k})$, extends to $k = n$. If true, then by our results, solving Set Cover significantly faster than $O^*(2^n)$-time is equivalent to achieving the same running time in the special case of Δ-Set Cover with $\Delta = O(\log n)$, which can be seen as an analogue to the SETH sparsification lemma [19]. Another interesting consequence of our results is that if kTree can be solved significantly faster than $O^*(2^k)$ than Directed nTree can be solved significantly faster than $O^*(2^n)$. Such a reduction from a directed problem to its undirected version is not obvious, even when the latter has extra freedom in the form of parameterization. A potentially interesting conclusion from the special instances of Δ-Set Cover produced in Theorem 1.4, where the goal could be stated as finding a sub-collection of disjoint sets that covers the entire ground set, which we call Exact Cover, is that Directed Hamiltonicity could be more closely related to Exact Cover than to Set Cover. This is despite the fact that Set Cover and Exact Cover were shown to be equivalent with respect to solvability in $O^*(2^{(1-\varepsilon)n})$ time [30, 32], as there is an exponential blowup in the number of sets in the reduction from Set Cover to Exact Cover. As we observe below, Exact Cover with polynomially many sets can indeed be solved significantly faster than $O^*(2^n)$. See Figure 1 for an overview of new and known reductions, where problem A being drawn above problem B implies that there is a path, and a reduction, from A to B. The following open problem formalizes the foregoing discussion.

▶ **Open Problem 1.6.** *Does an $O^*(2^{(1-\varepsilon)n})$-time algorithm for Δ-Set Cover with $\Delta = O(\log n)$ imply an $O^*(2^{(1-\varepsilon'(\varepsilon))n})$-time algorithm for Set Cover?*

Perhaps surprisingly, we can resolve the Exact Cover analogue of Open Problem 1.6 in the special but common case $m = n^{O(1)}$, as follows. Here, $O(c \log n)$-Exact Cover is Exact Cover with sets of size bounded by $O(c \log n)$.

▶ **Observation 1.7.** *If for some fixed $\varepsilon > 0$ and $c > 0$, $O(c \log n)$-Exact Cover can be solved in time $O^*(2^{(1-\varepsilon)n})$, then for some $\delta(\varepsilon) > 0$, Exact Cover with $m = O(n^c)$ can be solved in time $O^*(2^{(1-\delta)n})$.*

To see this, simply guess which sets of size larger than Δ participate in an optimal solution, using an exhaustive search over at most $n \cdot \binom{m}{n/\Delta}$ choices, and then apply the assumed algorithm for the remaining sets.

We note that the results can be easily generalized to weighted Directed Hamiltonicity (i.e., TSP) and Directed nTree by using a generalized conjecture about the weighted version of Set Cover, whose input is similar to the Set Cover only with a positive weight for each set, and the goal is to find a minimum-weight sub-collection whose union is the entire ground set. The generalized conjecture then states that for every fixed $\varepsilon > 0$, weighted Set Cover with the cardinality of every set bounded by $O(1/\varepsilon \cdot \log n)$ cannot be solved in time $O^*(2^{(1-\varepsilon)n})$.

Prior Work

Relevant state-of-the-art algorithms to Set Cover and Subgraph Isomorphism variants are as follows. Set Cover can be solved in time $(m + 2^n)\mathsf{poly}(n)$ [9], which for $m = n^{\omega(1)}$ is faster than the aforementioned $O(mn2^n)$ algorithm of [15]. The case where all sets are of size q and the goal is to determine whether p pairwise-disjoint sets can be packed, can be solved in time $O^*(2^{(1-\varepsilon)pq})$ for $\varepsilon(q) > 0$ [10]. Determining whether a Set Cover instance has a solution

of size at most σn can be done in time $O^*(2^{(1-\Omega(\sigma^4))n})$ [30]. The fastest known running time for Directed Hamiltonicity is $O^*(2^{n-\Theta(\sqrt{n/\log n})})$ [7]. Finally, several problems, including Directed Hamiltonicity and Set Cover, were shown to belong to the class EPNL, defined as all problems that can be solved by a non-deterministic turing machine with space $n + O(\log n)$ bits [20].

Techniques

To demonstrate our basic technique for Theorems 1.3 and 1.4, let us present an extremely simple reduction from Directed Hamiltonicity to Δ-Set Cover with $\Delta = O(\log n)$. Given a directed graph G, first guess (by exhaustive search) a relatively small set of nodes ("representatives"), and an ordering for them $z_1, z_2, \ldots$ in a potential Hamiltonian cycle. Then construct a Set Cover instance whose ground set is the nodes of G and has the following sets: for every possible path of length Δ in G from some z_i to z_{i+1} that does not visit any representative in between, there is a set that contains all the nodes in this path except for z_{i+1}. A Hamiltonian cycle in G clearly corresponds to a set cover using exactly n/Δ sets, and vice versa. The main challenge we deal with when reducing from the more general Directed nTree is that the pattern tree does not decompose easily into appropriate subgraphs.

The intuition for Theorem 1.5 is as follows. In the reduction from Set Cover to kTree we first guess a partition of n (the number of elements) that represents how an optimal solution covers the elements, by exhaustive search over $2^{O(\sqrt{n})}$ unordered partitions of n. Then, we represent the Set Cover instance using a Subgraph Isomorphism instance, whose pattern tree T succinctly reflects the guessed partition of n, and the idea is that this tree is isomorphic to a subgraph of the Set Cover graph if and only if the Set Cover instance has a solution that agrees with our guess. The main difficulty here is that we reduce to the undirected version of kTree, and thus additional attention is required to make the tree fit only in specific locations in the host graph.

2 Reduction from Directed nTree to Set Cover

In this section we prove Theorem 1.3. The heart of the proof is actually the following lemma.

▶ **Lemma 2.1.** *Directed nTree on $\tilde{n}$ nodes can be reduced, for every $\Delta \in [\tilde{n}]$, to $O(\tilde{n}^{9\tilde{n}/\Delta})$ instances of Δ-Set Cover, each with $n \leq \tilde{n} + 9\tilde{n}/\Delta$ elements, in time $O(\tilde{n}^{\Delta+1} + \tilde{n}^{9\tilde{n}/\Delta})$.*

Proof of Theorem 1.3. Assume there is an algorithm for Δ-Set Cover on n elements and $\Delta = O(1/\varepsilon \cdot \log n)$ that runs in time $O^*(2^{(1-\varepsilon)n})$. Given an instance of Directed nTree on $\tilde{n}$ nodes, apply Lemma 2.1 with

$$\Delta = 81/\varepsilon \cdot \log \tilde{n} = O(1/\varepsilon \cdot \log n),$$

and then solve each of the resulting $O(\tilde{n}^{9\tilde{n}/\Delta}) = O^*(2^{\varepsilon\tilde{n}/9})$ instances of Δ-Set Cover, using the assumed algorithm, in time

$$O^*(2^{(1-\varepsilon)n}) \leq O^*(2^{(1-\varepsilon)(\tilde{n}+9\tilde{n}/(81/\varepsilon\cdot\log\tilde{n}))}) \leq O^*(2^{(1-\varepsilon)(\tilde{n}+\varepsilon\tilde{n}/(9\cdot\log\tilde{n}))}).$$

The total running time is

$$O^*(2^{81/\varepsilon\cdot\log^2\tilde{n}+\log\tilde{n}+\varepsilon\tilde{n}/9+(1-\varepsilon)(\tilde{n}+\varepsilon\tilde{n}/(9\cdot\log\tilde{n}))}) \leq O^*(2^{\tilde{n}-\varepsilon\tilde{n}/2}),$$

which concludes the proof for $\delta(\varepsilon) = \varepsilon/2$. ◀

It remains to prove Lemma 2.1, and we start with an overview of this proof. Consider an instance (G, T) of Directed nTree, and for this overview, assume that the tree T is rooted at some node r, and all edges are directed away from it. The idea is to create roughly $\tilde{n}^{9\tilde{n}/\Delta}$ instances of Δ-Set Cover on $n \leq \tilde{n} + 9\tilde{n}/\Delta$ elements each, such that at least one of them has a solution of size $t \leq 9\tilde{n}/\Delta$ if and only if the instance (G, T) has a solution. The first step is to cover the tree T with t small subtrees, each of size at most Δ, such that the union of their node sets is T and they may intersect only at their roots (the root of a subtree is the node closest to r). Then guess, by enumerating over all possible choices, how the solution to (G, T) maps the root of each subtree to a node in G, and create a corresponding an instance of Δ-Set Cover. For every such instance, perform an inner enumeration to further guess, what is the (unordered) set of nodes in G that each subtree is mapped to, and add a corresponding set to the Δ-Set Cover instance, but only if this guess does not violate the local and global structure of T. That is, taking into account the edges within and between the subtrees, by testing whether the set can be an isomorphic copy of the subtree, testing for the edges between roots, respectively. For the correctness, we need to show that a solution of size t to the Δ-Set Cover instance implies a one-to-one correspondence between the t sets and the roots of the subtrees, and hence a copy of T in G. The general case where the edges of T are orientated arbitrarily is similar, except that the edge orientations are taken into account when comparing subtrees but not when computing a cover of T by small subtrees.

We proceed to the algorithm that computes the aforementioned cover of T by small subtrees. This algorithm traverses the tree using DFS and add subtrees to the cover whenever the DFS accumulates enough nodes, see Algorithm 1 for full details. Its output is a set S, where each $s \in S$ is a connected subset of the nodes of T, and thus we can refer to each such s as a subtree of T, and let $r(s)$ denote its root, i.e., its node that is closest to r in T. The following lemma describes the guarantees of this algorithm and will be later used to prove Lemma 2.1.

▶ **Lemma 2.2.** *Given a tree T with root r on $\tilde{n}$ nodes and an integer $l \leq \tilde{n}$, Algorithm 1 finds in polynomial time a collection S of subtrees of T such that:*

a. *the number of nodes in each subtree is at most $2(l-1)$;*
b. *every node in T is in some subtree;*
c. *two subtrees in S may only intersect in their roots; and*
d. *the number of subtrees is $|S| \leq \frac{3\tilde{n}}{l-1}$.*

Proof of Lemma 2.2. We first show that items (a)–(c) are satisfied by the output of Algorithm 1. Since in the worst case Algorithm 1 adds a subtree in the first time the accumulated number of nodes exceeds l, the number of nodes of each subtree is bounded by $2(l-1)$. In addition, every node v appears in some subtree, since at some point during the DFS it will be the child, and then it will be passed up the tree and eventually added to S. To see why the last requirement holds, observe that whenever an accumulated set is passed up the tree and encounters an existing root, this set will be added to S.

To prove item (d), denote denote by S_{big} the collection of sets in S of size at least l (added in line 6), and by S_{sml} the collection of sets in S of size smaller than l (added in lines 11 and 13). A set $s \in S_{\text{sml}}$ was created only if $r(s)$ at the time of its creation was the root of at least one (other) set in S_{big} (line 11) or was the last traversed node in the DFS (line 13). Together with the fact that each root has at most one set from S_{sml}, we conclude that each set $s \in S_{\text{sml}}$ excluding at most one, can be associated with a distinct set in S_{big}, one that contains $r(s)$. Hence, $|S_{\text{sml}}| - 1 \leq |S_{\text{big}}|$. The big sets have size at least l, and

Algorithm 1

Input: tree T rooted at r and size parameter $l \in [n]$
Output: cover S of T by subtrees of size at most $2(l-1)$

$S \leftarrow \emptyset$
for all $v \in V$ **do** $s(u) \leftarrow \{u\}$
traverse T using a DFS from r, and whenever returning from a node v to its parent p in T, do the following:
    let $s(p) \leftarrow s(p) \cup s(v)$
    if $|s(p)| \geq l$ **then**
        add $s(p)$ to S
        if p has unvisited children **then**
            let $s(p) \leftarrow \{p\}$
        else let $s(p) \leftarrow \emptyset$
    else if p has no unvisited children and $p \in s$ for some $s \in S$ **then**
        add $s(p)$ to S and let $s(p) \leftarrow \emptyset$
    else if p is the last node traversed in the tree **then**
        add $s(p)$ to S
return S

except for their roots they have distinct vertices, hence $|S_{\text{big}}| \leq \frac{\tilde{n}}{l-1}$. We conclude that

$$|S| = |S_{\text{sml}}| + |S_{\text{big}}| \leq 2|S_{\text{big}}| + 1 \leq \frac{2\tilde{n}}{l-1} + 1 \leq \frac{3\tilde{n}}{l-1},$$

which completes the proof of Lemma 2.2. ◀

Proof of Lemma 2.1. We describe the reduction in stages.

- Apply the aforementioned Algorithm 1 for partition T into subtrees that satisfy the conditions in Lemma 2.2. By picking $l = \Delta/3 + 1$, we obtain that each set is bounded by Δ and that $|S| \leq 9\tilde{n}/\Delta$. Hence, the cardinality of $R := \{r(s)\}_{s \in S}$ is bounded by $9\tilde{n}/\Delta$. For S returned by Algorithm 1, let $R_T = \{r(s) : s \in S\}$ (note that $|R_T|$ may be smaller than $|S|$).
- Then, guess $|R_T|$ nodes in G that will function as the image of the nodes in R_T in a potential subgraph isomorphism function and denote them by R_G, and then guess a bijection f from R_T to R_G. The guessing is done by exhaustive search over $\binom{\tilde{n}}{|R_T|}$ choices of nodes, and together with the number of ways to choose a bijection it can be done in time $\binom{\tilde{n}}{|R_T|}|R_T|!$.
- Finally, enumerate all sets s' of nodes of size at most Δ in G, and denote by $G(s')$ the graph induced from each on G. For every subtree $s \in S$, look by brute force for an isomorphic copy of s in subgraphs $G(s')$ that contain $f(r(s))$ as a root and no other node in R_G, and that satisfy $|s'| = |s|$. For each one that was found, add to the constructed Set Cover instance a set ${s'}_G$ with the root r' labeled ${r'}_s$ where s corresponds to the subtree s of T whose copy found to be in $G(s')$. Note that the number of elements in the Set Cover instance is exactly $\tilde{n} - |R_T| + |S|$, and that the time spent per each subgraph isomorphism test is at most $|s|! \leq \Delta!$, and thus the total time spent in this step is $|S|\binom{\tilde{n}}{\Delta}\Delta! = |S|\tilde{n} \cdot (\tilde{n}-1) \cdots (\tilde{n} - \Delta + 1) \leq 9\tilde{n}/\Delta \cdot \tilde{n}^{\Delta} \leq \tilde{n}^{\Delta+1}$.

Now we show that the size constraints follow. As $|R_T| \leq 9\tilde{n}/\Delta$, similar to before, the number of Set Cover instances is bounded by

$$\binom{\tilde{n}}{9\tilde{n}/\Delta}(9\tilde{n}/\Delta)! = \tilde{n} \cdot (\tilde{n}-1) \cdots (\tilde{n}-9\tilde{n}/\Delta+1) \leq \tilde{n}^{9\tilde{n}/\Delta}$$

as required.

We now prove that at least one of the Set Cover instances has solution of size at most $|S|$ (in fact exactly $|S|$ as no smaller solutions available) if and only if the Directed nTree instance is a yes instance. For the first direction, assume that the Directed nTree instance is a yes instance. Considering the isomorphic copy of T in G, its $|S|$ subtrees as Algorithm 1 outputs on T will be sets in the Set Cover instance the reduction outputs, and so it has a solution of size at most $|S|$. For the second direction, if a Set Cover instance has a solution I of size at most $|S|$ and since the number of labeled roots is $|S|$, it must be that for each subtree $s \in S$ its labeled root is in exactly one set in I, and so $|I| = |S|$. Since I is a legal solution and S covers all the nodes, no node in $V(G) \setminus R_G$ appears twice in I. The conclusion is that these sets together form the required tree, concluding the proof of Lemma 2.1. ◀

We note that in the case of Theorem 1.4 for Directed Hamiltonicity, we do not have to use Algorithm 1, but simply guess n/Δ representative nodes in G and their ordering in the potential cycle, and then enumerate all paths of size Δ to represent paths between consecutive representatives. Hence we obtain a Δ-Set Cover instance with the additional constraints of Theorem 1.4.

3 Reduction from Set Cover to kTree

In this section we prove Theorem 1.5. In order to make the proof simpler, we will have an assumption regarding the Set Cover instance, as follows. For a constant $g > 0$ to be determined later, we can assume that all the sets in the Set Cover instance are of size at most n/g^2, as otherwise such instance can already be solved significantly faster than $O^*(2^n)$, proving the theorem in a degenerate manner. We formalize it as follows.

▶ **Assumption 3.1.** *All the sets in the Set Cover instance are of size at most n/g^2.*

To justify this assumption, notice that one can remove all sets of size more than n/g^2 from the Set Cover instance. Indeed, if some optimal solution for the Set Cover instance contains a set of size at least n/g^2, such optimal solution can be found by simply guessing one set of at least this size (using exhaustive search over at most m choices) and then applying the known dynamic programming algorithm on the still uncovered elements (at most $n - n/g^2$ of them), and return the optimal solution in total time $O^*(2^{(1-1/g^2)n})$. We continue to the following lemma, which is the heart of the proof.

▶ **Lemma 3.2.** *For every fixed $\varepsilon > 0$, Set Cover on a ground set $N = [n]$ and a collection M of m sets that satisfies assumption 3.1, can be reduced to $2^{O(\sqrt{n})}$ instances of kTree with $k = (1+\varepsilon)n + O(1)$.*

We will use this lemma to prove Theorem 1.5, the proof of Lemma 3.2 will be given after.

▶ **Theorem 1.5** (*restated*). *If for some fixed $\varepsilon > 0$, kTree can be solved in time $O^*((2-\varepsilon)^k)$, then for some $\delta(\varepsilon) > 0$, Set Cover on n elements and m sets can be solved in time $O^*((2-\delta)^n)$.*

Proof of Theorem 1.5. Assume that for some $\varepsilon' \in (0,1)$, kTree can be solved in time $O^*((2-\varepsilon')^k) \leq O^*(2^{(1-\varepsilon'/2)k})$. We reduce the Set Cover instance by applying Lemma 3.2 with $\varepsilon = \varepsilon'/4$, and then solve each of the $2^{c_1\sqrt{n}}$ instances of kTree in the assumed time of $O^*(2^{(1-\varepsilon'/2)((1+\varepsilon)n+c_2)})$, where $c_1, c_2 > 0$ are the constants implicit in the terms $2^{O(\sqrt{n})}$ and $O(1)$ in the lemma, respectively. The total running time is $O^*(2^{(1-\varepsilon'/2)(1+\varepsilon)n+c_1\sqrt{n}}) = O^*(2^{(1-\varepsilon'/4-\varepsilon'^2/8)n+c_1\sqrt{n}}) \leq O^*(2^{(1-\varepsilon'/4)n}) \leq O^*((2-\varepsilon'/4)^n)$, which concludes the proof for $\delta(\varepsilon') = \varepsilon'/4$. ◀

To outline the proof of Lemma 3.2, we will need the following definition. For an integer $a > 0$, let $p(a)$ be the set of all unordered partitions of a, where a *partition* of a is a way of writing a as a sum of positive integers, and *unordered* means that the order of the summands is insignificant. The asymptotic behaviour of $|p(a)|$ (as a tends to infinity) is known [16] to be

$$e^{\pi\sqrt{2a/3}}/(4a\sqrt{3}) = 2^{O(\sqrt{a})}.$$

It is possible to enumerate all the partitions of a with constant delay between two consecutive partitions, exclusive of the output [31, Chapter 9].

Now the intuition for our reduction of Set Cover to kTree is to first guess a partition of n (the number of elements) that represents how an optimal solution covers the elements, as follows. Associate each element arbitrarily with one of the sets that contain it (so in effect, we assume each element is covered only once) and count how many elements are covered by each set in the optimal solution. This guessing is done by exhaustive search over $p(n) \leq 2^{O(\sqrt{n})}$ partitions of n. Then, we represent the Set Cover instance using a Subgraph Isomorphism instance, whose pattern tree T succinctly reflects the guessed partition of n. The idea is that the tree is isomorphic to a subgraph of the Set Cover graph if and only if the Set Cover instance has a solution that agrees with our guess.

▶ **Lemma 3.2** (*restated*). *For every fixed $\varepsilon > 0$, Set Cover on a ground set $N = [n]$ and a collection M of m sets that satisfies assumptions 3.1, can be reduced to $2^{O(\sqrt{n})}$ instances of kTree with $k = (1+\varepsilon)n + O(1)$.*

Proof of Lemma 3.2. Given a Set Cover instance on n elements $N = \{n_i : i \in [n]\}$ and m sets $M = \{S_i\}_{i\in[m]}$ and an $\varepsilon > 0$, construct $2^{O(\sqrt{n})}$ instances of kTree as follows. For a constant $g(\varepsilon)$ to be determined later, the host graph $G_g = (V_g, E_g)$ is the same for all the instances, and is built on the bipartite graph representation of the Set Cover instance, with some additions. This is done in a way that a constructed tree will fit in G_g if and only if the Set Cover instance has a solution that corresponds to the structure of the tree, as follows (see Figure 2). The set of nodes is $V_g = N \cup M \cup M_g \cup R \cup \{r_g, r_1, r_2, r\}$, where $M_g = \{X \subseteq M : |X| = g\}$ and $R = \{v^i_j : i \in [4], j \in [n/(g/2)]\}$. Intuitively, the role of M_g is to keep the size of the trees small by representing multiple vertices in M (multiple sets in Set Cover) at once as the "powering" technique for Set Cover done in [12][3], and the role of R and $\{r_g, r_1, r_2, r\}$ is to enforce that the trees the reduction constructs will fit only in certain ways.

The set of edges is constructed as follows. Edges between N and M are the usual bipartite graph representation of Set Cover (i.e., connect vertices $n_j \in N$ and $S_i \in M$ whenever $n_j \in S_i$). Also, connect vertex $X \in M_g$ to vertex $n_j \in N$ if at least one of the sets

[3] We can slightly simplify this step in the construction by using the equivalence from [12] between solving Set Cover in time $O^*(2^{(1-\varepsilon)n})$ and in time $O^*(2^{(1-\varepsilon')(n+t)})$ where t is the solution size. However, we opted to reduce directly from Set Cover for compatibility with our parameters and for sake of generality.

in X contains n_j. Additionally, add edges between r_g and every vertex in M_g, and $v_j^4 \in R$ for $j \in [n/(g/2)]$, between r_i and v_j^i for every $i \in \{1,2\}$ and $j \in [n/(g/2)]$, and finally between r and every vertex $v \in \{r_g, r_1, r_2\}$, $S_i \in M$, and $v_j^3 \in R$ for $j \in [n/(g/2)]$.

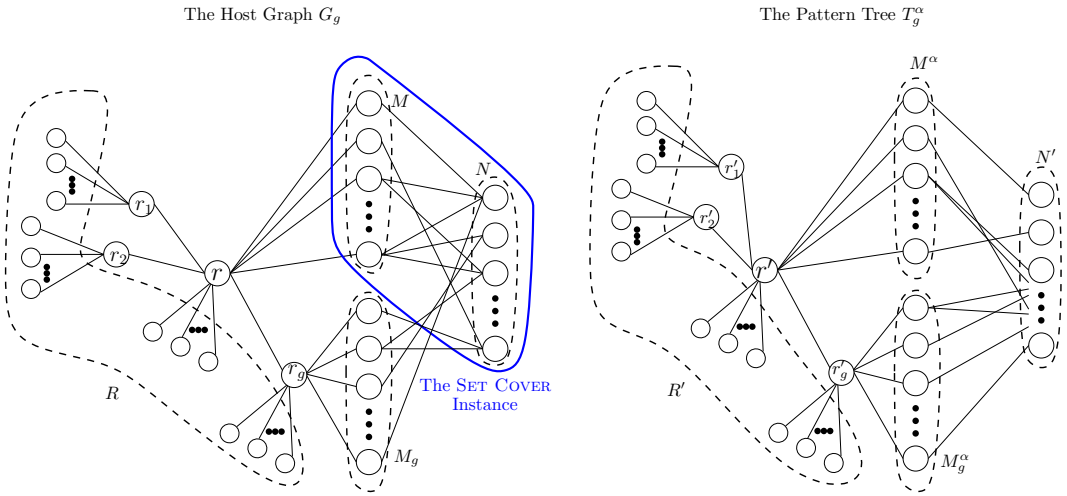


Figure 2 An illustration of part of the reduction. The Set Cover instance is depicted in blue, and sets of vertices are indicated by dashed curves.

Next, construct $2^{O(\sqrt{n})}$ trees such that identifying those that are isomorphic to a subgraph of G_g will determine the optimum of the Set Cover instance.

For every partition $\alpha = (p_1, p_2, ..., p_l) \in p(n)$ (with possible repetitions) where $p(n)$ is as defined above, construct a tree $T_g^\alpha = (V_g^\alpha, E_g^\alpha)$. This tree has the same set of edges and vertices as G_g, except for the vertices in $M \cup M_g$ and the edges incident to them, which are replaced by a set of new vertices $M^\alpha \cup M_g^\alpha$, and these new vertices are connected to the rest in a way that the resulting graph is a tree. In more detail, $V_g^\alpha = N' \cup M^\alpha \cup M_g^\alpha \cup R' \cup \{r'_g, r'_1, r'_2, r'\}$ where $N', R', r'_g, r'_1, r'_2, r'$ are tagged copies of the originals, and M^α, M_g^α are initialized to be $\emptyset$.

We define α_g to be a partition of n which is also a shrinked representation of α by partitioning α into sums of g numbers for a total of $\lfloor l/g \rfloor$ such sums, and a remaining of less than g numbers. Formally,

$$\alpha_g = (\sum_{i=1}^{g} p_i, \sum_{i=g+1}^{2g} p_i, ..., \sum_{i=(g-1)\cdot\lfloor l/g \rfloor + 1}^{g\lfloor l/g \rfloor} p_i, p_{g\lfloor l/g \rfloor + 1}, ..., p_l)$$

Note that all the numbers in α_g are a sum of g numbers in α, except (maybe) for the last $g' := l - g\lfloor l/g \rfloor < g$ numbers in α_g, a (multi)set which we denote $s(\alpha_g)$. For every $i \in \alpha_g$ (with possible repetitions), add a star on $i+1$ vertices to the constructed tree T_g^α. If $i \in \alpha_g \setminus s(\alpha_g)$, add the center vertex to M_g^α, connect it to r'_g, and add the rest i vertices to N'. Else, if $i \in s(\alpha_g)$, add the center vertex to M^α, connect it to r', and again add the rest i vertices to N'. Return the minimum cardinality of α for which (G_g, T_g^α) is a yes-instance. To see that this construction is small enough, note that the size of G_g is at most $4 + 4 \cdot n/(g/2) + m^g + m + n$ which is polynomial in m, and the size of the tree T_g^α is at most

$$4 + 4 \cdot n/(g/2) + n/g + g + n = n \cdot (1 + 9/g) + O(1) = n \cdot (1 + \varepsilon) + O(1)$$

where the last equality holds for $g = 9/\varepsilon$, and so the size constraint follows.

We now prove that at least one of the trees T_g^α returns yes and satisfies $|\alpha| \leq d$, if and only if the Set Cover instance has a solution of size at most d. For the first direction, assume that the Set Cover instance has a solution I with $|I| \leq d$. Consider a partition $\alpha_I \in p(n)$ of n that corresponds to I in the following way. Associate every element with exactly one of the sets in I that contains it, and then consider the list of sizes of the sets in I according to this association (eliminating zeroes). Clearly, $(G_g, T_g^{\alpha_I})$ is a yes-instance and so the reduction will return a number that is at most $|I|$.

For the second direction, assume that every solution to the Set Cover instance is of size at least $d+1$. We need to prove that for every tree T_g^α with $|\alpha| \leq d$, (G_g, T_g^α) is a no-instance. Assume for the contrary that there exists such α for which (G_g, T_g^α) is a yes-instance with the isomorphism function f from T_g^α to G_g. We will show that the only way f is feasible is if $f(r') = r$, $f(M^\alpha) \subseteq M$, $f(M_g^\alpha) \subseteq M_g$, and also $f(N') = N$, which together allows us to extract a corresponding solution for the Set Cover instance, leading to a contradiction. We start with the vertex $r' \in T_g^\alpha$. Since its degree is at least $n/(g/2)+3$ and by Assumption 3.1 and the construction of G_g, it holds that $f(r') \notin \{r_1, r_2\} \cup R \cup M \cup M_g$. Moreover, if it was the case that $f(r') \in \{r_g\} \cup N$ then $\{f(r'_1), f(r'_2)\} \cap (M \cup M_g) \neq \emptyset$, however, the degree of r'_1 and r'_2 in T_g^α is $n/(g/2)$, and the degree of the vertices in $M \cup M_g$ in G_g is at most $g \cdot n/g^2 = n/g$, so it must be that $f(r) = r$. Our next claim is that $f(r'_g) = r_g$. Observe that Assumption 3.1 implies that every solution for the Set Cover instance is of size at least g^2 and so $M_g^\alpha \neq \emptyset$, which means r'_g in the tree has vertices in distance 2 from it and away from r', a structural constraint that cannot be satisfied by any vertex in $\{r_1, r_2\} \cup R$. Furthermore, the degree of r'_g is at least $n/(g/2)$ and so again by Assumption 3.1 it is also impossible that $f(r'_g) \in M^\alpha$, and hence it must be that $f(r'_g) = r_g$. Finally, by the same Assumption and the degrees of r_1 and r_2, $f(r'_1)$ and $f(r'_2)$ must be in $\{r_1, r_2\}$. Altogether, it must be that $f(M_g^\alpha) \subseteq M_g$, $f(M^\alpha) \subseteq M$ and that $f(N') = N$, and therefore it is possible to extract a feasible solution to the Set Cover instance that has at most d sets in it, which is a contradiction, concluding the proof of Lemma 3.2.

3.1 Reduction from p-Partial Cover

In this subsection we show that Theorem 1.5 is correct also assuming a weaker conjecture, that p-Partial Cover cannot be solved significantly faster than $O^*(2^p)$. Notice that p-Partial Cover can be solved in time $O^*(2^p)$ by a simple application of the method in [23], as pointed out to us by Cornelius Brand and anonymous referees. We now reduce from p-Partial Cover to Directed kTree by following Lemma 3.2 with the following adjustments.

Instead of enumerating over all the partitions of n, do it only for p and hence the number of partitions is $2^{O(\sqrt{p})}$ with each partition α inducing a tree T_g^α in a similar way to Lemma 3.2, of size at most $2p/g + p$. Note that Assumption 3.1 adjusted to the p-Partial Cover case hold also here, since it is possible to use the $O^*(2^p)$-time algorithm for p-Partial Cover mentioned above after removing large sets of size $\geq p/g^2$. From here onwards, the proof of correctness is similar to Lemma 3.2, and thus we omit it. Regarding running time, assume that for some $\varepsilon' \in (0,1)$, kTree can be solved in time $O^*((2-\varepsilon')^k) \leq O^*(2^{(1-\varepsilon'/2)k})$. Setting $g = 8/\varepsilon'$ for $\varepsilon' = 64(1 - \log_2(2-\varepsilon))$ (without loss of generality, assume that ε' is small enough), we get a total running time of

$$
\begin{aligned}
&O(m^{c_1 g} 2^{p - p/g^2} + 2^{(p+2p/g)(1-\varepsilon'/2)+c_2\sqrt{p}} \cdot m^{c_3 g}) \\
&\qquad = O(m^{c_1 8/\varepsilon'} 2^{p - \varepsilon' p/64} + 2^{p+\varepsilon'/4 \cdot p - \varepsilon'/2 \cdot p - \varepsilon'^2/8 \cdot p + c_1\sqrt{p}} \cdot m^{c_3 4/\varepsilon'}) \\
&\qquad \leq O(2^{(1-\varepsilon'/64)p} \cdot m^{c_1 4/\varepsilon'}) \\
&\qquad \leq O((2-\varepsilon)^p \cdot m^{c_1 4/\varepsilon}),
\end{aligned}
$$

where c_1 is the constant derived from the method of [23], c_2 is the constant implicit in the term $2^{O(\sqrt{p})}$, and c_3 is the constant in the exponent of m implicit in the term $O^*(2^{(1-)p})$, as required.

▶ **Lemma 3.3.** *For every fixed $\varepsilon > 0$, p-Partial Cover on a ground set $N = [n]$ and a collection M of m sets can be reduced to $2^{O(\sqrt{p})}$ instances of kTree with $k = (1+\varepsilon)p + O(1)$.*

We thus proved the following theorem.

▶ **Theorem 3.4.** *If for some fixed $\varepsilon > 0$, kTree can be solved in time $O^*((2-\varepsilon)^k)$, then for some $\delta(\varepsilon) > 0$, p-Partial Cover on n elements and m sets can be solved in time $O^*((2-\delta)^p)$.*

4 Moderate Improvements to Δ-Set Cover Imply New Algorithms for Directed nTree and Directed Hamiltonicity

In this section we show how moderate improvements for variants of Set Cover imply new algorithms for Directed nTree. Given any algorithm for Δ-Set Cover with runtime $f(n,m,\Delta)$, by Lemma 2.1 Directed nTree admits an algorithm with running time $O(\tilde{n}^{\Delta} + \tilde{n}^{\tilde{n}/\Delta} f(n,m,\Delta))$. We now demonstrate how this algorithm behaves with different regimes of Δ.

If there exists $\varepsilon > 0$ such that for every $\Delta = \mathsf{poly}(\log n)$, $f(n,m,\Delta) = O^*(2^{(1-1/\Delta^{1-\varepsilon})n})$ then by considering $\Delta = \log^{(1+\varepsilon')/\varepsilon} n = \mathsf{poly}(\log n)$ for $\varepsilon' > 0$, Directed nTree has an algorithm with runtime

$$O(2^{\log^{(1+\varepsilon')/\varepsilon+1}\tilde{n}}) + O^*(2^{\tilde{n}/\log^{(1+\varepsilon')/\varepsilon-1}\tilde{n}} \cdot 2^{(1-1/(\log^{(1+\varepsilon')/\varepsilon}\tilde{n})^{1-\varepsilon})\tilde{n}})$$
$$= O^*(2^{(1-1/(\log^{(1+\varepsilon')/\varepsilon-2}\tilde{n}))\tilde{n}})$$

Considering larger regimes, if for some fixed $\varepsilon > 0$, $\delta \in (0, 1/2)$, and $\Delta = O(n^{\delta})$, $f(n,m,\Delta) = O^*(2^{(1-\frac{(1+\varepsilon)\log\Delta}{\delta\Delta})n})$ then Directed nTree can be solved in time

$$2^{\tilde{n}^{\delta}\log\tilde{n}} + 2^{\tilde{n}^{1-\delta}\log\tilde{n}} \cdot O^*(2^{(1-\frac{(1+\varepsilon)\log\Delta}{\delta\Delta})\tilde{n}}) = O^*(2^{(1-\varepsilon/\tilde{n}^{\delta})\tilde{n}}) = 2^{\tilde{n}-\Theta(\tilde{n}^{1-\delta})}$$

Note that to break the fastest known $2^{\tilde{n}-\Theta(\sqrt{\tilde{n}/\log\tilde{n}})}$ algorithm for Directed Hamiltonicity by [7], it is enough to have either $f(n,m,\Delta) = O^*(2^{(1-\frac{(2+\varepsilon)\log\Delta}{\Delta})n})$ for $\Delta = n^{1/2-\delta'}$ with every fixed $\delta' > 0$, or $f(n,m,\Delta) = O(m \cdot 2^{(1-\frac{(4+\varepsilon)\log\Delta}{\Delta})n})$ for $\Delta = \sqrt{n}$, taking into account that most algorithms for variants of Set Cover that have the factor m in their runtime, do not have it with higher power than one. ◀

References

1. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-Based Lower Bounds for Subset Sum and Bicriteria Path. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 41–57, 2019. doi:10.1137/1.9781611975482.3.
2. Amir Abboud, Virginia Vassilevska-Williams, and Huacheng Yu. Matching Triangles and Basing Hardness on an Extremely Popular Conjecture. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 41–50. ACM, 2015. doi:10.1145/2746539.2746594.
3. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.
4. Richard Bellman. Combinatorial processes and dynamic programming. In *Combinatorial analysis*, Proceedings of Symposia in Applied Mathematics, pages 217–249. American Mathematical Society, 1960. doi:10.1090/psapm/010.

5 Richard Bellman. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM*, 9(1):61–63, 1962. doi:10.1145/321105.321111.

6 Andreas Bjorklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014. doi:10.1137/110839229.

7 Andreas Björklund. Below All Subsets for Some Permutational Counting Problems . In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:11, 2016. doi:10.4230/LIPIcs.SWAT.2016.17.

8 Andreas Björklund, Dell Holger, and Thore Husfeldt. The Parity of Set Systems under Random Restrictions with Applications to Exponential Time Problems. In *42nd International Colloquium on Automata, Languages and Programming (ICALP 2015)*, volume 9134, pages 231–242. Springer, 2015. doi:10.1007/978-3-662-47672-7_19.

9 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM J. Comput.*, 39(2):546–563, July 2009. doi:10.1137/070683933.

10 Andreas Björklund, Thore Husfeldt, Kaski Ptteri, and Mikko Koivisto. Narrow Sieves for Parameterized Paths and Packings. *Journal of Computer and System Sciences*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

11 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Constrained Multilinear Detection and Generalized Graph Motifs. *Algorithmica*, 74(2):947–967, 2016. doi:10.1007/s00453-015-9981-1.

12 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems As Hard As CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

13 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socała. Tight Bounds for Graph Homomorphism and Subgraph Isomorphism. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1643–1649. SIAM, 2016. doi:10.1137/1.9781611974331.ch112.

14 Anders Dessmark, Andrzej Lingas, and Andrzej Proskurowski. Faster Algorithms for Subgraph Isomorphism of k-Connected Partial k-Trees. *Algorithmica*, 27(3):337–347, January 2000. doi:10.1007/s004530010023.

15 Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (Exponential) Algorithms for the Dominating Set Problem. In *30th International Conference on Graph-Theoretic Concepts in Computer Science*, WG'04, pages 245–256. Springer-Verlag, 2004. doi:10.1007/978-3-540-30559-0_21.

16 Godfrey H. Hardy and Srinivasa Ramanujan. Asymptotic Formulaæ in Combinatory Analysis. *Proceedings of the London Mathematical Society*, s2-17(1):75–115, 1918. doi:10.1112/plms/s2-17.1.75.

17 Michael Held and Richard M. Karp. A Dynamic Programming Approach to Sequencing Problems. In *Proceedings of 16th ACM National Meeting*, ACM '61, pages 71.201–71.204. ACM, 1961. doi:10.1145/800029.808532.

18 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. doi:10.1006/jcss.2000.1727.

19 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.

20 Yoichi Iwata and Yuichi Yoshida. On the Equivalence among Problems of Bounded Width. In *23rd Annual European Symposium on Algorithms (ESA 2015)*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

21 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. The IBM Research Symposia Series. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 Mikko Koivisto. Partitioning into Sets of Bounded Cardinality. In *Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *Lecture Notes in Computer Science*, pages 258–263. Springer-Verlag, 2009. doi:10.1007/978-3-642-11269-0_21.

23 Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, May 2016. doi:10.1145/2885499.

24 Lukasz Kowalik and Juho Lauri. On Finding Rainbow and Colorful Paths. *Theoretical Computer Science*, 628(C):110–114, 2016. doi:10.1016/j.tcs.2016.03.017.

25 R. Krithika, Abhishek Sahu, and Prafullkumar Tale. Dynamic Parameterized Problems. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2016.19.

26 Andrzej Lingas. Subgraph Isomorphism for Biconnected Outerplanar Graphs in Cubic Time. *Theoretical Computer Science*, 63(3):295–302, 1989. doi:10.1016/0304-3975(89)90011-X.

27 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Society, 2009.

28 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 542–553. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

29 Jiří Matoušek and Robin Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1):343–364, 1992. doi:10.1016/0012-365X(92)90687-B.

30 Jesper Nederlof. Finding Large Set Covers Faster via the Representation Method. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.69.

31 Albert Nijenhuis and Herbert S. Will. *Combinatorial Algorithms: For Computers and Hard Calculators*. Academic Press, 2nd edition, 1978.

32 Ohad Trabelsi. Nearly Optimal Time Bounds for kPath in Hypergraphs. *CoRR*, 2018. URL: http://arxiv.org/abs/1803.04940.

33 Gerhard J. Woeginger. Exact Algorithms for NP-hard Problems: A Survey. In Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization - Eureka, You Shrink!*, pages 185–207. Springer-Verlag, 2003. doi:10.1007/3-540-36478-1.

Algorithmic Properties of Sparse Digraphs

Stephan Kreutzer
Technische Universität Berlin, Germany
stephan.kreutzer@tu-berlin.de

Irene Muzi
University of Warsaw, Poland
imuzi@mimuw.edu.pl

Patrice Ossona de Mendez
Centre d'Analyse et de Mathématiques Sociales (CNRS, UMR 8557), Paris, France,
Computer Science Institute of Charles University (IUUK), Prague, Czech Republic
pom@ehess.fr

Roman Rabinovich
Technische Universität Berlin, Germany
roman.rabinovich@tu-berlin.de

Sebastian Siebertz
Humboldt-Universität zu Berlin, Germany
sebastian.siebertz@hu-berlin.de

Abstract

The notions of bounded expansion [56] and nowhere denseness [58], introduced by Nešetřil and Ossona de Mendez as structural measures for undirected graphs, have been applied very successfully in algorithmic graph theory. We study the corresponding notions of directed bounded expansion and nowhere crownfulness on directed graphs, introduced by Kreutzer and Tazari [48]. The classes of directed graphs having those properties are very general classes of sparse directed graphs, as they include, on one hand, all classes of directed graphs whose underlying undirected class has bounded expansion, such as planar, bounded-genus, and H-minor-free graphs, and on the other hand, they also contain classes whose underlying undirected class is not even nowhere dense. We show that many of the algorithmic tools that were developed for undirected bounded expansion classes can, with some care, also be applied in their directed counterparts, and thereby we highlight a rich algorithmic structure theory of directed bounded expansion and nowhere crownful classes.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases Directed graphs, graph algorithms, parameterized complexity, approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.46

Related Version A full version of the paper is available at [44], `https://arxiv.org/abs/1707.01701`.

Funding *Stephan Kreutzer*: Supported by ERC consolidator grant DISTRUCT, No. 648527.
Irene Muzi: Supported by ERC starting grant CUTACOMBS, agreement No. 714704.
Patrice Ossona de Mendez: Supported by grant ERCCZ LL-1201 and by the European Associated Laboratory "Structures in Combinatorics" (LEA STRUCO), and partially supported by ANR project Stint under reference ANR-13-BS02-0007.
Sebastian Siebertz: Supported by the National Science Centre of Poland via POLONEZ grant agreement UMO-2015/19/P/ST6/03998, which has received funding from the European Union's Horizon 2020 research and innovation programme (Marie Skłodowska-Curie grant agreement No. 665778).

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 46; pp. 46:1–46:20

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

1 Introduction

Structural graph theory has made a deep impact on the design of graph algorithms for hard problems. It provides a wealth of different tools for dealing with the intrinsic complexity of NP-hard problems on graphs and these methods have been applied very successfully in *algorithmic graph theory*, in *approximation theory*, *optimisation* and the design of *exact* and *parameterised algorithms* for problems on undirected graphs, see e.g. [11, 14, 16, 15, 17, 18, 28, 29, 66].

Concepts such as *tree width* or *excluded (topological) minors* as well as density based graph parameters such as *bounded expansion* or *nowhere denseness* capture important properties of graphs and make them applicable for algorithmic applications.

The notions of bounded expansion and nowhere denseness were introduced in [56] and [58] to capture structural sparseness of undirected graphs. Classes of bounded expansion are very general and properly generalise, for instance, planar graphs or more generally classes with excluded (topological) minors. But the concept goes far beyond excluded minor classes.

Starting with [56, 58], many algorithmic results for problems on classes of graphs excluding a fixed minor have been extended to the more general case of bounded expansion and nowhere dense classes of graphs, see e.g. [9, 13, 20, 21, 23, 24, 25, 33, 38, 43, 45, 49, 55, 61, 69, 71]. Furthermore, Demaine et al. [19] and Nadara et al. [53] analysed a range of real-world networks and showed that many of them indeed fall within the framework of bounded expansion. This shows that this concept captures many types of real world instances.

An important aspect of classes of bounded expansion and classes which are nowhere dense is that they can equivalently be defined in many different and seemingly unrelated ways: by the density of *bounded-depth minors*, by *low tree-depth colourings* [56], by *generalised colouring numbers* [73], by wideness properties such as *uniformly quasi-wideness* [57], by *sparse neighbourhood covers* [37, 38], *vc-density* [62], and many more. Each of these different aspects of the theory comes with its own set of algorithmic tools and many of the more advanced algorithmic results on bounded expansion classes mentioned above crucially rely on a combination of several of these techniques.

Developing a structural theory for directed graphs that yields classes of digraphs with a similarly broad algorithmic impact has so far not seen a comparable success as for the undirected case. The general goal is to identify structural parameters which define interesting and general classes of digraphs for which there is a comparably rich set of algorithmic tools. However, essentially all approaches, e.g. in [6, 7, 34, 40, 59, 67], of generalising even the well-understood and fairly basic concept of tree width to digraphs have failed to produce digraph parameters that come even near the wide spectrum of algorithmic applications that tree width has found. This even has led to claims that this programme cannot be successful and that such measures for digraphs cannot exist [35].

In this paper we exhibit examples of digraph parameters which we believe challenge this negative outlook on the potential of digraph parameters. Our main conceptual contribution is to give a positive example of a digraph parameter that satisfies the conditions of the programme outlined above: we identify a very general type of digraph classes which have a similar set of algorithmic tools available as their undirected counterparts. We believe that these classes give a positive answer to the question whether interesting graph parameters can successfully be generalised to the directed setting and we support this claim by algorithmic applications described below.

The classes of digraphs we study are classes of *directed bounded expansion* and *nowhere crownful classes* of digraphs which are modeled after the concepts of *bounded expansion* and *nowhere denseness* for undirected graphs, respectively. They were originally defined

in [48], where basic properties of these classes were developed. In particular, it was shown that nowhere crownful classes can equivalently be defined in terms of *directed uniformly quasi-wideness*, analogous to its undirected counterpart, which easily implies fixed-parameter tractability of the directed dominating set problem on these classes. See Section 2 for details. The first improvement of these initial results appeared in [47], where structural properties of classes of digraphs of bounded expansion were studied. The main contribution of [47] was to establish their relation to a certain form of generalised colouring numbers, a concept which in the undirected setting has had huge algorithmic impact on the development of algorithms for nowhere dense and bounded expansion classes.

Our contributions.

These initial results are the starting point for our investigation in this paper. In addition to directed bounded expansion and nowhere crownful classes, we also define a new type of digraph classes which we call *bounded crownless expansion*.

Our main contributions are both structural and algorithmic. We show that classes of digraphs of directed bounded expansion, and especially classes of bounded crownless expansion, have structural properties very similar to their undirected counterpart. As a consequence, we are able to show that many of the algorithmic tools that were developed for undirected bounded expansion have their directed counterpart, resulting in a rich and diverse set of algorithmic techniques that can be applied in the design of algorithms for these classes. To the best of our knowledge, this is the first time that the generalisation of one of the widely studied and very general undirected graph parameter to the digraph setting has indeed led to a digraph concept with a similarly broad set of algorithmic tools as its undirected counterpart. We are therefore optimistic that classes of directed bounded expansion or crownless expansion will find a broad range of applications. We support this belief by providing several algorithmic results we describe next.

As a test case for these algorithmic techniques we use the directed variant of the (Distance-r) Dominating Set problem defined as follows. For a positive integer r, a *distance-r dominating set* in a digraph G is set $D \subseteq V(G)$ such that every $v \in V(G)$ is reachable by a directed path of length at most r from a vertex $d \in D$, i.e. $N_r^+(D) = V(G)$.

(Distance-r) Dominating Set is a common benchmark problem for the design of (parameterised or approximation) algorithms on graph classes with structural restrictions. It is NP-complete in general [42], and (under standard complexity theoretical assumptions) cannot be approximated better than up to a factor $\mathcal{O}(\log n)$ [64]. Better results can be achieved, e.g., on sparse graph classes, see e.g. [3, 4, 10, 12, 21, 22, 36, 39], but these classes do not contain classes of digraphs of bounded (crownless) expansion.

We study the complexity of the Directed (Distance-r) Dominating Set problem from the point of view of approximation, exact parameterised algorithms and kernelisation.

Approximation on directed bounded expansion. In [21], Dvořák proves a linear duality between distance-r dominating sets and r-scattered sets in classes of undirected bounded expansion. From this he derives an elegant polynomial-time constant-factor approximation algorithm on these classes of undirected graphs. Unfortunately, as we show in Section 3, no such duality holds in digraph classes of bounded directed expansion. In Theorem 7, we therefore use a different approach, inspired by recent results in [22], which is based on a combination of an LP-based approach and the characterisation of directed bounded expansion in terms of weak colouring numbers to obtain a constant-factor approximation algorithm for Directed-r Dominating Set on classes of directed bounded expansion.

Approximation on bounded crownless expansion. We then study classes of bounded crownless expansion. We first re-establish a polynomial duality between distance-r dominating sets and r-scattered sets on these classes. Towards this aim, we employ methods from stability theory, a branch of infinite model theory, developed in [50] in the digraph setting. The application of stability theory in this context is not straightforward. It is known that a class of (di)graphs which is closed under taking subgraphs is *stable*, if and only if, its underlying class of undirected graphs is nowhere dense [1]. However, classes of bounded crownless expansion in general are not nowhere dense and thus the stability theoretic techniques cannot be applied as such. Therefore, we have to carefully establish a situation in which stability is applicable, which then allows us to derive the polynomial duality theorem. As a consequence of this duality we also obtain a polynomial-time approximation algorithm for distance-r dominating sets (Corollary 11).

Parameterised complexity. We then study the parameterised complexity of the Distance-r Dominating Set problem. It is known that the problem is fixed-parameter tractable on nowhere crownful digraph classes [48] but the parameterised complexity of the problem on directed bounded expansion classes was still open. We first establish that classes of directed bounded expansion have *bounded directed neighbourhood depth*, a notion introduced in [26]. We then show that the methods developed in [26] can also be applied in the directed setting and establish that the Distance-r Dominating Set problem on classes of directed bounded expansion is fixed-parameter tractable (Theorem 14).

Kernelisation. Once fixed-parameter tractability is established, we turn our attention to the kernelisation problem for Distance-r Dominating Set. Recall that a kernelisation algorithm is a polynomial-time preprocessing algorithm that transforms a given instance into an equivalent one whose size is bounded by a function of the parameter only, independently of the overall input size. Fixed-parameter tractability implies the existence of a kernelisation algorithm, however, its output may be exponential or even larger in the parameter.

Starting with the groundbreaking work of Alber et al. [2], kernelisation for the Dominating Set and Distance-r Dominating Set problem on undirected graphs has received significant attention in the literature, see e.g. [8, 30, 31, 32]. In particular, Dominating Set admits polynomial kernels on graphs of bounded degeneracy [60]. The Distance-r Dominating Set problem admits a linear kernel on classes of bounded expansion [20], and an almost linear kernel on nowhere dense classes of graphs [45]. It is easy to observe that the result of [60] extends to digraphs of bounded degeneracy.

We show that the Distance-r Dominating Set problem admits a polynomial kernel on classes of bounded crownless expansion (Theorem 21). At a high level, our kernelisation algorithm follows the overall approach of [20] for undirected bounded expansion classes. Using our result above establishing the duality between distance-r dominating sets and r-scattered sets on bounded crownless expansion classes, the key property that remains to be established to apply the techniques from [20] are bounds on their distance-r neighbourhood complexity (the number of different intersections of r-balls with a given set). To establish these properties, we study the VC-dimension of set systems corresponding to r-neighbourhoods in digraphs of bounded directed expansion. In Section 4.2, we show that it is bounded on all classes of bounded crownless expansion which enables us to capture local separation properties in classes of bounded expansion. With this in place we can complete our kernelisation algorithm.

Steiner trees. As a further indication that digraphs of bounded expansion constitute a very useful notion, in Section 5 we consider the parameterised Directed Steiner Tree (Dst) problem, which is defined as follows. As input we are given a digraph G, a root $r \in V(G)$, a

set $T \subseteq V(G) \setminus \{r\}$ of terminals and an integer k. The problem is to decide if there is a set $S \subseteq V(G) \setminus (\{r\} \cup T)$ of size at most k such that in $G[\{r\} \cup S \cup T]$ there is a directed path from r to every terminal T. The Steiner Tree problem is an intensively studied graph problem in computer science with many important applications. We refer to the textbook of Prömel and Steger [63] for background information. It is known for this parameterisation that both the directed and the undirected versions are W[2]-hard on general graphs [52], and even on graphs of degeneracy two [41]. On the positive side, Jones et al. [41] proved that the problem is fixed-parameter tractable on graphs excluding a topological minor when parameterised by the number of non-terminals. Their result is based on a preprocessing rule which allows to contract strongly connected subsets of terminal vertices to individual vertices. The authors furthermore show that if the subgraph induced by the terminals is required to be acyclic, then the problem becomes fixed-parameter tractable on graphs of bounded degeneracy. In this case, the strongly connected subsets of terminals have diameter 0. This suggests to consider the problem parameterised by the number k of non-terminals plus the maximal diameter s of a strongly connected component in the subgraph induced by the terminals. In fact, bounded expansion classes of digraphs are exactly those classes whose graphs have bounded degeneracy after bounded radius contractions. Therefore, the Steiner tree problem is fixed-parameter tractable on classes of bounded directed expansion under this parameterisation. On the other hand, it is straightforward to modify the example in [41] to show that the parameterisation $k + s$ cannot be replaced by taking only k as parameter: there exist classes of directed bounded expansion on which the directed Steiner tree problem parameterised by solution size k is W[2]-hard. Hence, we show that the results of Jones et al. [41] exactly identify classes of directed bounded expansion as those on which the Directed Steiner Tree problem parameterised by the number of non-terminal vertices and the maximal diameter of strongly connected components in the subgraph induced by the terminals is fixed parameter tractable (Theorem 23). At the time of writing, Jones et al. simply did not have the notions of bounded expansion available.

Connected dominating sets. Finally, we show that the restriction to classes of bounded crownless expansion is not sufficient to find efficient algorithms for the Strongly Connected Dominating Set problem and Strongly Connected Steiner Subgraph (Scss) problem, which is defined as the Steiner tree problem but here we need to find a set $S \subseteq V(G)$ of size at most k such that $G[S \cup T]$ is strongly connected. We prove that there exist classes of bounded crownless expansion on which the Strongly Connected Dominating Set problem and the Strongly Connected Steiner Subgraph problem remain W[1]-hard (Theorems 30 and 31).

Summary. The results reported above demonstrate that classes of bounded (crownless) expansion indeed exhibit a very rich set of algorithmic tools, broad enough so that even recent sophisticated algorithms for undirected bounded expansion can be extended to the digraph setting. We therefore believe that these concepts are new and interesting digraph parameters which hold the promise for further algorithmic applications. The hardness results for strongly connected dominating sets, on the other hand, indicate that for problems which in addition require control over strong connectivity, one may have to consider further restrictions, e.g. by combining directed expansion with directed treewidth. We leave this for future research.

2 Directed Minors and Directed Bounded Expansion

We refer to [5] for standard notation and background on digraph theory. Let G be a digraph, let $v \in V(G)$ and let $r \geq 1$ be an integer. The *r-out-neighbourhood* of v, denoted

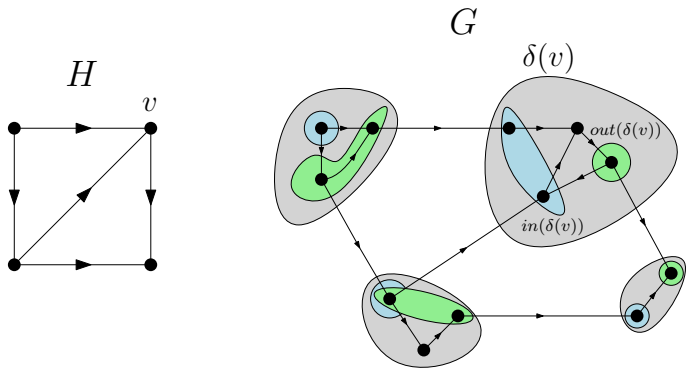


Figure 1 The graph H (left) is a directed minor of the graph G (right).

by $N^+_{G,r}(v)$, or just $N^+_r(v)$ if G is understood, is defined as the set of vertices u in G such that G contains a directed path of length at most r from v to u. We write $N^+(v)$ for $N^+_1(v) \setminus \{v\}$. The *r-in-neighbourhood* $N^-_{G,r}(v)$ and $N^-(v)$ are defined analogously. The *out-degree* of a vertex $v \in V(G)$ is $d^+(v) := |N^+(v)|$, its *in-degree* is $d^-(v) := |N^-(v)|$ and its *degree* is $d(v) := |N^+(v)| + |N^-(v)|$. The *minimum out-degree* of G is defined as $\delta^+(G) := \min\{d^+(v) \ : \ v \in V(G)\}$, *minimum in-degree* and *minimum degree* are defined analogously. A set $U \subseteq V(G)$ is *r-scattered* if there is no $v \in V(G)$ and $u_1, u_2 \in U$ with $u_1 \neq u_2$ and $u_1, u_2 \in N^+_r(v)$. If the arc relation of a digraph G is symmetric, i.e. if $(u, v) \in E(G)$ implies $(v, u) \in E(G)$, then we speak of an *undirected graph*. If G is a digraph, we write $\bar{G}$ for the *underlying undirected graph of* G, which has the same vertices as G and for each arc $(u, v) \in E(G)$ we have $(u, v) \in E(\bar{G})$ and $(v, u) \in E(\bar{G})$. Note that $|E(G)| \leq \left|E(\bar{G})\right| \leq 2\,|E(G)|$.

Directed minors. We are going to work with directed minors and directed topological minors. The following definition of directed minors is from [48]. A digraph H has a *directed model* in a digraph G if there is a function δ mapping vertices $v \in V(H)$ of H to sub-graphs $\delta(v) \subseteq G$ and arcs $e \in E(H)$ to arcs $\delta(e) \in E(G)$ such that

1. if $v \neq u$, then $\delta(v) \cap \delta(u) = \emptyset$;
2. if $e = (u, v)$ and $\delta(e) = (u', v')$ then $u' \in \delta(u)$ and $v' \in \delta(v)$. For $v \in V(H)$ let $\mathrm{in}(\delta(v)) := V(\delta(v)) \cap \bigcup_{e=(u,v)\in E(H)} V(\delta(e))$ and $\mathrm{out}(\delta(v)) := V(\delta(v)) \cap \bigcup_{e=(v,w)\in E(H)} V(\delta(e))$;
3. we require that for every $v \in V(H)$ (a) there is a directed path in $\delta(v)$ from every $u \in \mathrm{in}(\delta(v))$ to every $u' \in \mathrm{out}(\delta(v))$; (b) there is at least one source vertex $s_v \in \delta(v)$ that reaches (by a directed path in $\delta(v)$) every element of $\mathrm{out}(\delta(v))$; (c) there is at least one sink vertex $t_v \in \delta(v)$ that can be reached (by a directed path in $\delta(v)$) from every element of $\mathrm{in}(\delta(v))$.

A digraph H has a *directed model* in a digraph G if there is a function δ mapping vertices $v \in V(H)$ of H to sub-graphs $\delta(v) \subseteq G$ and arcs $e \in E(H)$ to arcs $\delta(e) \in E(G)$ such that

1. if $v \neq u$, then $\delta(v) \cap \delta(u) = \emptyset$;
2. if $e = (u, v)$ and $\delta(e) = (u', v')$ then $u' \in \delta(u)$ and $v' \in \delta(v)$.
3. Furthermore, we require that for each $v \in V(H)$ there are non-empty sets $\mathrm{in}(\delta(v))$ and $\mathrm{out}(\delta(v))$ such that $\mathrm{in}(\delta(v))$ contains the head of every arc $\delta((u, v))$ and $\mathrm{out}(\delta(v))$ contains the tail of every arc $\delta((v, u))$ and for every $s \in \mathrm{in}(\delta(v))$ and $t \in \mathrm{out}(\delta(v))$ there is a path in $\delta(v)$ from s to t.

We write $H \preccurlyeq G$ if H has a directed model in G and call H a *directed minor* of G. We call the sets $\delta(v)$ for $v \in V(H)$ the *branch-sets* of the model.

For $r \geq 0$, a digraph H is a *depth-r minor* of a digraph G, denoted as $H \preccurlyeq_r G$, if there exists a directed model δ of H in G in which for all $v \in V(H)$ and all $s \in \mathrm{in}(\delta(v))$ and $t \in \mathrm{out}(\delta(v))$ there is a path from s to t in $\delta(v)$ of length $\leq r$.

We write $H \preccurlyeq G$ if H has a directed model in G and call H a *directed minor* of G. We call the sets $\delta(v)$ for $v \in V(H)$ the *branch-sets* of the model.

For $r \geq 0$, a digraph H is a *depth-r minor* of a digraph G, denoted as $H \preccurlyeq_r G$, if there exists a directed model of H in G in which the length of all the paths in the branch sets of the model described in 3a)-c) above are bounded by r. Note that every subgraph of G is a depth-0 minor of G.

Directed topological minors. A digraph H is a *topological minor* of a digraph G if there is an injective function δ mapping vertices $v \in V(H)$ to vertices of $V(G)$ and arcs $e \in E(H)$ to directed paths in G such that if $e = (u, v) \in E(H)$, then $\delta(e)$ is a path from $\delta(u)$ to $\delta(v)$ in G which is internally vertex disjoint from all vertices $\delta(w)$ (for $w \in V(H)$) and all paths $\delta(e')$ (for $e' \in E(H)$, $e' \neq e$). For $r \geq 0$, H is a *topological depth-r minor* of G, written $H \preccurlyeq_r^{top} G$, if it is a topological minor and all paths $\delta(e)$ have length at most $2r$.

Grads, bounded expansion and crowns. Let G be a digraph and let $r \geq 0$. The *greatest reduced average density of rank r* (short *grad*) of G is

$$\nabla_r(G) := \max\left\{ \frac{|E(H)|}{|V(H)|} \;:\; H \preccurlyeq_r G \right\}$$

and its *topological greatest average density of rank r* (short *top-grad*) is

$$\widetilde{\nabla}_r(G) := \max\left\{ \frac{|E(H)|}{|V(H)|} \;:\; H \preccurlyeq_r^{top} G \right\}.$$

▶ **Definition 1.** *A class $\mathcal{C}$ of digraphs has* bounded expansion *if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for all $r \geq 0$ we have $\nabla_r(G) \leq f(r)$ (or equivalently, $\widetilde{\nabla}_r(G) \leq f(r)$) for all $G \in \mathcal{C}$.*

A *crown* of order q is a 1-subdivision of a clique of order q with all arcs oriented away from the subdivision vertices, that is, the digraph S_q with vertex set $\{v_1, \ldots, v_q\} \cup \{v_{ij} : 1 \leq i < j \leq q\}$ and arc set $\{(v_{ij}, v_i), (v_{ij}, v_j) : 1 \leq i < j \leq q\}$.

▶ **Definition 2.** *A class $\mathcal{C}$ of digraphs has* bounded crownless expansion *if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for all $r \geq 0$ we have $\nabla_r(G) \leq f(r)$ and $S_{f(r)} \not\preccurlyeq_r G$ for all $G \in \mathcal{C}$.*

Generalised colouring numbers. We next review the definition of generalised colouring numbers in the directed setting. Let G be a digraph. By $\Pi(G)$ we denote the set of all linear orders of $V(G)$. For $r \geq 0$, we say that u is *weakly r-reachable* from v with respect to an order $L \in \Pi(G)$ if there is a path P of length at most r, connecting u and v, *in either direction*, such that u is minimum among the vertices of P with respect to L. By $\mathrm{WReach}^{\rightleftarrows}_r[G, L, v]$ we denote the set of vertices that are weakly r-reachable from v with respect to L. We define the *weak r-colouring number* $\mathrm{wcol}^{\rightleftarrows}_r(G)$ of G as

$$\mathrm{wcol}^{\rightleftarrows}_r(G) := \min_{L \in \Pi(G)} \max_{v \in V(G)} \left| \mathrm{WReach}^{\rightleftarrows}_r[G, L, v] \right|.$$

▶ **Theorem 3** ([47]). *A class $\mathcal{C}$ of digraphs has bounded expansion if, and only if, there is $f : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{wcol}^{\rightleftarrows}_r(G) \leq f(r)$ for all $G \in \mathcal{C}$ and all $r \geq 1$.*

The next lemma shows that the weak r-colouring numbers are very useful to describe local separation properties in graphs of bounded expansion. The lemma is immediate by the definition of $\mathrm{WReach}_r^{\vec{\leftarrow}}$.

▶ **Lemma 4.** *Let G be a digraph and let $r \geq 1$. Let P be a path of length at most r with endpoints u and v in either direction. Let L be an order of $V(G)$ and let z be the minimal vertex of P with respect to L. Then $z \in \mathrm{WReach}_r^{\vec{\leftarrow}}[G, L, u] \cap \mathrm{WReach}_r^{\vec{\leftarrow}}[G, L, v]$.*

We will also need an efficient algorithm to compute good weak reachability orders. We show in the full version that this is possible. All statements marked with ($\star$) are proved in the full version [44].

▶ **Theorem 5** ($\star$). *Let $\mathcal{C}$ be a class of digraphs of bounded expansion. There exists a function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial-time algorithm which for an input graph $G \in \mathcal{C}$ and $r \in \mathbb{N}$ computes an order L with $|\mathrm{WReach}_r^{\vec{\leftarrow}}[G, L, v]| \leq f(r)$ for all $v \in V(G)$.*

3 Approximation of distance-r dominating sets and duality between distance-r dominating sets and r-scattered sets

In this section we study the duality between distance-r dominating sets and r-scattered sets and prove that for every fixed value $r \in \mathbb{N}$ the Distance-r Dominating Set problem admits a constant-factor approximation on every class of digraphs of bounded expansion. Given a digraph G, a set $U \subseteq V(G)$ is *r-scattered* if there is no $v \in V(G)$ and $u_1, u_2 \in U$ with $u_1 \neq u_2$ and $u_1, u_2 \in N_r^+(v)$. We write $\gamma_r(G)$ for the size of a minimum distance-r dominating set in a digraph G and $\alpha_{2r}(G)$ for the size of a maximum r-scattered set in G. Observe that in undirected graphs an r-scattered set corresponds to a distance-$2r$ independent set, which explains the index in the notation $\alpha_{2r}(G)$.

Clearly, every vertex $v \in V(G)$ can dominate at distance r at most one vertex of an r-scattered set. Hence we have $\alpha_{2r}(G) \leq \gamma_r(G)$ for every digraph G. In general, $\gamma_r(G)$ is not bounded in terms of $\alpha_{2r}(G)$. Dvořák proved in [21] that on classes of undirected graphs of bounded expansion $\gamma_r(G)$ is linearly bounded by $\alpha_{2r}(G)$, where the linear factor is the undirected weak colouring number $\mathrm{wcol}_{2r}(G)^2$, i.e., on undirected graphs the inequality $\gamma_r(G) \leq \mathrm{wcol}_{2r}(G)^2 \cdot \alpha_{2r}(G)$ holds. Furthermore, he derived an elegant linear time constant-factor approximation algorithm for the Distance-r Dominating Set problem.

As a first negative result we prove that no such duality theorem holds on digraphs of bounded expansion.

▶ **Theorem 6.** *There is a class of directed bounded expansion such that for every constant c we have $\gamma_1(G) \geq c$ for infinitely many $G \in \mathcal{C}$ and $\alpha_2(G) = 2$ for all $G \in \mathcal{C}$.*

Hence, we cannot follow the duality based approach to compute approximations for the Distance-r Dominating Set problem on classes of directed bounded expansion. Instead, we follow a very recent approach of Dvořák [22], which combines rounding of a linear program and a greedy choice based on the generalised colouring numbers. We consider the following linear programs. For each vertex $v \in V(G)$ we have one variable x_v.

Distance-r Dominating Set LP

- **Objective:** minimise $\gamma_r^\star = \sum_{v \in V(G)} x_v$
- **Subject to:** $\sum_{u \in N_r^-[v]} x_u \geq 1$ for all $v \in V(G)$
- **Constraints:** $x_v \geq 0$ for all $v \in V(G)$.

The dual linear program is the following program for r-Scattered Set.

r-Scattered Set LP

- **Objective:** maximise $\alpha^\star_{2r} = \sum_{v \in V(G)} x_v$
- **Subject to:** $\sum_{u \in N^-_r[v]} x_u \leq 1$ for all $v \in V(G)$
- **Constraints:** $x_v \geq 0$ for all $v \in V(G)$.

Integer solutions for the Distance-r Dominating Set LP correspond to minimum size distance-r dominating sets in G, and analogously, integer solutions for the r-Scattered Set LP correspond to maximum size r-scattered sets in G. Observe that since the linear programs are dual to each other, for every graph G and every positive integer r we have

$$\alpha_{2r}(G) \leq \alpha^\star_{2r}(G) = \gamma^\star_r(G) \leq \gamma_r(G),$$

while in general $\gamma_r(G)$ is not functionally bounded by $\alpha_{2r}(G)$. Also note that $\alpha^\star_{2r}(G)$ and $\gamma^\star_r(G)$ can be determined exactly in polynomial time by solving the linear programs that define them.

Dvořák in [22] proved that $\gamma_r(G)$ is bounded linearly by $\gamma^\star_r(G)$ on classes of undirected graphs of bounded expansion. We are able to prove an analogous statement in digraphs of bounded expansion. Furthermore, the theorem is constructive and yields a polynomial-time approximation algorithm.

▶ **Theorem 7** ($\star$). *Let $\mathcal{C}$ be a class of directed bounded expansion and let $r \in \mathbb{N}$. Then there exists a function $f\colon \mathbb{N} \to \mathbb{N}$ and a polynomial-time algorithm which on input $G \in \mathcal{C}$ computes a distance-r dominating set of G of size at most $f(r) \cdot \gamma^\star_r(G)$.*

We show next that for classes of bounded crownless expansion the values γ_r and α_{2r} are polynomially related. Thus, for such classes we can re-establish the duality between d-domination and r-scattered sets which we proved to fail in the general directed setting. Our proof is algorithmic in the sense that we apply the directed analogue of the algorithm of Dvořák [21] to the digraph G and prove that it finds both a distance-r dominating set and a polynomially smaller r-scattered set. Without requiring the duality to be polynomial we could have used standard Ramsey-type arguments. To establish a polynomial relation between the two parameters, we facilitate tools from stability theory, related to those developed in [50] and [46]. We first explain the stability theoretic tools used in the sequel.

Let T be a (rooted) binary tree, where each vertex (except the root) is marked as a left or right successor of its predecessor. We call w a *left (right) descendant* of v if the first successor on the unique v-w path in T is a left (right) successor.

Fix an enumeration $a_1, \ldots, a_\ell$ of a set $A \subseteq V(G)$. The *r-independence tree* of $(a_1, \ldots, a_\ell)$ is a binary tree which is constructed recursively as follows. We make a_1 the root of the tree. Assume that $a_1, \ldots, a_i$ have already been inserted into the tree. In order to insert the next element a_{i+1}, we follow a root-leaf path to find a position for it. Starting from the root a_1, at each point we are at some node a_j and we have to decide whether we continue along the left or to the right branch at a_j. If there is an element u such that $a_j, a_{i+1} \in N^+_r(u)$, we continue along the right branch at a_j, otherwise we follow the left branch. If there is no right successor (or left successor, respectively), we insert a_{i+1} as a right (or left) child of a_j.

▶ **Lemma 8** ($\star$). *Let T be a rooted binary tree and let $t \geq 1$ be an integer. Assume that no root-leaf path in T contains a sub-sequence $a_1, \ldots, a_t$ (of pairwise distinct elements) such that a_j is a right descendant of a_i for all $1 \leq i < j \leq t$. If T has height at most h, then T has at most h^{t+1} vertices.*

The following lemma is proved using the Finite Canonical Ramsey Theorem.

▶ **Lemma 9** (⋆). *For all integers r, c, K there exists an integer N such that the following property holds. Let G be a digraph with maximum out-degree at most c and let S, T be subsets of vertices of G, such that $|T| \geq N$ and for each $t, t' \in T$ there exist a vertex $s = s(t, t') \in S$, a directed path $P_{s,t}$ of length at most r from s to t and a directed path $P_{s,t'}$ of length at most r from s to t'. Then G contains a crown of order K as a depth-r minor.*

We can now prove the polynomial duality theorem.

▶ **Theorem 10.** *Let G be a digraph with $\mathrm{wcol}^{\geq}_r(G) \leq c$ and $S_q \not\preccurlyeq_r G$. Then there exists $N = N(c, q, r)$ such that $\gamma_r(G) \in \mathcal{O}(\alpha_r(G)^N)$.*

Proof. The following algorithmic construction corresponds to the algorithm of Dvořák for undirected graphs [21]. Fix an order L witnessing that $\mathrm{wcol}^{\geq}_r(G) \leq c$. We compute a distance-$r$ dominating set D as follows. Initialise $D := \emptyset$, $A := \emptyset$ and $M := V(G)$. While there is a vertex $v \in M$, the set of non-dominated vertices, pick the smallest such vertex v with respect to L. Add v to A and $\mathrm{WReach}^{\geq}_{2r}[G, L, v]$ to D. Mark all newly dominated vertices, that is, remove $N^+_r[\mathrm{WReach}^{\geq}_{2r}[G, L, v]]$ from M. If $M = \emptyset$, return D. Clearly, D is a distance-r dominating set of G.

We now prove that we find a large r-scattered subset of A. Construct the undirected graph H with vertex set A such that two vertices $a, b \in A$ are connected in H if there is $u \in V(G)$ such that $a, b \in N^+_r(u)$. An independent set in H corresponds to an r-scattered subset of A in G.

We claim that every vertex $u \in V(G)$ satisfies $|N^+_r(u) \cap A| \leq c$. Fix $u \in V(G)$. Assume towards a contradiction that $|N^+_r(u) \cap A| > c$. For each $a \in N^+_r(u) \cap A$ fix a path P_{ua} of length at most r from u to a. For each path P_{ua}, denote by m_{ua} its minimal element with respect to L. Since $\mathrm{wcol}^{\geq}_r(G) \leq c$, we have $|\{m_{ua} : N^+_r(u) \cap A\}| \leq c$. Since we have more than c paths P_{ua}, there must be two paths P_{ua_1}, P_{ua_2}, $a_1 \neq a_2$, which have the same element m as their minimal element. Without loss of generality assume that $a_1 < a_2$. Since m is the smallest vertex on the path P_{ua_1}, the subpath of P_{ua_1} between m and a_1 certifies that m is weakly r-reachable from a_1. Hence, when a_1 was added to A, the element m was added to the set D. Now, the subpath of P_{ua_2} between m and a_2 shows that a_2 is at distance at most r from m, and hence a_2 is marked as dominated at this point. This again proves $a_2 \notin A$, a contradiction.

We now build the r-independence tree T of $a_1, \ldots, a_\ell$ (the enumeration of A with respect to L). Using Lemma 9, we conclude that there is $N'(c, r, q)$ such that T does not contain a path with $s = N'$ right descendants. Let $N := N' + 1$.

Hence, by Lemma 8, if we have $|A| > (m + N)^N$, then we find a sequence of length m with all left descendants. This set is r-scattered, which proves the theorem. ◀

Clearly, the r-independence tree of a sequence of vertices can be computed in polynomial time, which gives us the following corollary.

▶ **Corollary 11.** *Let $\mathcal{C}$ be a class of digraphs which has bounded crownless expansion. Then for every $r \in \mathbb{N}$, there is a polynomial-time algorithm which computes a distance-r dominating set D with $|D| \leq p(\gamma_r(G))$ for some polynomial p.*

4 Parameterised complexity of Distance-r Dominating Set

In this section we study the parameterised complexity of the Distance-r Dominating Set problem on classes of directed bounded expansion. We follow the approach of [26] and establish that digraphs of bounded expansion have bounded neighbourhood depth

(which corresponds to having bounded semi-ladder index in that paper). We then show that a straight forward modification of the Semi-ladder-algorithm of [26] for the Distance-r Dominating Set problem on undirected graphs of bounded neighbourhood depth is an fpt-algorithm on digraphs of bounded expansion.

Let $\mathcal{F}$ be a family of subsets of some universe U. A *chain* in $\mathcal{F}$ is a family $\mathcal{H} \subseteq \mathcal{F}$ such that for all $X, Y \in \mathcal{H}$, we have either $X \subseteq Y$ or $Y \subseteq X$. The *depth* of $\mathcal{F}$ is the cardinality of the longest chain in $\mathcal{F}$. The *intersection closure* of $\mathcal{F}$ is the family of all sets of the form $X_1 \cap X_2 \cap \ldots \cap X_n$ for some $n \in \mathbb{N}$ and $X_1, X_2, \ldots, X_n \in \mathcal{F}$. For $n = 0$ we assume by convention that the intersection of an empty sequence of sets is equal to U, thus the intersection closure always contains the universe U.

▶ **Definition 12.** *Let G be a digraph and $r \in \mathbb{N}$. The* r-neighbourhood depth *of G, denoted* $\mathsf{depth}_r(G)$*, is the depth of the intersection closure of the family $\{N_r^+(v) \colon v \in V(G)\}$. We say that a graph class $\mathcal{C}$ has* bounded neighbourhood depth *if there is a function $f \colon \mathbb{N} \to \mathbb{N}$ such that for all $G \in \mathcal{C}$ we have $\mathsf{depth}_r(G) \leq f(r)$.*

We show that classes of directed bounded expansion have bounded neighbourhood depth.

▶ **Lemma 13** ($\star$)**.** *Let $\mathcal{C}$ be a class of directed bounded expansion. Then $\mathcal{C}$ has bounded neighbourhood depth.*

4.1 Fixed-parameter tractability on bounded expansion classes

In this section we show that a straightforward modification of the so-called Semi-ladder-algorithm of [26] is an fpt-algorithm on digraphs of bounded neighbourhood depth.

We say that a set of vertices A *r-dominates* another set of vertices B if $B \subseteq N_r^+(A)$. The Semi-ladder-algorithm maintains two sets: $D, S \subset V(G)$. Initially, both are empty, and at each moment, D will have at most k elements. The algorithm proceeds in rounds, each consisting of two steps: first the *S-step* and then the *D-step*.

S-step: Check whether D r-dominates $V(G)$. If so, terminate and output D as an r-dominating set of size at most k. Otherwise, pick any vertex u which is not r-dominated by D and add it to S.

D-step: Check whether some set of at most k vertices r-dominates S. If so, set D to be any such set and proceed to the next round. Otherwise, terminate and conclude that there is no r-dominating set of size at most k.

As in the undirected case one can easily implement each D-**step** using standard dynamic programming on subsets of S in time $\mathcal{O}(2^{|S|} \cdot |S| \cdot n)$. Since at each round the size of S grows by exactly 1, it is not hard to see that the ℓth round of the algorithm can be implemented in time $\mathcal{O}(2^\ell \cdot \ell n + km)$, and hence the time needed to execute it for L rounds is bounded by $\mathcal{O}(2^L \cdot Ln + kLm)$.

Clearly, the algorithm correctly decides whether a graph contains a distance-r dominating set of size at most k. It remains to show that it is in fact a fixed-parameter algorithm on classes of directed bounded expansion. We prove this by showing that the neighbourhood depth gives an upper bound on the number L of rounds executed by the algorithm.

▶ **Theorem 14** ($\star$)**.** *Let $\mathcal{C}$ be a class with bounded neighbourhood depth and let $r \in \mathbb{N}$. Then for every $k \in \mathbb{N}$ there is a constant $L \in \mathbb{N}$, depending only on $k, r, \mathcal{C}$ and computable from k for fixed r and $\mathcal{C}$, such that the Semi-ladder-algorithm terminates after at most L rounds when applied to any $G \in \mathcal{C}$ and k. In particular, if G has n vertices and m edges, then the running time is bounded by $f(k) \cdot m$ for some computable function f.*

4.2 VC-dimension and neighbourhood complexity

Towards the goal of developing a kernelisation algorithm for the Distance-r Dominating Set problem on classes of bounded crownful expansion, we first study the VC-dimension and neighbourhood complexity of radius-r balls in classes of directed bounded expansion.

Let $\mathcal{F} \subseteq 2^A$ be a family of subsets of a set A. For a set $X \subseteq A$, we denote $X \cap \mathcal{F} = \{X \cap F : F \in \mathcal{F}\}$. The set X is *shattered by* $\mathcal{F}$ if $X \cap \mathcal{F} = 2^X$. The *Vapnik-Chervonenkis dimension*, short *VC-dimension*, of $\mathcal{F}$ is the maximum size of a set X that is shattered by $\mathcal{F}$.

Note that if $\mathcal{F}$ has VC-dimension d, then also $B \cap \mathcal{F}$ for every subset $B \subseteq A$ of the ground set has VC-dimension at most d. The following theorem was first proved by Vapnik and Chervonenkis [72], and rediscovered by Sauer [68] and Shelah [70]. It is often called the Sauer-Shelah lemma in the literature.

▶ **Theorem 15.** *If $|A| \leq n$ and $\mathcal{F} \subseteq 2^A$ has VC-dimension d, then $|\mathcal{F}| \leq \sum_{i=0}^{d} \binom{n}{i} \in \mathcal{O}(n^d)$.*

The study of the distance-r dominating set problem in context of bounded VC-dimension motivates the following definition. Let G be a digraph and $r \geq 1$. The *distance-r VC-dimension* of G is the VC-dimension of the set family $\{N_r^-(v) \colon v \in V(G)\}$ over the set $V(G)$. If $X \subseteq V(G)$, the *distance-r neighbourhood complexity of X in G*, denoted $\nu^-(G)$, is defined by

$$\nu^-(G, X) \coloneqq \left|\{N_r^-(v) \cap X \ : \ v \in V(G)\}\right|.$$

Analogously, one can define the *distance-r out-neighbourhood complexity* when using $N_r^+(v)$ and the *distance-r mixed neighbourhood complexity* when using $(N_r^+(v) \cup N_r^-(v))$ in the above definition and our proofs can be analogously carried out for these measures.

It was proved in [65] that a class $\mathcal{C}$ of undirected graphs has bounded expansion, if and only if, for every $r \geq 1$ there is a constant c_r such that for all $G \in \mathcal{C}$ and all $X \subseteq V(G)$ we have $\nu(G, X) \leq c_r \cdot |X|$ (where ν denotes the undirected neighbourhood complexity). The analogous statement for classes of directed graphs does not hold, not even for $r = 1$, as pointed out in [47]. However, we prove that the distance-r neighbourhood complexity of a digraph can be bounded in terms of its weak r-colouring numbers.

Using Lemma 4 we can well control the interaction of distance-r neighbourhoods with a set X. Let G be a digraph and let L be a linear order on $V(G)$ and let $r \geq 1$. Let $A \subseteq V(G)$ be enumerated as $a_1, \ldots, a_{|A|}$, consistently with the order. For $v \in V(G)$ let $D_r^-(v, A)$ denote the *distance-r vector* of v and A, that is, the vector $(d_1, \ldots, d_{|A|})$, where $d_i = \operatorname{dist}(a_i, v)$ if $0 \leq \operatorname{dist}(a_i, v) \leq r$, and ∞ otherwise. Here $\operatorname{dist}(a_i, v)$ is the length of a shortest path from a_i to v.

▶ **Lemma 16 ($\star$).** *Let G be a digraph, let $X \subseteq V(G)$ and let $r \geq 1$. Let $c \coloneqq \operatorname{wcol}_r^{\vec{>}}(G)$. Then the number of distinct distance-r vectors $D_r^-(v, X)$ is bounded by $((r+2) \cdot c \cdot |X|)^c$, and in particular, $\nu_r^-(G, X) \leq ((r+2) \cdot c \cdot |X|)^c$.*

▶ **Corollary 17.** *Let G be a digraph and $r \geq 1$. Then the distance-r VC-dimension of G is bounded by $(r+2) \cdot (2\operatorname{wcol}_r^{\vec{>}}(G))^2$.*

4.3 Kernelisation on classes of bounded crownful expansion

Recall that a kernelisation algorithm is a polynomial-time preprocessing algorithm that transforms a given instance into an equivalent one whose size is bounded by a function of the parameter only, independently of the overall input size. We are mostly interested in kernelisation algorithms whose output guarantees are polynomial in the parameter. In this

section we prove that for every fixed value of $r \geq 1$, the distance-r dominating set problem admits a polynomial kernel on every class of bounded crownless expansion.

Our strategy follows on a high level that of Drange et al. [20] for kernelisation on classes of undirected bounded expansion. The first step is to compute a small domination core.

▶ **Definition 18** (r-domination core). *Let G be a digraph. A set $Z \subseteq V(G)$ is an* r-domination core *in G if every minimum-size set which r-dominates Z also r-dominates G.*

Clearly, the set $V(G)$ is an r-domination core. We will show how to iteratively remove vertices from this trivial core, to arrive at smaller and smaller domination cores, until finally, we arrive at a core of polynomial size in k. Observe that we do not require that every r-dominating set for Z is also an r-dominating set for G; there can exist dominating sets for Z which are not of minimum size and which do not dominate the whole graph.

▶ **Lemma 19** ($\star$). *There exists a polynomial p and a polynomial-time algorithm that, given an r-domination core $Z \subseteq V(G)$ with $|Z| > p(k)$, either correctly decides that G cannot be dominated by k vertices, or finds a vertex $z \in Z$ such that $Z \setminus \{z\}$ is still an r-domination core.*

Hence, by gradually reducing $|Z|$, we arrive at the following theorem.

▶ **Theorem 20.** *There exists a polynomial p and a polynomial-time algorithm that, given an instance (G, k) where $G \in \mathcal{C}$, either correctly decides that G cannot be dominated by k vertices, or finds an r-domination core $Z \subseteq V(G)$ with $|Z| \leq p(k)$.*

Now that it remains to dominate a subset Z, we may keep one representative from each equivalence class in the equivalence relation: $u \cong_{Z,r} v \Leftrightarrow N_r^+(u) \cap Z = N_r^+(v) \cap Z$. As before, there are only polynomially many equivalence classes, hence from a polynomial domination core we can construct a polynomial kernel.

▶ **Theorem 21.** *Let $\mathcal{C}$ be a class of bounded expansion. There is a polynomial time algorithm which on input G, k and r computes a subgraph $G' \subseteq G$ and a set $Z \subseteq V(G')$ such that G can be r-dominated by k vertices if, and only if, Z can be r-dominated by k vertices in G' and $|Z| \leq p(k)$.*

5 Steiner trees

▶ **Definition 22.** *The Directed Steiner Tree (DST) problem is defined as follows. The input is a tuple (G, r, T, k) where G is a digraph, $r \in V(G)$ is a vertex (a root), $T \subseteq V(G) \setminus \{r\}$ is a set of terminals and k is an integer. The problem is to decide if there is a set $S \subseteq V(G) \setminus (\{r\} \cup T)$ of size at most k such that in $G[\{r\} \cup S \cup T]$ there is a directed path from r to every terminal T.*

The Dst problem has been widely studied in the area of approximation algorithms as it generalises several routing and domination problems. We are interested in the parameterised complexity of this problem. It follows from an algorithm by Nederlof [54] and Misra et al. [51], that the problem can be solved in time $2^{|T|} \cdot p(n)$, for some polynomial $p(n)$. In this paper, we are interested in the standard parameterisation in parameterised complexity, where as parameter we take the solution size, i.e. we take the number k of non-terminals as parameter. This models the case where we need to pay for any node we add to the solution and we want to keep the bound k on these nodes as small as possible without any restriction on the number of terminals to connect.

In [41], Jones et al. show that DST with this parameterisation is fixed-parameter tractable on any class of digraphs such that the class of underlying undirected graphs excludes a fixed graph H as an undirected topological minor, as well as on any class of degenerate graphs if the set T of terminal vertices induces an acyclic graph. We immediately conclude the following.

▶ **Theorem 23** (⋆). *Let $\mathcal{C}$ be a class of digraphs of bounded expansion.* DST *is fixed-parameter tractable on $\mathcal{C}$ parameterised by the number k of non-terminals in the solution plus the maximal diameter s of the strongly connected components in the subgraph induced by the terminals.*

The proof of the theorem has the following immediate consequences.

▶ **Corollary 24.** *Let $\mathcal{C}$ be a class of digraphs closed under taking directed minors for which $\nabla_0(G) \leq c$ for a constant c for all $G \in \mathcal{C}$. Then* DST(G, r, T, k) *can be solved for all $G \in \mathcal{C}$, $r \in V(G)$, $T \subseteq V(G) \setminus \{r\}$ and k in time $2^{\mathcal{O}(k)} \cdot p(n)$, for some fixed polynomial $p(n)$.*

Note that this strictly generalises classes of undirected graphs excluding a fixed minor.

Another consequence of this is the following result, which immediately follows from the well-known observation in parameterised complexity (see e.g. [41, Lemma 7]), that for all functions $g(n) = o(\log n)$ there is a function $f(k)$ such that $f(k) \leq 2^{g(n)\cdot k}$, for all k and all n.

▶ **Corollary 25.** *Let $\mathcal{C}$ be a class of digraphs such that $\nabla_{|G|}(G) \cdot \log \nabla_{|G|}(G) \leq o(\log n)$ for all $G \in \mathcal{C}$. Then* DST *is fixed-parameter tractable on $\mathcal{C}$ with parameter k.*

Finally, the result also implies an fpt factor-2-approximation algorithm for the STRONGLY CONNECTED STEINER SUBGRAPH problem, SCSS, on classes of bounded directed expansion. In the SCSS we are given a digraph G, a number k, and a set T of terminals and we are asked to compute a set S of at most k non-terminals such that $G[T \cup S]$ is strongly connected.

▶ **Theorem 26** (⋆). *Let $\mathcal{C}$ be a class of digraphs of bounded expansion. There is an fpt factor-2-approximation algorithm for* SCSS *on $\mathcal{C}$ parameterised by the number k of non-terminals in the solution plus the maximal diameter s of a strongly connected component in the subgraph of G induced by the terminal nodes.*

We close the section by showing that for bounded expansion classes, the parameterisation $k + s$ in Theorem 23 cannot be replaced by taking only k as parameter. This follows immediately from a result of [41] where it is shown that SET COVER can be reduced to DST on 2-degenerate graphs. It is straightforward to modify this example so that the resulting class of graphs has bounded directed expansion.

▶ **Theorem 27.** *The* DST*-problem restricted to classes of digraphs of bounded expansion parameterised by the solution size k is $W[2]$-hard.*

6 Hardness Results

In this section, we investigate the problems of DOMINATING SET and STEINER SUBGRAPH in classes of digraphs of bounded crownless expansion when we require strong connectivity for the graph induced by the output sets. We will show that both STRONGLY CONNECTED DOMINATING SET and STRONGLY CONNECTED STEINER SUBGRAPH are W[1]-hard.

The parameterised STRONGLY CONNECTED DOMINATING SET problem (SCDS) is the problem to decide whether a given digraph G contains a dominating set $D \subseteq V(G)$ of size at most k such that the digraph induced by D is strongly connected, where k is an input

parameter. We prove that SCDS parameterised by solution size is W[1]-hard even in graphs of bounded crownless expansion. The proof is a reduction from the MULTICOLOURED CLIQUE problem, which is known to be W[1]-hard [27]. Given an integer k and a graph G whose vertex set is partitioned into k independent sets $V_1, V_2, \ldots, V_k$ called *colour classes*, the MULTICOLOURED CLIQUE problem asks whether there exists a k-vertex clique in G with exactly one vertex from every colour class.

The reduction. Let (G, k) be an instance of MULTICOLOURED CLIQUE and let $V_1, \ldots, V_k$ be the colour classes of G. We can assume that V_i is independent for each $1 \leq i \leq k$. We construct an instance (H, p) of STRONGLY CONNECTED DOMINATING SET, for a parameter p to be defined, starting from the graph G. For each class V_i we add two vertices s_i^1, s_i^2 and we connect each vertex $v \in V_i$ to both of these vertices with two edges vs_i^1 and vs_i^2.

For each $1 \leq i, j \leq k$ with $i \neq j$, let $E_{i,j}$ be the set of edges of G with one end in V_i and the other in V_j. For each $E_{i,j}$ we define $\mathcal{C}_{i,j}$ to be a set of $|E_{i,j}|$ directed cycles of length $2k + 6$ such that they intersect in a single vertex $v_{i,j}$. Further, we index the set $\mathcal{C}_{i,j}$ by the elements of $E_{i,j}$. For each cycle C_e of $\bigcup_{1 \leq i,j \leq k} \mathcal{C}_{i,j}$ we denote with x_e the vertex of C_e such that the length of the path starting in x_e and ending in $v_{i,j}$ is $k + 1$. Similarly we define y_e to be the vertex such that the length of the path starting in $v_{i,j}$ and ending in y_e is $k + 1$. We further denote by z_e the vertex of C_e in $N^-(x_e)$.

For each $i < j$, with the exception of the pair $(1, k)$, we replace each edge $e = uv \in E_{i,j}$ with $u \in V_i$ and $v \in V_j$, with the two directed edges ux_e and $y_e v$. The pair $\{V_1, V_k\}$ is connected in the following way. For the pair $(1, k)$ we replace each $e' = u'v' \in E_{1,k}$ with $u' \in V_1$ and $v' \in V_k$ with the edges $v'x_e$ and $y_e u'$.

In addition, for each pair of classes $\{V_i, V_j\}$ we add two vertices $s_{i,j}^1, s_{i,j}^2$ and draw edges $z_e s_{i,j}^1$ and $z_e s_{i,j}^2$ for all $C_e \in \mathcal{C}_{i,j}$. We call the vertices of the set $\{s_{i,j}^l : l = 1, 2, 1 \leq i, j \leq k, i \neq j\} \cup \{s_i^l : 1 \leq i \leq k, l = 1, 2\}$ the *top vertices* of H.

Lastly, we add one vertex q and we draw qv directed edges for each v in one of the colour classes and an edge qv' for each vertex v' in one of the cycles $C \in \bigcup \mathcal{C}_{i,j}$. Since we want to be able to maintain strong connectivity when q is in D, we must add some edges directed towards q. In particular we add an edge vq for an arbitrary v in each cycle C_e with $e \in E_{1,2}$. This concludes the construction of (H, p).

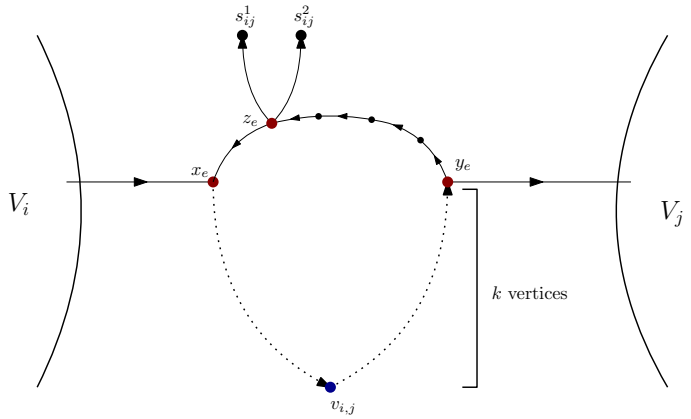


Figure 2 Each cycle C_e is connected to s_{ij}^1 and s_{ij}^2 with two edges incident to z_e. All cycles corresponding to edges of E_{ij} intersect in the vertex v_{ij}.

Before we proceed with our proof of the hardness, we will prove that the graph obtained is of bounded crownless expansion. We need the following easy lemma.

▶ **Lemma 28** (⋆). *Let G be a graph of density $\frac{|E(G)|}{|V(G)|} = D$. Let G' be a graph with $|V(G')| = |V(G)| + t$ and $|E(G')| = |E(G)| + d$ edges. The density D' of G' is greater than D if and only if $D < d/t$.*

▶ **Lemma 29** (⋆). *Let (G, k) be an instance of* Multicoloured Clique *and let (H, p) be the correspondent instance of* Strongly Connected Dominating Set *constructed as described above. We prove that $\widetilde{\nabla}_r(H) \leq (r-1)/2$ and $S_{f(r)} \not\preccurlyeq_r H$.*

▶ **Theorem 30.** *There exists a class $\mathcal{C}$ of digraphs of bounded crownless expansion such that* Strongly Connected Dominating Set *parameterised by size of the solution is* W[1]*-hard on $\mathcal{C}$.*

Proof. Let (G, k) be an instance of Multicoloured Clique and let (H, p) be the correspondent instance of Strongly Connected Dominating Set with $p = k + \binom{k}{2}(6 + 2k) + 1$.

We claim that if a multicoloured clique K_k exists in G then there is strongly connected dominating set D of size at most p in H. For each uv in K_k, let u' and v' be the corresponding vertices in H and let C_{uv} be the cycle of $\mathcal{C}_{i,j}$ that they are connected to. Take S to be the union of $V(C_{uv})$ and the vertices u' and v' over all $uv \in E(K_k)$. Further, we take $D = S \cup \{q\}$. The size of D is equal to $k + \binom{k}{2}(6 + 2k) + 1$ which in turn is equal to the parameter p of the instance (H, p). It is easy to see that D is a dominating set. The vertex q dominates the vertices included in the colour classes. What is left to check are the top vertices. By taking a vertex in each class V_i, we ensure that s^1_i and s^2_i are dominated. In addition, for each edge uv of K_k, with $u \in V_i$ and $v \in V_j$, we add the vertices of the cycle C_{uv} which include the vertex $z_{C_{uv}}$. Hence, the vertices $s^1_{i,j}, s^2_{i,j}$ are dominated. It is easy to see that D is strongly connected. For every $i < j$, $u \in V_i$ and $v \in V_j$, there is a directed path starting in u and ending in v. The vertices on the cycles are connected through the cycles to the same paths. The path starting in V_k and ending in V_i ensures the strong connectivity for these vertices. Since we have added at least one cycle $C \in \mathcal{C}_{1,2}$, strong connectivity is preserved for $q \in D$.

We will now prove that the existence of a strongly connected dominating set D of size at most p in H implies the existence of a multicoloured clique K_k in G. We know that for each pair i, j with $1 \leq i, j \leq k$, D needs to dominate $s^1_{i,j}$ and $s^2_{i,j}$. Hence D must contain at least a vertex z_e for some $C_e \in \mathcal{C}_{i,j}$. Hence, since D is strongly connected $G[D]$ must contain the cycle C_e. In addition, for each pair of cycles C_e and $C_{e'}$ there must be a path connecting them. It follows that D contains edges ux_e and $y_e u$ for some u, v with $u \in V_i$ and $v \in V_j$. Let us assume that G does not contain a multicoloured clique K_k, we will prove that the $|D| > p$. We know that for each pair i, j, D contains at least cycle C_e plus two vertices u, v. Since there does not exist a multicoloured clique in G, there exists at least three vertices $u, v, w \in D$ in distinct colour classes, such that the corresponding vertices u', v', w' of G are such that $u'v' \in E(G)$ and $u'w' \notin E(G)$. Hence, there exists at least one class V_i such that D contains two vertices in that class. Further, D must dominate all the vertices contained in cycles that are not in D and the vertex q. As a consequence, the vertex q must be in D and $|D| \geq k + \binom{k}{2}(6 + 2k) + 2$ which is larger than the parameter p. ◀

Similarly, we show that Strongly Connected Steiner Subgraph is hard.

▶ **Theorem 31** (⋆). *There exists a class $\mathcal{C}$ of digraphs of bounded crownless expansion such that* Strongly Connected Steiner Subgraph *parameterised by size of the solution is* W[1]*-hard on $\mathcal{C}$.*

References

1 Hans Adler and Isolde Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics*, 36:322–330, 2014.

2 Jochen Alber, Michael R Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM (JACM)*, 51(3):363–384, 2004.

3 Saeed Akhoondian Amiri, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz. Distributed Domination on Graph Classes of Bounded Expansion. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 143–151, 2018.

4 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

5 Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

6 János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

7 Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. DAG-width and parity games. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 524–536. Springer, 2006.

8 Hans L Bodlaender, Fedor V Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M Thilikos. (Meta) kernelization. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 629–638. IEEE, 2009.

9 Marthe Bonamy, Łukasz Kowalik, Michał Pilipczuk, and Arkadiusz Socała. Linear kernels for outbranching problems in sparse digraphs. *Algorithmica*, pages 1–30, 2015.

10 Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

11 Julia Chuzhoy and Shi Li. A Polylogarithmic Approximation Algorithm for Edge-Disjoint Paths with Congestion 2. *J. ACM*, 63(5):45:1–45:51, November 2016. doi:10.1145/2893472.

12 Anuj Dawar, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Approximation schemes for first-order definable optimisation problems. In *21st Annual IEEE Symposium on Logic in Computer Science, 2006*, pages 411–420. IEEE, 2006.

13 Anuj Dawar and Stephan Kreutzer. Domination Problems in Nowhere-Dense Classes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 157–168, 2009.

14 E. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, pages 332–337, 2008.

15 E. D. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic Graph Minor Theory: Decomposition, Approximation, and Coloring. In *46th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 637–646, 2005.

16 Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005.

17 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Fast Algorithms for Hard Graph Problems: Bidimensionality, Minors, and Local Treewidth. In *Graph Drawing*, pages 517–533, 2004. URL: http://springerlink.metapress.com/openurl.asp?genre=article{&}issn=0302-9743{&}volume=3383{&}spage=517.

18 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *Symp. on Discrete Algorithms (SODA)*, pages 590–601, 2005. doi:10.1145/1070432.1070514.

19 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural Sparsity of Complex Networks: Random Graph Models and Linear Algorithms. *CoRR*, abs/1406.2587, 2014. URL: http://arxiv.org/abs/1406.2587, arXiv:1406.2587.

20 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and Sparseness: the Case of Dominating Set. In *STACS*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

21 Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5):833–840, 2013.

22 Zdeněk Dvořák. On distance r-dominating and 2r-independent sets in sparse graphs. *arXiv preprint arXiv:1710.10010*, 2017.

23 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM (JACM)*, 60(5):36, 2013.

24 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy Kernels for Connected Dominating Set on Sparse Graphs. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 29:1–29:15, 2018.

25 Kord Eickmeyer, Archontia C Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michał Pilipczuk, Roman Rabinovich, Sebastian Siebertz, et al. Neighborhood complexity and kernelization for nowhere dense classes of graphs. *arXiv preprint arXiv:1612.08197*, 2016.

26 Grzegorz Fabiański, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Progressive Algorithms for Domination and Independence. *CoRR*, abs/1811.06799, 2018. `arXiv:1811.06799`.

27 Michael R Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

28 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *SODA*, pages 748–759, 2011. URL: `http://www.siam.org/proceedings/soda/2011/SODA11_058_fominf.pdf`.

29 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and Kernels. In *SODA*, pages 503–510, 2010. URL: `http://www.siam.org/proceedings/soda/2010/SODA10_043_fominf.pdf`.

30 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Bidimensionality and kernels. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 503–510. SIAM, 2010.

31 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 82–93. Society for Industrial and Applied Mathematics, 2012.

32 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In *30th International Symposium on Theoretical Aspects of Computer Science*, page 92, 2013.

33 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sanchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *European Symposium on Algorithms*, pages 529–540. Springer, 2013.

34 Robert Ganian, Petr Hliněný, Joachim Kneis, Alexander Langer, Jan Obdržálek, and Peter Rossmanith. On digraph width measures in parameterized algorithmics. In *International Workshop on Parameterized and Exact Computation*, pages 185–197. Springer, 2009.

35 Robert Ganian, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *J. Comb. Theory, Ser. B*, 116:250–286, 2016. `doi:10.1016/j.jctb.2015.09.001`.

36 Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

37 Martin Grohe, Stephan Kreutzer, Roman Rabinovich, Sebastian Siebertz, and Konstantinos Stavropoulos. Colouring and covering nowhere dense graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 325–338. Springer, 2015.

38 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 89–98. ACM, 2014.

39 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM Journal on Computing*, 46(6):1712–1744, 2017.

40 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008.

41 M. Jones, D. Lokshtanov, M.S. Ramanujan, S. Saurabh, and O. Suchy. Parameterized Complexity of Directed Steiner Tree on Sparse Graphs. In *Proceedings of European Symposium on Algorithms (ESA 2013)*, 2013.

42 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

43 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 297–308. ACM, 2013.

44 Stephan Kreutzer, Irene Muzi, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz. Algorithmic Properties of Sparse Digraphs. *CoRR*, abs/1707.01701, 2017.

45 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial Kernels and Wideness Properties of Nowhere Dense Graph Classes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1533–1545. SIAM, 2017.

46 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial Kernels and Wideness Properties of Nowhere Dense Graph Classes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1533–1545, 2017. doi:10.1137/1.9781611974782.100.

47 Stephan Kreutzer, Roman Rabinovich, Sebastian Siebertz, and Grischa Weberstädt. Structural Properties and Constant Factor-Approximation of Strong Distance-r Dominating Sets in Sparse Directed Graphs. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2017.48.

48 Stephan Kreutzer and Siamak Tazari. Directed nowhere dense classes of graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1552–1562. SIAM, 2012.

49 Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. In *Workshop on Algorithms and Data Structures*, pages 506–517. Springer, 2015.

50 Maryanthe Malliaris and Saharon Shelah. Regularity lemmas for stable graphs. *Transactions of the American Mathematical Society*, 366(3):1551–1585, 2014.

51 N. Misra, G. Philip, V. Raman, S. Saurabh, and S. Sikdar. FPT algorithms for connected feedback vertex set. In *WALCOM 2010*, pages 269–280, 2010.

52 Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory of Computing Systems*, 43(2):234–253, 2008.

53 Wojciech Nadara, Marcin Pilipczuk, Roman Rabinovich, Felix Reidl, and Sebastian Siebertz. Empirical Evaluation of Approximation Algorithms for Generalized Graph Coloring and Uniform Quasi-Wideness. In *17th International Symposium on Experimental Algorithms, SEA*, pages 14:1–14:16, 2018.

54 J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In *Proc. of ICALP 2009*, pages 713–725, 2009.

55 Jaroslav Nešetřil and Patrice Ossona de Mendez. Linear time low tree-width partitions and algorithmic consequences. In *Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing (STOC)*, pages 391–400. ACM, 2006.

56 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

57 Jaroslav Nešetřil and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(03):868–887, 2010.

58 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

59 Jan Obdržálek. DAG-width: connectivity measure for directed graphs. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 814–821. Society for Industrial and Applied Mathematics, 2006.

60 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms (TALG)*, 9(1):11, 2012.

61 MichałPilipczuk and Sebastian Siebertz. Kernelization and approximation of distance-r independent sets on nowhere dense graphs. *arXiv preprint arXiv:1809.05675*, 2018.

62 MichałPilipczuk, Sebastian Siebertz, and Szymon Torunczyk. On the number of types in sparse graphs. *arXiv preprint arXiv:1705.09336*, 2017.

63 Hans Jürgen Prömel and Angelika Steger. *The Steiner tree problem: a tour through graphs, algorithms, and complexity*. Springer Science & Business Media, 2012.

64 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.

65 Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *Eur. J. Comb.*, 75:152–168, 2019.

66 N. Robertson and P.D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

67 Mohammad Ali Safari. D-width: A more natural measure for directed tree width. In *International Symposium on Mathematical Foundations of Computer Science*, pages 745–756. Springer, 2005.

68 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

69 Luc Segoufin and Alexandre Vigny. Constant Delay Enumeration for FO Queries over Databases with Local Bounded Expansion. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory (ICDT 2017)*, volume 68 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICDT.2017.20.

70 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

71 Sebastian Siebertz. Reconfiguration on Nowhere Dense Graph Classes. *Electr. J. Comb.*, 25(3):P3.24, 2018.

72 VN Vapnik and A Ya Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Measures of Complexity*, 16(2):11, 1971.

73 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Mathematics*, 309(18):5562–5568, 2009.

Tree Automata with Global Constraints for Infinite Trees

Patrick Landwehr
RWTH Aachen University, Germany
landwehr@cs.rwth-aachen.de

Christof Löding
RWTH Aachen University, Germany
loeding@cs.rwth-aachen.de

Abstract

We study an extension of tree automata on infinite trees with global equality and disequality constraints. These constraints can enforce that all subtrees for which in the accepting run a state q is reached (at the root of that subtree) are identical, or that these trees differ from the subtrees at which a state q' is reached. We consider the closure properties of this model and its decision problems. While the emptiness problem for the general model remains open, we show the decidability of the emptiness problem for the case that the given automaton only uses equality constraints.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Tree automata, infinite trees, global constraints

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.47

Acknowledgements We thank Arnaud Carayol for discussions on the subject.

1 Introduction

In this paper, we start the investigation of a model of finite automata on infinite trees with global equality and disequality constraints. These constraints are specified over the state space of the tree automaton: an equality constraint is of the form $q_1 \approx q_2$ for states q_1 and q_2 of the automaton, and a run satisfies the constraint if the subtrees at all nodes at which q_1 and q_2 appear in the run are equal. Similarly, a disequaltiy constraint $q_1 \not\approx q_2$ requires that the subtrees at q_1-positions in the run are all different from the subtrees at q_2-positions.

Originally, such a model with global constraints has been defined in [7] over finite trees for analysing a logic called TQL for querying semi-structured data [6]. The model (referred to as TAGED, tree automata with global equality and disequality constraints) has then been further investigated in [8, 9] because it turned out to be a useful tool for deciding logics whose expressive power goes beyond that of monadic second-order logic (MSO), the latter being equivalent to standard finite tree automata [21] (see also [23]). In [8, 9] closure properties of TAGED are studied, and decidability results for emptiness on restricted classes of TAGED have been obtained. These results have then been further generalised in [2] to the full class of TAGED (even enriched with arithmetic constraints). Besides the use of TAGED for the logic TQL, there are variants of monadic second-order logic (MSO) with equality and disequality tests that characterise the class of TAGED, and the algorithmic results lead to decision procedures for these versions of MSO [9, 2].

As mentioned above, we want to extend this theory to infinite trees. Automata on infinite trees were originally introduced for solving decision problems for MSO over infinite trees [18]. Since then, many algorithms and decision procedures based on these automata have been developed for solving problems in verification, synthesis, language equations, and set constraints (see [3, 14, 11, 1] for some examples).

While there are many extensions of finite automata over finite trees, e.g., automata with global (dis)equality constraints as described above, automata with (dis)equality constraints between siblings [4], automata over infinite alphabets [12, 25], or automata with counting constraints [13, 20], there is not much work on extended models for infinite trees. One reason for this is that extended automaton models over finite trees are motivated by problems coming from term rewriting or from XML document processing, and thus an extension to infinite trees does not seem to be useful. However, we believe that (dis)equality constraints over infinite trees can be used to express some interesting properties. For example, tree automata over infinite trees can represent sets word functions of the form $f : \Sigma_1^* \to \Sigma_2$ (for alphabets Σ_1, Σ_2). Such a function corresponds to a tree of branching degree $|\Sigma_1|$ with nodes labeled by Σ_2 (a node u in the tree can be identified with the directions one has to take from the root to reach this node, and thus corresponds to a word in Σ_1^*; its label is then the function value $f(u)$). This correspondence is used in algorithms for the synthesis of such functions from logical specifications [19, 17, 14]. In this setting, one can use equality constraints to model, for example, resets of synthesized functions (if $w \in \Sigma^*$ corresponds to an input string that models a reset, then the function should behave the same as from the root of the tree, which can be expressed by a global equality constraint in a tree automaton).

A first step for developing a theory of automata on infinite trees with (dis)equality constraints (we only use the term constraints in the following to refer to (dis)equality constraints) was made in [5], where automata on infinite trees with local constraints are analyzed. It is shown that parity tree automata with constraints for direct subtrees of a node have a decidable emptiness problem and the languages recognized by these automata form a Boolean algebra. In [15] it is shown that these automata with a Büchi condition are closed under projection, and that they can be used to decide satisfiability of MSO formulas enriched with a predicate for testing (dis)equality of direct subtrees of a node.

We continue this line of research and analyze automata on infinite trees with global constraints. We show that (similar to the case of finite trees) the class of languages recognizable by this model is closed under union and intersection, but not under complement. We compare different acceptance conditions and prove - among other results - that the equivalence in expressive power between parity- and Muller-acceptance (see [23]) extends to automata with global constraints. We also consider a few decision problems. To be precise, we show the undecidability of the universality- and the regularity-problem for tree automata with global constraints. (Again, analogous results for finite trees have been shown in [9, 2].)

The main result of this paper is the decidability of the emptiness problem if we restrict the automaton to only use equality constraints. It relies on a construction of a parity automaton that only uses reflexive equality constraints of the form $q \approx q$. This construction is the core of the decidability result because the game-based emptiness algorithm for parity tree automata (see [26] or [23]) also solves the emptiness problem in the case of reflexive equality constraints (using memoryless determinacy of parity games).

For the whole class of tree automata with global constraints on infinite trees, the decidability of the emptiness problem is still an open question. The approaches that have been developed for finite trees do not work on infinite trees. For example, in infinite trees it is possible that a tree is equal to one of its subtrees, which is impossible in finite trees.

This paper is structured as follows: In Section 2, we introduce basic notations and formally define the automaton model. Then in Section 3, we discuss closure properties of the classes of languages recognizable by tree automata with global constraints, and we analyze the expressive power of different restricted models (with regard to acceptance condition and the form of the constraints). In Section 4, we first present undecidability results for the universality and regularity problem (which easily generalize from finite trees). We then show that the emptiness problem for parity tree automata with only global equality constraints, is decidable. We give some concluding remarks in Section 5.

2 Preliminaries

2.1 Trees, Languages and Tree Automata

An *alphabet* is a finite set Σ of letters, Σ^* denotes the set of all finite words over Σ, whereas Σ^ω is the set of infinite words over Σ. The *empty word* is notated by ε.

Let Σ be some finite alphabet. An *infinite Σ-labelled (binary) tree* is a mapping $t : \{0,1\}^* \to \Sigma$. The elements of $\{0,1\}^*$ are called *nodes* and the node ε is called the *root*. We sometimes write dom_t instead of $\{0,1\}^*$. For any node u the node $u0$ is called the *left child* of u and $u1$ is the *right child* of u. A *branch* is an infinite word in $\{0,1\}^\omega$. The set of all infinite Σ-labelled trees is denoted by T_Σ^ω.

Let t be a σ-labelled tree. And let $u \in \{0,1\}^*$ be a node then $t|_u$ denotes the *subtree* of t rooted at u. That is $t|_u(v) := t(uv)$ for all $v \in \{0,1\}^*$.

A *(non-deterministic) tree automaton* for infinite trees over the alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, \mathrm{Acc})$ where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation, $q_0 \in Q$ is the initial state and $\mathrm{Acc} \subseteq Q^\omega$ is the acceptance condition.

Let t be an infinite Σ-labelled tree. A *run* of $\mathcal{A}$ on t is a Q-labelled tree ρ such that $\rho(\varepsilon) = q_0$ and for every node $u \in \{0,1\}^*$, $(\rho(u), t(u), \rho(u0), \rho(u1)) \in \Delta$. A run ρ is *accepting* if for every branch $\alpha_1\alpha_2 \ldots \in \{0,1\}^\omega$, the sequence of states $\rho(\varepsilon)\rho(\alpha_1)\rho(\alpha_1\alpha_2)\ldots$ forms an infinite word in Acc. A tree t is accepted by $\mathcal{A}$ if there exists an accepting run ρ of $\mathcal{A}$ on t. The language recognized by $\mathcal{A}$ is defined as $L(\mathcal{A}) := \{t \in T_\Sigma^\omega \mid \mathcal{A} \text{ accepts } t\}$.

Several forms of acceptance conditions appear in the literature. In this paper we mainly focus on so called safety-, Büchi-, parity- and Muller-acceptance:

- If the tree automaton has a *safety* acceptance condition, it holds $\mathrm{Acc} = Q^\omega$. That is every run is accepting.
- For a *Büchi* acceptance condition a subset $F \subseteq Q$ of states is given and an infinite sequence of states $q_1q_2q_3\ldots \in Q^\omega$ is in Acc if and only if there are infinitely many positions $i \in \mathbb{N}$ with $q_i \in F$.
- In the case of a *parity* acceptance condition a mapping $p : Q \to \mathbb{N}$ from states to natural numbers is given and an infinite sequence of states $q_1q_2q_3\ldots \in Q^\omega$ is in Acc if and only if the highest number $p(q_i)$ that appears infinitely often for that sequence is even.
- A *Muller* acceptance condition is given by a set $\mathcal{F} \subseteq 2^Q$ and an infinite sequence of states $q_1q_2q_3\ldots \in Q^\omega$ is in Acc if and only if the set of states that appear infinitely often in that sequence is an element of $\mathcal{F}$.

A language $L \subseteq T_\Sigma^\omega$ is called *regular* if it can be accepted by a parity tree automaton. As a shortened notation, if $\mathcal{A}$ is an automaton, we write $Q_\mathcal{A}$ to denote the set of states of $\mathcal{A}$ and $\Delta_\mathcal{A}$ to denote the transition relation of $\mathcal{A}$.

Even though we present our results only for binary trees, the statements in this paper also hold true for ranked trees with higher branching degree.

2.2 Tree Automata with Global Equality Constraints

A *tree automaton with global constraints (TAGC)* is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, \mathrm{Acc}, C)$, such that $(Q, \Sigma, \Delta, q_0, \mathrm{Acc})$ is a tree automaton, denoted by $\mathsf{ta}(\mathcal{A})$, and C is a Boolean combination of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, where $q, q' \in Q$.

A *run* ρ of the TAGC $\mathcal{A}$ on a tree $t \in T_\Sigma^\omega$ is a run of $\mathsf{ta}(\mathcal{A})$ such that ρ satisfies $C_\mathcal{A}$ (denoted by $(t, \rho) \models C_\mathcal{A}$ or $\rho \models C_\mathcal{A}$ if t is clear from the context), where the satisfiability of constraints is defined as follows: For atomic constraints $\rho \models q \approx q'$ holds (resp. $\rho \models q \not\approx q'$

holds), if and only if for all nodes $u, u' \in \mathrm{dom}_t$ with $u \neq u'$, $\rho(u) = q$ and $\rho(u') = q'$, it holds $t|_u = t|_{u'}$ (or $t|_u \neq t|_{u'}$ resp.). This notion of satisfiability is extended to Boolean combinations as usual. In particular, a run satisfies a negated atomic constraint $\neg(q \approx q')$ if there exist two nodes u, u' with $\rho(u) = q$, $\rho(u') = q'$, and $t|_u = t|_{u'}$. So it is important to note that the semantics of $(q \not\approx q')$ is different from $\neg(q \approx q')$ because of the quantifier over the tree nodes.

A run ρ is *accepting* if it is accepting for $\mathsf{ta}(\mathcal{A})$. As usual, the language recognized by $\mathcal{A}$ is the set $L(\mathcal{A})$ of trees $t \in T^\omega_\Sigma$ for which there exists a accepting run of $\mathcal{A}$ on t. Two automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if they recognize the same language.

Before we give some examples, we define some classes of TAGC with restricted forms of constraints that are used in the later sections. We follow the terminology for TAGC on finite trees as presented in [2] (the model called TAGED in [9] has a slightly different definition, and the notion of 'positive' in [9] differs from the one in [2]). A TAGC is called *positive (PTAGC)* if C is a positive Boolean combination (i.e. it does not use the negation symbol $\neg$ in the Boolean combination of atomic constraints). Note that atomic constraints of the form $(q \not\approx q')$ can be used in PTAGC because they do not use the negation symbol. In a *conjunctive* TAGC, the constraint is a single conjunction of possibly negated atomic constraints. Accordingly, a conjunctive PTAGC has only one conjunction of atomic constraints. A PTAGC is called *rigid* if it is conjunctive and uses only reflexive equality constraints, that is, C is just a single conjunction of atomic constraints, and for each atomic constraint of the form $q \approx q'$, we have $q = q'$ (the term rigid comes from [10]). For some constructions it is helpful if the initial state does not appear in any of the constraints. We call a TAGC that satisfies this restriction *simplified*.

▶ **Example 1.** Let $\Sigma = \{a, b\}$, then the set $\{t \in T^\omega_\Sigma \mid t|_0 = t|_1\}$ is a tree language, which is well known not to be regular. However it is recognized by the safety-PTAGC $\mathcal{A} := (\{q_0, q_1, ☺\}, \Sigma, \Delta, q_0, q_1 \approx q_1)$, where $\Delta := \{(q_0, x, q_1, q_1) \mid x \in \Sigma\} \cup \{(q, x, ☺, ☺) \mid q \in \{q_1, ☺\}, x \in \Sigma\}$.

In example 1 the automaton did not make use of the "globality" of the constraints. Therefore we present an additional example.

▶ **Example 2.** The language of all trees $t \in T^\omega_\Sigma$ that do not contain themselves as a proper subtree is recognized by the Büchi-PTAGC $\mathcal{A} := (\{q_0, q_1\}, \Sigma, \Delta, q_0, \{q_1\}, q_0 \not\approx q_1)$ where $\Delta := \{(q, x, q_1, q_1) \mid q \in \{q_0, q_1\}, x \in \Sigma\}$.

In both examples we presented PTAGC. The following example uses a negated constraint and also illustrates an interesting interplay between constraints and acceptance conditions:

▶ **Example 3.** Consider the constraint formula $\neg(q_0 \approx q_1 \wedge q_0 \not\approx q_1)$ and note that this formula is not a tautology. If for example the state q_1 does not appear in a run, then both $q_0 \approx q_1$ as well as $q_0 \not\approx q_1$ are satisfied. Thus to satisfy the whole formula, both q_0 and q_1 have to appear in the run. We now use this constraint formula in an automaton:

$$\mathcal{A} := (\{q_0, q_1, ☺\}, \{a, b, c\}, \Delta, q_0, \neg(q_0 \approx q_1 \wedge q_0 \not\approx q_1)) \text{ , where}$$

$$\Delta := \{(q_0, x, q_0, ☺), (q_0, x, ☺, q_0), (q_0, x, q_1, ☺), (q_0, x, ☺, q_1), (☺, x, ☺, ☺) \mid x \in \{a, b, c\}\} \cup \{(q_1, a, ☺, ☺)\}.$$

This safety TAGC starts in q_0, guesses a position at which q_1 appears and checks whether the label at that position is an a. Thus $\mathcal{A}$ accepts precisely those trees over the alphabet

$\{a, b, c\}$, that contain a position labelled with a. This regular language cannot be recognized by a standard safety tree automaton, thus showing that negated constraints actually can introduce stronger acceptance conditions even inside the class of regular languages.

The following examples show some possible applications of TAGC:

▶ **Example 4.** As explained in the introduction, infinite trees can be used to model functions of the form $f : \Sigma_1^* \to \Sigma_2$ for alphabets Σ_1, Σ_2, referred to as input and output alphabets, respectively. We can apply f to infinite input words $\alpha = a_1 a_2 \cdots$ obtaining $f(\alpha) = b_0 b_1 \cdots$ with $b_i = f(a_1 \cdots a_i)$ (with the special case $b_0 = f(\varepsilon)$). In the synthesis problem for such functions, we are given a relation $S \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, called the specification. A function f satisfies the specification if $(\alpha, f(\alpha)) \in S$ for all input words α. The task is to automatically synthesize a function satisfying S from a definition of S by a logical formula or an automaton.

Note that for $\Sigma_1 = \{0, 1\}$, the function f has exactly the same shape as an infinite binary tree (for larger alphabets Σ_1, one can use trees of higher branching degree). Thus, we can identify trees with the functions described above.

As shown in [19, 17] (see also [14]), a specification S defined in MSO logic or linear temporal logic (LTL) over infinite words, can be translated into a tree automaton $\mathcal{A}_S$ accepting precisely those trees that satisfy the specification. The emptiness test for tree automata then yields a tree (and thus a function) satisfying the specification.

Using TAGC, we can model additional properties of such functions. Assume, for example, that we are given a regular language $R \subseteq \Sigma_1^*$ of finite words over the input alphabet that model a reset of the system. The requirement for the function to be synthesized could be that it satisfies the specification S, and whenever the input string processed so far corresponds to a reset in R, then the function should behave in the same way (it moves to a specific reset state and forgets about the past). We can express this property by taking the product of the tree automaton $\mathcal{A}_S$ with a DFA $\mathcal{D}_R$ for the reset words, and impose equality constraints for all product states (q, p) such that p is a final state of $\mathcal{D}_R$. A tree accepted by this product automaton corresponds to a function with the desired reset behavior, and can by synthesized by the methods presented in Section 4.

▶ **Example 5.** Let L be a language of infinite words over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$. We say L is an *ω-power* if $L = U^\omega$ for a regular language U of finite words. It is a *deterministic ω-power* if U is prefix-free (i.e. for all $u \in U$ and v strict prefix of u, $v \notin U$). Here the term 'deterministic' is appropriate, because if U is prefix-free then every word in L has a unique factorization into its U-factors. In general, regular ω-languages can be characterized in terms of concatenations and ω-powers of regular languages of finite words (see [22]). Similarly, deterministic ω-powers can be used to characterize the subclass of deterministic Büchi languages, and are used in [16] to characterize certain regular liveness properties.

We can construct for a given ω-regular language L a PTAGC, whose recognized language is non-empty if and only if L is a deterministic ω-power. We consider n-ary infinite $\{0, 1\}$-labelled trees, where the directions correspond to the n letters of Σ. The constructed parity-PTAGC $\mathcal{A}$ then verifies two conditions on such an input-tree:

- the branches on which infinitely many 1 appear are precisely the words in L.
- The whole tree is equal to all subtrees rooted at positions at which a 1 appears.

If $L = U^\omega$, then the tree t_U is accepted by $\mathcal{A}$, where $t_U(u) = 1$ iff $u \in U^*$. If on the other hand $\mathcal{A}$ accepts a tree, then it also accepts a regular tree t (see Corollary 23 in Section 4). Define U to be the set of all minimal (wrt. prefix order) words u with $t(u) = 1$. Then $L = U^\omega$ since t is accepted by $\mathcal{A}$. As a consequence, the problem whether a given regular ω-language is a deterministic ω-power, is decidable (using Theorem 22).

3 Properties

In this Section we first present some general properties of TAGC and then discuss their closure properties.

3.1 Expressive Power of different Acceptance Conditions

We start by comparing the expressive power of TAGC with different acceptance conditions. The following lemma is useful to convert between different acceptance conditions. The construction is a straightforward adaption of the one for tree automata without constraints commonly found in the literature.

▶ **Lemma 6.** *For a TAGC $\mathcal{A}$ and a deterministic ω-word automaton $\mathcal{B}$, one can construct an automaton $\mathcal{A} \ltimes \mathcal{B}$ that accepts all trees $t \in T_\Sigma^\omega$ for which there is a run ρ of $\mathcal{A}$ on t such that for each branch, the sequence of states along that branch in ρ are accepted by $\mathcal{B}$. The type of acceptance condition of $\mathcal{A} \ltimes \mathcal{B}$ (Büchi, parity, Muller) is the one of $\mathcal{B}$.*

Since all regular ω-languages can be accepted by deterministic parity automata (see, e.g., [23, 24]), we obtain the equivalence of TAGC with parity and Muller conditions.

▶ **Corollary 7.** *A language is recognizable by a parity-TAGC if and only if it is recognizable by a Muller-TAGC. And from a given parity/Muller Automaton one can construct an equivalent automaton with the other acceptance condition.*

3.2 Restricted Constraints

We now consider constructions for simplifying the shape of the constraints. Remember that we call a (P)TAGC conjunctive if its constraint formula is a conjunction of (possibly negated) atomic constraints. And we call a (P)TAGC simplified if its initial state does not appear in its constraint formula. Our goal in this section is to show the following lemma:

▶ **Lemma 8.** *For every Büchi- or parity-TAGC one can construct an equivalent conjunctive and simplified Büchi- or parity-PTAGC (respectively).*

Proof. Let $\mathcal{A}$ be a TAGC. We separate $\mathcal{A}$ into a set of conjunctive TAGC. Wlog. the constraint formula $C_\mathcal{A}$ is a disjunction of conjunctions $C_1, \ldots, C_k$ of possibly negated atomic constraints. Then $L(\mathcal{A}) = L(\mathcal{A}_1) \cup \ldots \cup L(\mathcal{A}_k)$ where $\mathcal{A}_j$ is obtained from $\mathcal{A}$ by replacing the constraint formula with C_j. Now all $\mathcal{A}_j$ are conjunctive. In the following we show in Lemma 9 how to obtain from a given conjunctive TAGC a set of conjunctive PTAGC. After that we show in Lemma 10 how to construct from a given conjunctive PTAGC a conjunctive and simplified PTAGC. Finally, according to Lemma 11 in Section 3.3, which states effective closure under union, we obtain the desired automaton. ◀

▶ **Lemma 9.** *For every conjunctive Büchi- or parity-TAGC $\mathcal{A}$ one can construct conjunctive Büchi- or parity-PTAGC $\mathcal{B}_1, \ldots, \mathcal{B}_n$ (respectively), such that $L(\mathcal{A}) = L(\mathcal{B}_1) \cup \ldots \cup L(\mathcal{B}_n)$.*

Proof sketch. The proof follows a similar idea to [2], where an analogous statement was proven for finite trees. The elimination of negated atomic constraints is based on the following idea. If we want the automaton to check a negated constraint $\neg(q \approx q')$, we can use copies $\hat{q}$, $\hat{q}'$ of the states that the automaton can use in exactly one position of the run in place of q and q'. With this modification, the constraint $\neg(q \approx q')$ can be rewritten as $(\hat{q} \not\approx \hat{q}')$. The general construction is a bit more technical because the automaton has to guess which negated constraints are checked, and then apply the above idea to these constraints in parallel. ◀

One can show, that the language of example 3, that is the language of all trees containing a node labelled with a, is not recognizable by safety-PTAGC. Therefore, the statement in Lemma 9 does not hold for safety automata.

▶ **Lemma 10.** *For every conjunctive safety-, Büchi- or parity-PTAGC $\mathcal{A}$ one can construct an equivalent conjunctive and simplified safety-, Büchi- or parity-PTAGC $\mathcal{B}$ (respectively).*

Proof sketch. The basic idea is to delay the constraints that compare a subtree with the full tree by one step. To be more precise, for a constraint $q_0 \approx q'$ involving the initial state of $\mathcal{A}$, the automaton $\mathcal{B}$ verifies that at each position $u \in \mathrm{dom}_t$ at which q' appears, the label $t(u)$ is identical to $t(\varepsilon)$, and it also verifies that the right subtree $t|_{u0}$ is equal to $t|_0$ as well as that the left subtree $t|_{u1}$ is equal to $t|_1$ (similarly for disequality constraints). ◀

3.3 Closure Properties

We now analyze the closure properties of tree automata with global constraints for infinite trees. As it turns out, the closure properties with regard to Boolean operations are identical to the case of finite trees ([2]). But the proofs sometimes require a few extra steps, since we use a model with a single initial state.

▶ **Lemma 11.** *The class of languages recognizable by conjunctive and simplified PTAGC is effectively closed under union.*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be conjunctive and simplified parity-PTAGC. Wlog. $Q_{\mathcal{A}_1} \cap Q_{\mathcal{A}_2} = \emptyset$. Define the automaton $\mathcal{A}_\cup := (Q, \Sigma, q_{\text{start}}, \Delta, p, C)$, where $Q := \{q_{\text{start}}\} \cup Q_{\mathcal{A}_1} \cup Q_{\mathcal{A}_2}$, $\Delta := \Delta_{\mathcal{A}_1} \cup \Delta_{\mathcal{A}_2} \cup \{(q_{\text{start}}, a, q', q'') \mid (q_0^{\mathcal{A}_1}, a, q', q'') \in \Delta_{\mathcal{A}_1} \vee (q_0^{\mathcal{A}_2}, a, q', q'') \in \Delta_{\mathcal{A}_2}\}$. For each $q \in Q_{\mathcal{A}_1}$, $p(q) := p_{\mathcal{A}_1}(q)$ and for each $q \in Q_{\mathcal{A}_2}$, $p(q) := p_{\mathcal{A}_2}(q)$. ($p(q_{\text{start}})$ can be defined arbitrarily.), and $C := C_{\mathcal{A}_1} \wedge C_{\mathcal{A}_2}$.

Note that $\mathcal{A}_\cup$ is indeed conjunctive and simplified and it holds that $L(\mathcal{A}_\cup) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ (since if $\mathcal{A}_i$ is positive, $C_{\mathcal{A}_i}$ is satisfied if no state form $Q_{\mathcal{A}_i}$ appears in the run). Also note that if $\mathcal{A}_1$ and $\mathcal{A}_2$ are safety- or Büchi-PTAGC, then so is $\mathcal{A}_\cup$. ◀

▶ **Theorem 12.** *The classes of languages recognizable by Büchi- and parity-TAGC are effectively closed under union and intersection, but they are not closed under complement.*

Proof sketch. The closure under union follows from Lemma 8 and Lemma 11. The closure under intersection can be shown by a standard product construction. For showing the non-closure under complement, we can closely follow the ideas from [9] for the corresponding result on finite trees. The idea is to take a language, for which the automaton would have to verify infinitely many independent equalities. On the other hand, to check membership for the complement language it suffices to guess only a single disequality. Concretely, we can show that for $\Sigma = \{a, b\}$, the language $L := \{t \in T_\Sigma^\omega \mid \forall i \in \mathbb{N} : t|_{0^i 10} = t|_{0^i 11}\}$ is not recognizable by a TAGC but its complement can be accepted by a Büchi-TAGC. ◀

Since Lemma 9 does not hold for safety automata, we have to prove closure properties in a different way in this case.

▶ **Theorem 13.** *The class of languages recognizable by safety-TAGC as well as the class of languages recognizable by safety-PTAGC is closed under union and intersection, but not under complement.*

Proof sketch. The proof in the case of PTAGC follows exactly the proof of Theorem 12. In the case of TAGC the we make use of the trick from example 3 (describing a reachability condition with the constraint formula). ◀

4 Decision Problems

In this section, we first present some undecidability results, and then show the decidability of the emptiness problem for PTAGC, which only use equality constraints.

4.1 Undecidability Results

The *universality problem* is to decide, given a TAGC $\mathcal{A}$ over Σ whether $L(\mathcal{A}) = T^\omega_\Sigma$.

The universality problem for tree automata with constraints on finite trees was shown to be undecidable in [8]. Therefore the following theorem is not surprising. In principle, we could give a reduction from the case of finite trees, using the results from [8] and [10]. However, one can also easily provide a direct proof by a reduction from the halting problem of two-register machines.

▶ **Theorem 14.** *The universality problem for Büchi-TAGC is undecidable, even for Büchi-PTAGC without disequality constraints.*

The *regularity problem* is to decide, given an automaton $\mathcal{A}$, whether $L(\mathcal{A})$ is regular. Barguñó et. al. have shown the undecidability of the regularity problem on finite trees.

▶ **Theorem 15** ([2])**.** *The regularity problem is undecidable for tree automata with global equality constraints for finite trees.*

▶ **Corollary 16.** *The regularity problem for Büchi-TAGC is undecidable.*

Proof Sketch. From a given tree automaton $\mathcal{A}$ with global equality constraints on finite trees (for a formal definition see eg. [8]) over the alphabet Σ construct a Büchi-TAGC $\mathcal{A}'$ over $\Sigma \cup \{\bot\}$ where $\bot$ is a new symbol, such that $\mathcal{A}'$ accepts precisely those trees $t' \in T^\omega_{\Sigma\cup\{\bot\}}$ for which there exists a tree $t \in L(\mathcal{A})$ with $t'(u) = t(u)$ for all $u \in \mathrm{dom}_t$ and $t'(u) = \bot$ for all $u \notin \mathrm{dom}_t$. Then $L(\mathcal{A}')$ is regular if and only if $L(\mathcal{A})$ is regular. ◀

4.2 Emptiness Problem

We now discuss the decidability of the emptiness problem for TAGC. To be precise, we prove that the emptiness problem for parity-PTAGC without $\not\approx$-constraints is decidable. We do this by a reduction to rigid equality constraints (recall that rigid PTAGC have only reflexive $\approx$-constraints).

We first note in Lemma 17 that for rigid parity-TAGC without $\not\approx$-constraints, the emptiness reduces to the case without constraints, which is known to be decidable (see, e.g., [23, 24]). In Lemma 20 we then present a construction to obtain a rigid automaton (this construction also works in the presence of non-reflexive disequality constraints, while Lemma 17 only works without disequality constraints). The construction in Lemma 20 relies on the existence of memoryless runs for conjunctive parity-PTAGC (to be defined below), which we prove in Lemma 18.

▶ **Lemma 17.** *For each rigid parity-TAGC $\mathcal{A}$ without $\not\approx$-constraints, $L(\mathcal{A}) = \emptyset$ if and only if $L(\mathsf{ta}(\mathcal{A})) = \emptyset$.*

Proof. It trivially holds that $L(\mathcal{A}) \subseteq L(\mathsf{ta}(\mathcal{A}))$. Thus we only have to show that $L(\mathsf{ta}(\mathcal{A})) \neq \emptyset$ implies $L(\mathcal{A}) \neq \emptyset$. Now let $L(\mathsf{ta}(\mathcal{A})) \neq \emptyset$. Then the standard emptiness game (see [23, 24]) for parity tree automata gives us a tree $t \in L(\mathsf{ta}(\mathcal{A}))$ with accepting run ρ such that for all $u, u' \in \mathsf{dom}_t$, $\rho(u) = \rho(u')$ implies $t|_u = t|_{u'}$. Therefore ρ is an accepting run of $\mathcal{A}$ as well, which implies $L(\mathcal{A}) \neq \emptyset$. ◀

Let $\mathcal{A}$ be a parity-PTAGC and $t \in T_\Sigma^\omega$ be a tree. Let ρ be a run of $\mathcal{A}$ on t. We say ρ is *memoryless* if for all nodes $u, u' \in \mathrm{dom}_t$ we have that $\rho(u) = \rho(u')$ and $t|_u = t|_{u'}$ implies $\rho|_u = \rho|_{u'}$. That is, if ρ is in the same state at two nodes with identical subtrees, then ρ is the same on these two subtrees.

▶ **Lemma 18.** *Let $\mathcal{A}$ be a parity-PTAGC and let $t \in L(\mathcal{A})$ be a tree accepted by $\mathcal{A}$. Then there exists a memoryless run of $\mathcal{A}$ on t.*

Proof sketch. A corresponding result is known for standard parity tree automata (see [26, end of Section 7]). The idea is to modify the membership game for an automaton $\mathcal{A}$ and a tree t (called coloring game in [26], Automaton-Pathfinder-Game in [24], and referred to as $\Gamma_{\mathcal{A},t}$ in [23]) by quotienting the game graph w.r.t. equal subtrees. A memoryless (also called positional) winning strategy for the player called "Automaton" in this new game then yields a run of the desired form. In the presence of constraints, we additionally need to start from an accepting run of $\mathcal{A}$, and restrict the game to the positions corresponding to the run. ◀

▶ **Example 19.** Lemma 18 requires the automaton $\mathcal{A}$ to be positive. This requirement is indeed neccessary, since there exists a TAGC that recognizes a non-empty language, but does not have a memoryless run: Define the safety-TAGC $\mathcal{A} := (\{q_0, q_1, q_2\}, \{a\}, q_0, \Delta, C)$ by setting $\Delta := \{(q_0, a, q_1, q_0), (q_0, a, q_2, q_0), (q_1, a, q_0, q_0), (q_2, a, q_0, q_0)\}$ and $C := \neg(q_0 \not\approx q_1) \wedge \neg(q_0 \not\approx q_2)$. For C to be satisfied on the only tree t over the alphabet $\{a\}$, both states q_1 and q_2 have to appear. But there is no memoryless run of $\mathcal{A}$ in which both states appear.

▶ **Lemma 20.** *For each conjunctive and simplified parity-PTAGC $\mathcal{A}$ without reflexive $\not\approx$-constraints, one can construct an equivalent rigid and simplified parity-PTAGC $\mathcal{A}'$.*

Proof. Let $\mathcal{A}$ be the given conjunctive and simplified parity-PTAGC. The automaton $\mathcal{A}'$ that we construct basically guesses a memoryless run of $\mathcal{A}$ on the input tree and verifies that it is indeed an accepting run. The difficulty is that $\mathcal{A}'$ can only use reflexive equality constraints in order to verify the equality constraints imposed by the guessed run of $\mathcal{A}$. Assume, for example, that $q \approx q'$ is a constraint of $\mathcal{A}$, and that q and q' appear in the run ρ of $\mathcal{A}$ that is guessed by $\mathcal{A}'$. In order to check equality of the subtrees at the positions in which q and q' occur in ρ, $\mathcal{A}'$ needs to go into the same state at all these positions. Thus, $\mathcal{A}'$ cannot distinguish anymore between positions with q and q'. However, at the same time, $\mathcal{A}'$ has to check that the parity condition is satisfied along all the branches of the guessed run ρ. For this purpose, we use the existence of memoryless runs, as guaranteed by Lemma 18. Since all subtrees at q positions in the run are the same, in a memoryless run, all subruns at q positions are also the same. We can speak of the run from q. Intuitively, the automaton $\mathcal{A}'$ guesses how these subruns from the constrained states are connected. That is, if in the run from q there is path that leads to q', and k is the maximal priority on this path, then $\mathcal{A}'$ stores this information. If, furthermore, in the run from q' there is a path to q with maximal priority k', then the maximum of k, k' has to be even (otherwise, the underlying run of $\mathcal{A}$ would contain a rejecting path, and $\mathcal{A}'$ has to guess an accepting run of $\mathcal{A}$).

Obviously, the problems and ideas described above are not restricted to pairs of states: If there is an additional constraint of the form $q' \approx q''$, and q, q', q'' all occur in the run, then the states q, q', q'' are in the same class. Formally, for a given subset X of the states, the equality constraints of $\mathcal{A}$ naturally define a relation over X. We take the symmetric and transitive closure of this relation. The class of a state q (in the set X) consists of all states that are related with q in this closure. Note that the class of q is empty if and only if q is not related to any other state from X by an equality constraint. We say that q is a constrained state (again w.r.t. X) if its class is nonempty.

The automaton $\mathcal{A}'$ guesses the set X of states occurring in the run of $\mathcal{A}$, for each class a set $S \subseteq X$ of states that can occur at a subtree corresponding to that class, and for all (S, q) and (S', q') with $q \in S$ and $q' \in S'$, how these states are connected in the subruns on the subtrees. The information about the connections is stored in a graph G. This is a fixed information (X, G) that is guessed at the beginning of the run and does not change anymore.

Then $\mathcal{A}'$ starts in the initial state of $\mathcal{A}$ and guesses a run of $\mathcal{A}$. When the guessed run enters a constrained state q, then $\mathcal{A}'$ goes to a state that starts simulating runs from all the states that can occur at the root of the subtree of q. Whenever one of the simulated runs enters a constrained state, then this is only possible if the other simulated runs agree with the information on the connection of subruns stored in G. In this case, the simulation is restarted from the set of the new constrained state.

Formally, let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, p, C)$ be a conjunctive and simplified parity-PTAGC. We start by defining the graphs that where informally explained above. $\mathbb{G}_{\mathcal{A}}$ is defined as the set of vertex- and edge-labelled directed graphs $G = (V_G, E_G, \nu_G, \mu_G)$ with

1A. vertex labels $\nu_G : V_G \to \{(S, q) \in 2^Q \times Q \mid q \in S\}$ (as explained above, a set S corresponds to the set of states that appear at the root of some subtree that is constrained by an equality constraint in the run of $\mathcal{A}$).

1B. Each two vertices have different labels, and for each vertex with label (S, q) and each $q' \in S$ there is a vertex labelled (S, q').

1C. For each $q \approx q' \in C$, whenever labels (S, q) and (S', q') appear, then $S = S'$ (each class is contained in a unique set). And for each vertex labelled (S, q) there are states $q', q'' \in S$ with $q' \approx q'' \in C$ (each set contains at least one class).

1D. For each $q \not\approx q' \in C$ with $q \neq q'$, there is no vertex labelled (S, q) with $q' \in S$ (states with a disequality constraint cannot appear at the root of the same subtree).

2A. edge labels $\mu_G : E_G \to (2^{p(Q)} \setminus \emptyset)$. (Intuitively, an edge label for an edge $(S, q) \to (S', q')$ encodes the allowed maximal priorities on the paths from q to q' in the run of $\mathcal{A}$.)

2B. for every cycle $e_1 e_2 \dots e_k$ with $e_i \in E_G$, and all $x_1, x_2, \dots, x_k \in p(Q)$ with $x_i \in \mu_G(e_i)$, we have $\max\{x_i \mid 1 \leq i \leq k\}$ is even (concatenating the paths encoded in the edges of G leads to an accepting path in the run of $\mathcal{A}$).

Note that properties 1A. and 1B. imply that $\mathbb{G}_{\mathcal{A}}$ is finite. These graphs are used to define "allowed transitions" such that the underlying guessed run of $\mathcal{A}$ satisfies the parity condition.

▶ **Example 21.** We give an example for such a graph G. Let $\mathcal{A} := (Q, \Sigma, \Delta, q_0, p, C)$ be the parity-PTAGC with

- $Q := \{q_0, q_1, q_1', q_2, q_2', q_3, q_4\}$
- $\Delta := \Big\{(q_0, x, q_1, q_1'),\ (q_1, x, q_1, q_4),\ (q_1', x, q_4, q_2),\ (q_2, x, q_3, q_3),$
  $(q_3, x, q_2', q_2'),\ (q_2', x, q_3, q_3)\ (q_4, x, q_4, q_4) \mid\ x \in \Sigma\Big\}$
- $p(q_0) = p(q_1) = p(q_1') = p(q_4) = 0$ and $p(q_2) = p(q_2') = 1$ and $p(q_3) = 2$
- $C := \ (q_1 \approx q_1') \wedge (q_2 \approx q_2')$

One possible graph in $\mathbb{G}_{\mathcal{A}}$ is depicted in Figure 1. The constructed automaton $\mathcal{A}'$ then can guess $X = Q$ as the set of states that are allowed to appear in the run. And it can guess the graph from Figure 1 to remember allowed transitions.

For the formal construction of $\mathcal{A}'$, we use the following notation. For each $S \subseteq Q$ we enumerate the states as $S = \{q_1^S, \dots, q_{|S|}^S\}$, where $q_j^S < q_{j+1}^S$ for all $j \leq |S|$ and $<$ is an arbitrary but fixed total order on Q.

Now define $\mathcal{A}' := (Q', \Sigma, \Delta', q_0', \mathrm{Acc}', C')$ where

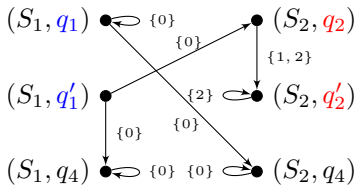


Figure 1 One possible graph from $\mathbb{G}_{\mathcal{A}}$ where $\mathcal{A}$ is the parity-PTAGC from example 21. Here S_1 is the set $\{q_1, q_1', q_4\}$ and S_2 is the set $\{q_2, q_2', q_4\}$. Note that every circle can produce only a maximum priority which is even.

- $Q' := \{q_I\} \cup (Q_{\text{state}} \times Q_{\text{fix}})$, where

$$Q_{\text{state}} \subseteq 2^Q \times \bigcup_{k \leq |Q|} (Q \times p(Q))^k \text{ and } Q_{\text{fix}} \subseteq 2^Q \times \mathbb{G}_{\mathcal{A}}$$

  such that $((S, \overline{w}), (X, G)) \in (Q_{\text{state}} \times Q_{\text{fix}})$ if and only if the following properties are satisfied (the intuition of the components is explained below):
  - Either $S = \{q_0\}$ or $S \subseteq Q$ such that there is a vertex labelled (S, q) in G for some (and therefore all) $q \in S$.
  - $\overline{w} \in (X \times p(X))^{|S|}$.
  - For each vertex labelled (S', q') in G, we have $q' \in X$ (and thus also $S' \subseteq X$).

  Informally, Q_{state} contains two kinds of information: The set for the most recently visited class, and a vector which contains for each element of that set a state and the highest priority visited up to now. This information is updated in every step. As explained earlier, Q_{fix} contains information that is guessed once at the very beginning and then remains unchanged for the remainder of the run.
- $q_0' := q_I$
- $\Delta' := \Delta_{\text{init}} \cup \Delta_{\text{trans}}$, where Δ_{init} contains the transitions that are applicable at the initial state, and Δ_{trans} contains all other transitions. We first define Δ_{trans}: Let $((S, \overline{w}), (X, G)) \in Q'$ be a state and let $a \in \Sigma$ be a letter. The automaton chooses for each entry $w_j = (q_j, p_j)$ in $\overline{w} = (w_1, \ldots, w_{|S|})$ a transition $(q_j, a, q_j^0, q_j^1) \in \Delta$, such that $q_j^i \in X$ for both $i \in \{0, 1\}$, and one of the following holds:
  - Either $\{q_j^i \mid j \leq |S|\} \cap \{q \in X \mid \exists q' \in X : q \approx q' \in C\} = \emptyset$. That is, none of the guessed successor states in direction i is a constrained state in X. In this case, the i-successor is $((S, \overline{w}'), (X, G))$, where $\overline{w}' := \big((q_1^i, \max(p_1, p(q_1^i))), \ldots, (q_{|S|}^i, \max(p_{|S|}, p(q_{|S|}^i)))\big)$.
  - Or there is a constrained state q_j^i and a vertex label (S', q_j^i) of G, such that
    - $S = \{q_0\}$ (the run enters a class for the first time), or
    - $S \neq \{q_0\}$, and for each $j \leq |S|$, the edge label $\mu(e_j)$ of the edge from (S, q_j^S) to (S', q_j^i) contains the priority p_j (the path segments of the simulated runs are encoded in G).

    In these cases, let $\overline{w_{S'}} := \big((q_1^{S'}, p(q_1^{S'})), \ldots, (q_{|S'|}^{S'}, p(q_{|S'|}^{S'}))\big)$, and the i-successor be the state $((S', \overline{w_{S'}}), (X, G))$. (The simulation of the runs is reset to the states in S'.)

  This fully defines Δ_{trans}. We conclude the definition of Δ' by setting

$$\Delta_{\text{init}} := \Big\{(q_I, a, P', P'') \mid \Big(\big(\big(\{q_0\}, ((q_0, p(q_0)))\big), (X, G)\big), a, P', P''\Big) \in \Delta_{\text{trans}}, (X, G) \in Q_{\text{fix}}\Big\}.$$

- Let $\mathbb{P} \subseteq Q'$ be the set of states $((S, \overline{w_S}), (X, G))$ with $\overline{w_S} := \big(\big(q_1^S, p(q_1^S)\big), \ldots, \big(q_{|S|}^S, p(q_{|S|}^S)\big)\big)$, which are used when a class is entered (see the second case in the definition of Δ_{trans}). By a slight abuse of notation, we say for $q \in Q$ and $P \in Q'$ that $q \in P$ if $P = ((S, \overline{w}), (X, G))$ and $\overline{w} = ((q_1, p_1), \ldots, (q_{|S|}, p_{|S|}))$ with $q \in \{q_1, \ldots, q_{|S|}\}$. Then define $C' := \bigwedge_{P \in \mathbb{P}} P \approx P \ \wedge \ \bigwedge_{P,P' \in Q', q \not\approx q' \in C, q \in P, q' \in P'} P \not\approx P'$.
- Acc′ should verify, that each branch is accepting. For each branch in a run, there are two possibilities: Either there are infinitely many states in $\mathbb{P}$ or not. If there are, then the allowed transitions in G verify that each branch of the run of $\mathcal{A}$ that is simulated in this branch of the run of $\mathcal{A}'$ was accepting. In the case that there are only finitely many states from $\mathbb{P}$, the acceptance condition has to verify that in each of the branches of the run of $\mathcal{A}$ that are simulated along this branch of the run of $\mathcal{A}'$, the maximum priority visited infinitely often is even.
  Instead of directly defining Acc′, we use an approach similar to Lemma 6. Here we have to be very careful to not reintroduce new irreflexive constraints. First note that it is possible to construct a deterministic parity word automaton, that reads the states along the branches in a run of $\mathcal{A}'$, and verifies that (assuming no additional states from $\mathbb{P}$ are read) all described paths of $\mathcal{A}$ satisfy the acceptance condition of $\mathcal{A}$. This can easily be seen if one constructs a nondeterministic automaton for the complement language, i.e. guess an index $j \in 1, \ldots, |Q|$, and check whether the highest priority along this branch is odd. This automaton then can be determinized and complemented (see [23]). Let this deterministic word automaton be $\mathcal{B} = (S, Q, s_0, \delta, p_{\mathcal{B}})$. Define the deterministic parity word automaton $\mathcal{B}' := (S \cup \{\hat{s}\}, Q, s_0, \delta_{\mathcal{B}'}, p_{\mathcal{B}'})$ where

$$\delta_{\mathcal{B}'}(s,q) := \begin{cases} \hat{s} & \text{if } q \in \mathbb{P} \\ s_0 & \text{if } q \notin \mathbb{P} \text{ and } s = \hat{s} \\ \delta(s,q) & \text{if } q \notin \mathbb{P} \text{ and } s \neq \hat{s} \end{cases},$$

$$p_{\mathcal{B}'}(s) := p_{\mathcal{B}}(s) \text{ for all } s \in S \ \text{ and } \ p_{\mathcal{B}'}(\hat{s}) := \begin{cases} \max(p(S)) & \text{if } \max(p(S)) \text{ is even} \\ \max(p(S)) + 1 & \text{if } \max(p(S)) \text{ is odd.} \end{cases}$$

  That is, $\mathcal{B}'$ behaves like $\mathcal{B}$ on states in $Q \setminus \mathbb{P}$, and whenever a state from $\mathbb{P}$ is read, it transitions into $\hat{s}$. Thus this automaton accepts precisely those sequences of states that satisfy the acceptance condition Acc′. We then construct the automaton $\mathcal{A}' \ltimes \mathcal{B}'$ according to Lemma 6. A closer look at the construction of $\mathcal{A}' \ltimes \mathcal{B}'$ reveals, that this did introduce non-reflexive $\approx$-constraints of the form $(q, s) \approx (q, s')$ for $q \in \mathbb{P}$ and $s \in S \cup \{\hat{s}\}$. But by the definition of $\mathcal{B}'$, $\delta_{\mathcal{B}'}(s, q) = \hat{s}$ for all $q \in \mathbb{P}, s \in S \cup \{\hat{s}\}$, and therefore states (q, s) with $q \in \mathbb{P}$ and $s \neq \hat{s}$ are unreachable. Removing these states from $\mathcal{A}' \ltimes \mathcal{B}'$ does not change the recognized language, but eliminates all non-reflexive $\approx$-constraints. ◀

It turns out that Lemma 20 does not hold, if we allow reflexive $\not\approx$-constraints. One can show, that the language $\{t \in T_\Sigma^\omega \mid t|_0 = t|_1 \text{ and } t|_{0^i} \neq t|_{0^j} \text{ for all } i \neq j\}$ is not recognizable by a PTAGC with reflexive $\approx$-constraints.

Lemma 8, Lemma 20 and Lemma 17 combined with the fact that the emptiness problem for parity tree automata without constraints is decidable, imply the following theorem:

▶ **Theorem 22.** *The emptiness problem for parity-PTAGC without $\not\approx$-constraints is decidable.*

In order to compute a concrete witness for the non-empiness, the following result is important.

Table 1 Summary of our closure and decidability results.

	safety-		Büchi-	parity = Muller
	PTAGC	TAGC	PTAGC = TAGC	PTAGC = TAGC
Union ($\cup$)	✓	✓	✓	✓
Intersection ($\cap$)	✓	✓	✓	✓
Complement ($\neg$)	×	×	×	×
Emptiness ($= \emptyset?$)	✓without $\not\approx$	?	✓PTAGC without $\not\approx$	✓PTAGC without $\not\approx$
Universality ($= T^{\omega}_{\Sigma}?$)	?	×	×	×
Regularity	?	×	×	×

▶ **Corollary 23.** *Each nonempty language recognizable by a parity-PTAGC without disequality constraints contains a regular tree.*[1]

Proof. This Corollary follows from the proof of Lemma 17, since the regular tree obtained from the membership game of standard tree automata (see [23, 24]) is also contained in the language of the parity-PTAGC. ◀

5 Conclusion

We have made a first step in extending the theory of TAGC from finite to infinite trees. The conversion of acceptance conditions works in the same way as for tree automata without constraints. Our results on closure and decidability properties of TAGC are summarized in Table 1. As shown in the table, the models of TAGC and PTAGC have the same expressive power for Büchi- and parity conditions. Safety conditions are not preserved by our construction of PTAGC. In particular, we conjecture that Lemma 9 does not hold for safety-TAGC. In Example 3 we present a language that is recognizable by a safety-TAGC that makes use of Boolean negation in its constraint formula. We are convinced that this language is not recognizable by safety-PTAGC, but currently we only have a proof in the case that we disallow reflexive $\not\approx$-constraints. Showing that this language is not recognizable by safety-PTAGC would also show that safety-PTAGC are not closed under complement.

The main open problem that remains, is the emptiness problem in the presence of disequality constraints. The pumping arguments that are used in the case of finite trees [2], do not (directly) generalize to infinite trees because an infinite tree can be equal to one of its subtrees. So it seems that one needs to develop new methods to deal with disequality constraints on infinite trees.

References

1 Franz Baader and Alexander Okhotin. Solving Language Equations and Disequations with Applications to Disunification in Description Logics and Monadic Set Constraints. In *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18*, volume 7180 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2012. doi:10.1007/978-3-642-28717-6.

2 Luis Barguñó, Carles Creus, Guillem Godoy, Florent Jacquemard, and Camille Vacher. The Emptiness Problem for Tree Automata with Global Constraints. In *Proceedings of the 25th*

[1] A regular tree is the unfolding of a finite graph, see [23, 24] for more precise definitions.

Annual IEEE Symposium on Logic in Computer Science, LICS 2010, pages 263–272. IEEE Computer Society, 2010.

3 Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. In *Proceedings of the 6th International Conference on Computer Aided Verification, CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 1994.

4 Bruno Bogaert and Sophie Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In *Proceedings of STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 1992.

5 Arnaud Carayol, Christof Löding, and Olivier Serre. Automata on Infinite Trees with Equality and Disequality Constraints Between Siblings. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 227–236, 2016. doi:10.1145/2933575.2934504.

6 Luca Cardelli and Giorgio Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004. doi:10.1017/S0960129504004141.

7 Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Satisfiability of a Spatial Logic with Tree Variables. In *Proceedings of Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2007. doi:10.1007/978-3-540-74915-8_13.

8 Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Tree Automata with Global Constraints. In *Proceedings of Developments in Language Theory, 12th International Conference, DLT 2008*, volume 5257 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2008. doi:10.1007/978-3-540-85780-8_25.

9 Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Tree Automata with Global Constraints. *Int. J. Found. Comput. Sci.*, 21(4):571–596, 2010. doi:10.1142/S012905411000743X.

10 Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata and applications. *Inf. Comput.*, 209(3):486–512, 2011. doi:10.1016/j.ic.2010.11.015.

11 Barbara Jobstmann and Roderick Bloem. Optimizations for LTL Synthesis. In *Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006*, pages 117–124. IEEE Computer Society, 2006.

12 Michael Kaminski and Tony Tan. Tree Automata over Infinite Alphabets. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008.

13 Felix Klaedtke and Harald Rueß. Monadic Second-Order Logics with Cardinalities. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 681–696. Springer, 2003.

14 Orna Kupferman and Moshe Y. Vardi. Safraless Decision Procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science, FOCS'05, Proceedings*, pages 531–542. IEEE Computer Society, 2005.

15 Patrick Landwehr and Christof Löding. Projection for Büchi Tree Automata with Constraints Between Siblings. In *Developments in Language Theory, DLT 2018*, volume 11088 of *Lecture Notes in Computer Science*, pages 478–490. Springer, 2018. doi:10.1007/978-3-319-98654-8.

16 Frank Nießner and Ulrich Ultes-Nitsche. A complete characterization of deterministic regular liveness properties. *Theor. Comput. Sci.*, 387(2):187–195, 2007. doi:10.1016/j.tcs.2007.07.038.

17 Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proceedings of the Symposium on Principles of Programming Languages, POPL'89*, pages 179–190, 1989.

18 Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

19 Michael O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston, MA, USA, 1972.

20 Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Numerical document queries. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 155–166. ACM, 2003.

21 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

22 Wolfgang Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 133–192. Elsevier Science Publishers, Amsterdam, 1990.

23 Wolfgang Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

24 Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In *Logic and automata - history and perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–724. Amsterdam University Press, 2007.

25 Margus Veanes and Nikolaj Bjørner. Symbolic tree automata. *Inf. Process. Lett.*, 115(3):418–424, 2015. doi:10.1016/j.ipl.2014.11.005.

26 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Constructive Discrepancy Minimization with Hereditary L2 Guarantees

Kasper Green Larsen
Department of Computer Science, Aarhus University, Denmark
larsen@cs.au.dk

Abstract

In discrepancy minimization problems, we are given a family of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, with each $S_i \in \mathcal{S}$ a subset of some universe $U = \{u_1, \ldots, u_n\}$ of n elements. The goal is to find a coloring $\chi : U \to \{-1, +1\}$ of the elements of U such that each set $S \in \mathcal{S}$ is colored as evenly as possible. Two classic measures of discrepancy are ℓ_∞-discrepancy defined as $\mathrm{disc}_\infty(\mathcal{S}, \chi) := \max_{S \in \mathcal{S}} |\sum_{u_i \in S} \chi(u_i)|$ and ℓ_2-discrepancy defined as $\mathrm{disc}_2(\mathcal{S}, \chi) := \sqrt{(1/|\mathcal{S}|) \sum_{S \in \mathcal{S}} \left(\sum_{u_i \in S} \chi(u_i)\right)^2}$. Breakthrough work by Bansal [FOCS'10] gave a polynomial time algorithm, based on rounding an SDP, for finding a coloring χ such that $\mathrm{disc}_\infty(\mathcal{S}, \chi) = O(\lg n \cdot \mathrm{herdisc}_\infty(\mathcal{S}))$ where $\mathrm{herdisc}_\infty(\mathcal{S})$ is the hereditary ℓ_∞-discrepancy of $\mathcal{S}$. We complement his work by giving a clean and simple $O((m+n)n^2)$ time algorithm for finding a coloring χ such $\mathrm{disc}_2(\mathcal{S}, \chi) = O(\sqrt{\lg n} \cdot \mathrm{herdisc}_2(\mathcal{S}))$ where $\mathrm{herdisc}_2(\mathcal{S})$ is the hereditary ℓ_2-discrepancy of $\mathcal{S}$. Interestingly, our algorithm avoids solving an SDP and instead relies simply on computing eigendecompositions of matrices. To prove that our algorithm has the claimed guarantees, we also prove new inequalities relating both $\mathrm{herdisc}_\infty$ and $\mathrm{herdisc}_2$ to the eigenvalues of the incidence matrix corresponding to $\mathcal{S}$. Our inequalities improve over previous work by Chazelle and Lvov [SCG'00] and by Matousek, Nikolov and Talwar [SODA'15+SCG'15]. We believe these inequalities are of independent interest as powerful tools for proving hereditary discrepancy lower bounds. Finally, we also implement our algorithm and show that it far outperforms random sampling of colorings in practice. Moreover, the algorithm finishes in a reasonable amount of time on matrices of sizes up to 10000×10000.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Discrepancy, Hereditary Discrepancy, Combinatorics, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.48

Funding This work is supported by a Villum Young Investigator Grant and an AUFF Starting Grant.

Acknowledgements The author wishes to thank Nikhil Bansal for useful discussions and pointers to relevant literature. The author also thanks an anonymous STOC reviewer for comments that simplified the **PartialColor** algorithm.

1 Introduction

Combinatorial discrepancy minimization is an important field with numerous applications in theoretical computer science, see e.g. the excellent books by Chazelle [9] and Matousek [16]. In discrepancy minimization problems, we are typically given a family of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, with each $S_i \in \mathcal{S}$ a subset of some universe $U = \{u_1, \ldots, u_n\}$ of n elements. The goal is to find a red-blue coloring of the elements of U such that each set $S \in \mathcal{S}$ is colored as evenly as possible. More formally, if we define the $m \times n$ incidence matrix A with $a_{i,j} = 1$ if $u_j \in S_i$ and $a_{i,j} = 0$ otherwise, then we seek a coloring $x \in \{-1, +1\}^n$ minimizing either the ℓ_∞-discrepancy $\mathrm{disc}_\infty(A, x) := \|Ax\|_\infty$ or the ℓ_2-discrepancy $\mathrm{disc}_2(A, x) = (1/\sqrt{m})\|Ax\|_2$. We say that the ℓ_∞-discrepancy of A is $\mathrm{disc}_\infty(A) := \min_{x \in \{-1,+1\}^n} \mathrm{disc}_\infty(A, x)$ and the

ℓ_2-discrepancy of A is $\mathrm{disc}_2(A) := \min_{x\in\{-1,+1\}^n} \mathrm{disc}_2(A,x)$. With this matrix view, it is clear that discrepancy minimization makes sense also for general matrices and not just ones arising from set systems.

Much research has been devoted to understanding both the ℓ_∞- and ℓ_2-discrepancy of various families of set systems and matrices. In particular set systems corresponding to incidences between geometric objects such as axis-aligned rectangles and points have been studied extensively, see e.g. [17, 15, 1, 11]. Another fruitful line of research has focused on general matrices, including the celebrated "Six Standard Devitations Suffice" result by Spencer [21], showing that any $n \times n$ matrix with $|a_{i,j}| \leq 1$ admits a coloring $x \in \{-1,+1\}^n$ such that $\mathrm{disc}_\infty(A,x) = O(\sqrt{n})$. Finding low discrepancy colorings for set systems where each element appears in at most t sets (the matrix A has at most t non-zeroes per column, all bounded by 1 in absolute value) has also received much attention. Beck and Fiala [7] gave a deterministic algorithm that finds a coloring x with $\mathrm{disc}_\infty(A,x) = O(t)$. Banaszczyk [2] improved this to $O(\sqrt{t \lg n})$ when $t \geq \lg n$. Determining whether a discrepancy of $O(\sqrt{t})$ can be achieved remains one of the biggest open problems in discrepancy minimization.

Constructive Discrepancy Minimization. Many of the original results, like Spencer's [21] and Banaszczyk's [2] were purely existential and it was not clear whether polynomial time algorithms finding such colorings were possible. In fact, Charikar et al. [8] presented very strong negative results in this direction. More concretely, they proved that it is NP-hard to even distinguish whether the ℓ_∞- or ℓ_2-discrepancy of an $n \times n$ set system is 0 or $\Omega(\sqrt{n})$. The first major breakthrough on the upper bound side was due to Bansal [3], who amongst others gave a polynomial time algorithm for finding a coloring matching the bounds by Spencer. Brilliant follow-up work by Lovett and Meka [14] gave simpler randomized algorithms achieving the same. A deterministic algorithm for Spencer's result was later given by Levy et al. [12]. A number of constructive algorithms were also given for the "sparse" set system case, finally resulting in polynomial time algorithms [4, 6, 5] matching the existential results by Banaszczyk.

Another very surprising result in Bansal's seminal paper [3] shows that, given a matrix A, one can find in polynomial time a coloring x achieving an ℓ_∞-discrepancy roughly bounded by the *hereditary* discrepancy of A. Hereditary discrepancy is a notion introduced by Lovász et al. [13] in order to prove discrepancy lower bounds. The hereditary ℓ_∞-discrepancy of a matrix A is defined $\mathrm{herdisc}_\infty(A) := \max_B \mathrm{disc}_\infty(B)$, where B ranges over all matrices obtained by removing a subset of the columns in A. In the terminology of set systems, the hereditary discrepancy is the maximum discrepancy over all set systems obtained by removing a subset of the elements in the universe. We also have an analogous definition for hereditary ℓ_2-discrepancy: $\mathrm{herdisc}_2(A) := \max_B \mathrm{disc}_2(B)$. Based on rounding an SDP, Bansal gave a polynomial time algorithm for finding a coloring x achieving $\mathrm{disc}_\infty(A,x) = O(\lg n \,\mathrm{herdisc}_\infty(A))$. This is quite surprising in light of the strong negative results by Charikar et al. [8], since it shows that is is in fact possible to find a low discrepancy coloring of an arbitrary matrix as long as all its submatrices have low discrepancy.

Our Results Overview. Our main algorithmic result is an ℓ_2 equivalent of Bansal's algorithm with hereditary guarantees. More concretely, we give a polynomial time algorithm for finding a coloring x such that $\mathrm{disc}_2(A,x) = O(\sqrt{\lg n} \cdot \mathrm{herdisc}_2(A))$. We note that neither our result nor Bansal's approximately imply the other: In one direction, the coloring x we find might have very low ℓ_2 discrepancy, but a very large value of $\|Ax\|_\infty$. In the other direction, $\mathrm{herdisc}_\infty(A)$ may be much larger than $\mathrm{herdisc}_2(A)$, thus Bansal's algorithm does not give any guarantees wrt. $\mathrm{herdisc}_2(A)$.

Our algorithm takes a very different approach than Bansal's in the sense that we completely avoid solving an SDP. Instead, we first prove a number of new inequalities relating $\text{herdisc}_2(A)$ and $\text{herdisc}_\infty(A)$ to the eigenvalues of A^TA. Relating hereditary discrepancy to the eigenvalues of A^TA was also done by Chazelle and Lvov [10] and by Matoušek et al. [18]. However the result by Chazelle and Lvov is too weak for our applications as it degenerates exponentially fast in the ratio between m and n. The result of Matoušek et al. could be used, but can only show that we find a coloring such that $\text{disc}_2(A,x) = O(\lg^{3/2} n \cdot \text{herdisc}_2(A))$. We believe our new inequalities are of independent interest as strong tools for proving discrepancy lower bounds.

With these inequalities established, we design a simple and efficient deterministic algorithm, inspired by Beck and Fiala's [7] algorithm for sparse set systems. Our key idea is to find a coloring x that is almost orthogonal to all the eigenvectors of A^TA corresponding to large eigenvalues. This in turn means that $\|Ax\|_2$ becomes bounded by $\text{herdisc}_2(A)$.

We now proceed to present the previous results for proving lower bounds on the hereditary discrepancy of matrices in order to set the stage for presenting our new results.

Previous Hereditary Discrepancy Bounds. One of the most useful tools in proving lower bounds for hereditary discrepancy is the determinant lower bound proved in the original paper introducing hereditary discrepancy:

▶ **Theorem 1** (Determinant Lower Bound (Lovász et al. [13])). *For an $m \times n$ real matrix A it holds that*

$$\text{herdisc}_\infty(A) \geq \max_k \max_B \frac{1}{2} |\det(B)|^{1/k},$$

where k ranges over all positive integers up to $\min\{n,m\}$ and B ranges over all $k \times k$ submatrices of A.

While it is easier to bound the max determinant of a submatrix B than it is to bound the discrepancy of a matrix directly, it still requires one to argue that we can find some B where all eigenvalues are non-zero. Chazelle and Lvov demonstrated how it suffices to bound the k'th largest eigenvalue of a matrix in order to derive hereditary discrepancy lower bounds:

▶ **Theorem 2** (Chazelle and Lvov [10]). *For an $m \times n$ real matrix A with $m \leq n$, let $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of A^TA. For any integer $k \leq m$, it holds that*

$$\text{herdisc}_\infty(A) \geq \frac{1}{2} 18^{-n/k} \sqrt{\lambda_k}.$$

The result of Chazelle and Lvov has two substantial caveats. First, it requires $m \leq n$. Since we will be using the *partial coloring* framework, we will end up with matrices having very few columns but many rows. This completely rules out using the above result for analysing our new algorithm. Since $k \leq m$, the lower bound also goes down exponentially fast in the gap between m and n (we note that Chazelle and Lvov didn't explicitly state that one needs $k \leq m$, but since $\text{rank}(A) \leq m$, we have $\lambda_k = 0$ whenever $k > m$).

Chazelle and Lvov used their eigenvalue bound to prove the following trace bound which has been very useful in the study of set systems corresponding to incidences between geometric objects:

▶ **Theorem 3** (Trace Bound (Chazelle and Lvov [10])). *For an $m \times n$ real matrix A with $m \leq n$, let $M = A^TA$. Then:*

$$\text{herdisc}_\infty(A) \geq \frac{1}{4} 324^{-n \,\text{tr}\, M^2 / \,\text{tr}^2\, M} \sqrt{\text{tr}\, M/n}.$$

Matoušek et al. [18] presented an alternative to the result of Chazelle and Lvov, relating $\text{herdisc}_\infty(A)$ and $\text{herdisc}_2(A)$ to the sum of singular values of A, i.e. they proved:

▶ **Theorem 4** (Matoušek et al. [18]). *For an $m \times n$ real matrix A, let $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. Then*

$$\text{herdisc}_\infty(A) \geq \text{herdisc}_2(A) = \Omega\left(\frac{1}{\lg n}\sum_{k=1}^{n}\sqrt{\frac{\lambda_k}{mn}}\right).$$

which for all positive integers $k \leq \min\{m,n\}$ implies:

$$\text{herdisc}_\infty(A) \geq \text{herdisc}_2(A) = \Omega\left(\frac{k}{\lg n}\sqrt{\frac{\lambda_k}{mn}}\right).$$

Comparing the bound to the result of Chazelle and Lvov, we see that the loss in terms of the ratio between k and n is much better. However for k, m and n all within a constant factor of each other, Chazelle and Lvov's bound implies $\text{herdisc}_\infty(A) = \Omega(\sqrt{\lambda_k})$ whereas the bound of Matoušek et al. loses a $\lg n$ factor and gives $\text{herdisc}_\infty(A) \geq \text{herdisc}_2(A) = \Omega(\sqrt{\lambda_k}/\lg n)$ (strictly speaking, the bound in terms of the sum of $\sqrt{\lambda_k}$'s is incomparable, but the bound only in terms of the k'th largest eigenvalue does lose this factor).

Our Results. We first give a new inequality relating $\text{herdisc}_\infty(A)$ to the eigenvalues of $A^T A$, simultaneously improving over the previous bounds by Chazelle and Lvov, and by Matoušek et al.:

▶ **Theorem 5.** *For an $m \times n$ real matrix A, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. For all positive integers $k \leq \min\{n,m\}$, we have*

$$\text{herdisc}_\infty(A) \geq \frac{k}{2e}\sqrt{\frac{\lambda_k}{mn}}.$$

Notice that our lower bound goes down as $k/\sqrt{mn}$ whereas Chazelle and Lvov's goes down as $18^{-n/k}$ and requires $m \leq n$. Thus our loss is exponentially better than theirs. Compared to the bound by Matoušek et al., we avoid the $\lg n$ loss (at least compared to the bound of Matoušek et al. that is only in terms of the k'th largest eigenvalue and not the sum of eigenvalues).

Re-executing Chazelle and Lvov's proof of the trace bound with the above lemma in place of theirs immediately gives a stronger version of the trace bound as well:

▶ **Corollary 6.** *For an $m \times n$ real matrix A, let $M = A^T A$. Then:*

$$\text{herdisc}_\infty(A) \geq \frac{\text{tr}^2 M}{8e\min\{n,m\}\,\text{tr}\,M^2}\sqrt{\frac{\text{tr}\,M}{\max\{m,n\}}}.$$

In establishing lower bounds on $\text{herdisc}_2(A)$ in terms of eigenvalues, we need to first prove an equivalent of the determinant lower bound for non-square matrices (and for ℓ_2-discrepancy rather than ℓ_∞):

▶ **Theorem 7.** *For an $m \times n$ real matrix A, we have*

$$\text{herdisc}_\infty(A) \geq \text{herdisc}_2(A) \geq \sqrt{\frac{n}{8\pi e m}}\det(A^T A)^{1/2n}.$$

We remark that proving Theorem 7 for the ℓ_∞-case appears as an exercise in [16] and we make no claim that the proof of Theorem 7 requires any new or deep insights (we suspect that it is folklore, but have not been able to find a mentioning of the above theorem in the literature). We finally arrive at our main result for lower bounding hereditary ℓ_2-discrepancy:

▶ **Corollary 8.** *For an $m \times n$ real matrix A, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. For all positive integers $k \leq \min\{n, m\}$, we have*

$$\text{herdisc}_2(A) \geq \frac{k}{e}\sqrt{\frac{\lambda_k}{8\pi mn}}.$$

We note that Theorem 5 actually follows (up to constant factors) from Corollary 8 using the fact that $\text{herdisc}_\infty(A) \geq \text{herdisc}_2(A)$, but we will present separate proofs of the two theorems since the direct proof of Theorem 5 is very short and crisp.

The exciting part in having established Corollary 8, is that it hints the direction for giving an efficient algorithm for obtaining colorings x with $\text{disc}_2(A, x)$ being bounded by some function of $\text{herdisc}_2(A)$. More concretely, we give an algorithm that is based on computing an eigendecomposition of $A^T A$ and using this to perform partial coloring that is orthogonal to the eigenvectors corresponding to the largest eigenvalues. Via Corollary 8, this gives a coloring with hereditary ℓ_2 guarantees. The precise guarantees of our algorithm are given in the following:

▶ **Theorem 9.** *There is an $O((m+n)n^2)$ time algorithm that given an $m \times n$ matrix A, computes a coloring $x \in \{-1, +1\}^n$ satisfying $\text{disc}_2(A, x) = O(\sqrt{\lg n} \cdot \text{herdisc}_2(A))$.*

We implemented our algorithm and performed various experiments to examine its practical performance. Section 4 shows that the algorithm far outperforms random sampling a coloring $x \in \{-1, +1\}^n$. In fact, it far outperforms random sampling, even if we repeatedly sample vectors for as long time as our algorithm runs and use the best one sampled. Moreover, the algorithm is efficient enough that it can be run on 1000×1000 matrices in less than 10 seconds and on matrices of sizes up to 10000×10000 in about 4 hours on a standard laptop. While it is conceivable that Bansal's SDP based approach can be modified to give ℓ_2 guarantees with a polynomial running time, it seems highly unlikely that it can process such large matrices in a reasonable amount of time. Moreover, our algorithm is much simpler to analyse and implement.

2 Eigenvalue Bounds for Hereditary Discrepancy

In this section, we prove new results relating the hereditary discrepancy of a matrix A to the eigenvalues of $A^T A$. The section is split in two parts, one studying hereditary ℓ_∞-discrepancy and one studying hereditary ℓ_2-discrepancy.

2.1 Hereditary ℓ_∞-discrepancy

Our first result concerns hereditary ℓ_∞-discrepancy and is a strengthening of the previous bound due to Chazelle and Lvov [10] (see Section 1). The simplest formulation is the following:

▶ **Restatement of Theorem 5.** *For an $m \times n$ real matrix A, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. For all positive integers $k \leq \min\{n, m\}$, we have*

$$\text{herdisc}_\infty(A) \geq \frac{k}{2e}\sqrt{\frac{\lambda_k}{mn}}.$$

Theorem 5 is an immediate corollary of the following slightly more general result:

▶ **Theorem 10.** *For an $m \times n$ real matrix A, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. For all positive integers $k \leq \min\{n, m\}$, we have*

$$\operatorname{herdisc}_\infty(A) \geq \frac{1}{2}\left(\frac{\prod_{i=1}^k \lambda_i}{\binom{n}{k}\binom{m}{k}}\right)^{1/2k}$$

Theorem 5 follows from Theorem 10 by using that $\binom{n}{k} \leq (en/k)^k$ and that $\prod_{i=1}^k \lambda_i \geq \lambda_k^k$. Thus our goal is to prove Theorem 10. The first step of our proof uses the following linear algebraic fact:

▶ **Lemma 11.** *For an $m \times n$ real matrix A, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. For all positive integers $k \leq n$, there exists an $m \times k$ submatrix C of A such that $\det(C^T C) \geq (\prod_{i=1}^k \lambda_i)/\binom{n}{k}$.*

Proof. The k'th symmetric function of $\lambda_1, \ldots, \lambda_n$ is defined as (see e.g. the textbook [19] p. 494): $s_k = \sum_{1 \leq i_1 < \cdots < i_k \leq n} \lambda_{i_1} \cdots \lambda_{i_k}$. Since all λ_i are non-negative, we have $s_k \geq \prod_{i=1}^k \lambda_i$. If we let $\mathcal{S}_k(A^T A)$ denote the set of all $k \times k$ principal submatrices of $A^T A$, then it also holds that (see e.g. the textbook [19] p. 494): $s_k = \sum_{B \in \mathcal{S}_k(A^T A)} \det(B)$. Since $|\mathcal{S}_k(A^T A)| = \binom{n}{k}$ there must be a $B \in \mathcal{S}_k(A^T A)$ for which $\det(B) \geq \left(\prod_{i=1}^k \lambda_i\right)/\binom{n}{k}$. Since B is a $k \times k$ principal submatrix of $A^T A$, it follows that there exists an $m \times k$ submatrix C of A such that $B = C^T C$ and thus $\det(C^T C) \geq \left(\prod_{i=1}^k \lambda_i\right)/\binom{n}{k}$. ◀

With Lemma 11 established, we are ready to present the proof of Theorem 10:

Proof of Theorem 10. Let A be a real $m \times n$ matrix and let $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of $A^T A$. From Lemma 11, it follows that for every $k \leq n$, there is an $m \times k$ submatrix C of A such that $\det(C^T C) \geq (\prod_{i=1}^k \lambda_i)/\binom{n}{k}$. If we also have $k \leq m$, we can let $\mathcal{S}_k(C)$ denote the set of all $k \times k$ principal submatrices of C and use the Cauchy-Binet formula to conclude that: $\det(C^T C) = \sum_{D \in \mathcal{S}_k(C)} \det(D)^2$. But $\mathcal{S}_k(C) \subseteq \mathcal{S}_k(A)$ hence there must exist a $k \times k$ matrix $D \in \mathcal{S}_k(A)$ such that

$$\det(D)^2 \geq \frac{\det(C^T C)}{|\mathcal{S}_k(C)|} \geq \frac{\prod_{i=1}^k \lambda_i}{\binom{n}{k}\binom{m}{k}} \Rightarrow |\det(D)| \geq \sqrt{\frac{\prod_{i=1}^k \lambda_i}{\binom{n}{k}\binom{m}{k}}}.$$

It follows from the determinant lower bound for hereditary discrepancy (Theorem 1) that

$$\operatorname{herdisc}_\infty(A) \geq \frac{1}{2}|\det(D)|^{1/k} \geq \frac{1}{2}\left(\frac{\prod_{i=1}^k \lambda_i}{\binom{n}{k}\binom{m}{k}}\right)^{1/2k}. \qquad ◀$$

Having established a stronger connection between eigenvalues and hereditary discrepancy than the one given by Chazelle and Lvov [10], we can also re-execute their proof of the trace bound and obtain the following strengthening:

▶ **Restatement of Corollary 6.** *For an $m \times n$ real matrix A, let $M = A^T A$. Then:*

$$\operatorname{herdisc}_\infty(A) \geq \frac{\operatorname{tr}^2 M}{8e \min\{n, m\} \operatorname{tr} M^2}\sqrt{\frac{\operatorname{tr} M}{\max\{m, n\}}}.$$

Proof. Let $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of M. Chazelle and Lvov [10] proved that if we choose $k = \operatorname{tr}^2 M/(2 \operatorname{tr} M^2)$ then $\lambda_k \geq \operatorname{tr} M/(4n)$. Examining their proof, one can in fact strengthen it slightly to $\lambda_k \geq \operatorname{tr} M/(4 \min\{m,n\})$ (their proof of ([10] Lemma 2.4) considers a uniform random eigenvalue λ amongst $\lambda_1, \ldots, \lambda_n$ and uses that $\operatorname{tr} M = n\mathbb{E}[\lambda]$. However, one needs only λ to be uniform random amongst the non-zero eigenvalues and there are at most $\min\{m,n\}$ such eigenvalues yielding $\operatorname{tr} M = \min\{n,m\}\mathbb{E}[\lambda]$). Inserting these bounds in Theorem 5 gives us

$$\operatorname{herdisc}_\infty(A) \geq \frac{\operatorname{tr}^2 M}{8e \operatorname{tr} M^2}\sqrt{\frac{\operatorname{tr} M}{mn \min\{m,n\}}} = \frac{\operatorname{tr}^2 M}{8e \min\{n,m\} \operatorname{tr} M^2}\sqrt{\frac{\operatorname{tr} M}{\max\{m,n\}}}. \qquad \blacktriangleleft$$

2.2 Hereditary ℓ_2-discrepancy

This section proves the following determinant result for hereditary ℓ_2-discrepancy of $m \times n$ matrices:

▶ **Restatement of Theorem 7.** *For an $m \times n$ real matrix A with $\det(A^T A) \neq 0$, we have*

$$\operatorname{herdisc}_\infty(A) \geq \operatorname{herdisc}_2(A) \geq \sqrt{\frac{nm}{8\pi e}} \det(A^T A)^{1/2n}.$$

The fact $\operatorname{herdisc}_\infty(A) \geq \operatorname{herdisc}_2(A)$ is true for all A, thus the difficulty in proving Theorem 7 lies in establishing that $\operatorname{herdisc}_2(A) \geq \sqrt{nm/(8\pi e)} \det(A^T A)^{1/2n}$. Our proof uses many of the ideas from the proof of the determinant lower bound (Theorem 1) in [13]. We start by introducing the linear discrepancy in the ℓ_2 setting and summarize known relations between linear discrepancy and hereditary discrepancy.

▶ **Definition 12.** *Let A be an $m \times n$ real matrix. Then its linear ℓ_2-discrepancy is defined as:*

$$\operatorname{lindisc}_2(A) := \max_{c \in [-1,+1]} \min_{x \in \{-1,+1\}^n} \frac{1}{\sqrt{m}} \|A(x-c)\|_2.$$

The linear ℓ_2-discrepancy has a clean geometric interpretation (this is a direct translation of the similar interpretation of linear ℓ_∞-discrepancy given e.g. in [13, 16]). For an $m \times n$ real matrix A, let: $U_A := \{x : \|Ax\|_2 \leq \sqrt{m}\}$. For $t > 0$, place 2^n translated copies $U_1, \ldots, U_{2^n}$ of tU_A such that there is one copy centered at each point in $\{-1,+1\}^n$. Then $\operatorname{lindisc}_2(A)$ is the least number t for which the sets U_j cover all of $[-1,+1]^n$.

We will need the following relationship between the hereditary and linear discrepancy:

▶ **Lemma 13** (Lovász et al. [13])**.** *For all $m \times n$ real matrices A, it holds that* $\operatorname{lindisc}_2(A) \leq 2 \operatorname{herdisc}_2(A)$.

We remark that [13] proved Lemma 13 only for the ℓ_∞-discrepancy, but their proof only uses the fact that $\{x : \|Ax\|_\infty \leq 1\}$ is centrally symmetric and convex (see [13] Lemma 1). The same is true for the U_A defined above.

In light of Lemma 13, we set out to lower bound the linear discrepancy of an $m \times n$ matrix A in terms of $\det(A^T A)$. We will prove the following lemma using an adaptation of the ideas in [13] (we have not been able to find a proof of this result elsewhere, but remark that the case of $m = n$ should follow by adapting the proof in [13]):

▶ **Lemma 14.** *Let A be an $m \times n$ real matrix with $\det(A^T A) \neq 0$. Then* $\operatorname{lindisc}_2(A) \geq \sqrt{n/(2\pi e m)} \det(A^T A)^{1/2n}$.

Proof. From the geometric interpretation given earlier, we know that if we place a copy of $\text{lindisc}_2(A)U_A$ on each point in $\{-1,+1\}^n$, then they cover all of $[-1,1]^n$ hence $\text{vol}(\text{lindisc}_2(A)U_A) \geq \text{vol}([-1,1]^n)/2^n = 1$. But

$$\begin{aligned}\text{vol}(\text{lindisc}_2(A)U_A) &= (\text{lindisc}_2(A))^n \,\text{vol}(U_A)\\ &= (\text{lindisc}_2(A))^n \,\text{vol}(\{x : \|Ax\|_2 \leq \sqrt{m}\})\\ &= (\text{lindisc}_2(A))^n \,\text{vol}(\{x : x^T A^T A x \leq m\}).\end{aligned}$$

Observe now that $\{x : x^T A^T A x \leq m\} = \{x : x^T (m^{-1} A^T A) x \leq 1\}$ is an ellipsoid. It is well-known that the volume of such an ellipsoid equals $v_n/\sqrt{\det(m^{-1}A^TA)} = v_n/\sqrt{m^{-n}\det(A^TA)}$ where v_n is the volume of the n-dimensional ℓ_2 unit ball. Since $v_n = \pi^{n/2}/\Gamma(n/2+1) \leq (2\pi e/n)^{n/2}$, we conclude:

$$\begin{aligned}&1 \leq \frac{(\text{lindisc}_2(A))^n v_n}{\sqrt{m^{-n}\det(A^TA)}} \Rightarrow\\ &1 \leq (\text{lindisc}_2(A))^n \left(\frac{2\pi e m}{n}\right)^{n/2} \frac{1}{\sqrt{\det(A^TA)}} \Rightarrow\\ &\text{lindisc}_2(A) \geq \sqrt{\frac{n}{2\pi e m}}\det(A^TA)^{1/2n}.\end{aligned}$$

◀

Combining Lemma 13 and Lemma 14 proves Theorem 7.

Having establishes Theorem 7, we are ready to prove our last result on hereditary ℓ_2-discrepancy:

▶ **Restatement of Corollary 8.** *For an $m \times n$ real matrix A, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ denote the eigenvalues of A^TA. For all positive integers $k \leq \min\{n,m\}$, we have* $\text{herdisc}_2(A) \geq (k/e)\sqrt{\lambda_k/(8\pi mn)}$.

Proof. Let A be an $m \times n$ real matrix and let $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ be the eigenvalues of A^TA. From Lemma 11, we know that for all $k \leq n$, there is an $m \times k$ submatrix C of A such that $\det(C^TC) \geq (\prod_{i=1}^k \lambda_i)/\binom{n}{k} \geq (k\lambda_k/(en))^k$. From Theorem 7, we get that $\text{herdisc}_2(C) \geq \sqrt{k/(8\pi em)}\det(C^TC)^{1/2k} \geq (k/e)\sqrt{\lambda_k/(8\pi mn)}$. Since C is obtained from A by deleting a subset of the columns, it follows that $\text{herdisc}_2(A) \geq \text{herdisc}_2(C)$, completing the proof. ◀

3 Discrepancy Minimization with Hereditary ℓ_2 Guarantees

This section gives our new algorithm for discrepancy minimization. The goal is to prove the following:

▶ **Restatement of Theorem 9.** *There is an $O((m+n)n^2)$ time algorithm that given an $m \times n$ matrix A, computes a coloring $x \in \{-1,+1\}^n$ satisfying* $\text{disc}_2(A,x) = O(\sqrt{\lg n} \cdot \text{herdisc}_2(A))$.

Our algorithm follows the same overall approach as several previous algorithms. The general setup is that we first give a procedure for partial coloring. This procedure takes a matrix A and a partial coloring $x \in [-1,+1]^n$. We say that coordinates i of x such that $|x_i| < 1$ are *live*. If there are k live coordinates prior to calling the partial coloring method, then upon termination we get a new vector γ such that the number of live coordinates in $\hat{x} = x + \gamma$ is no more than $k/2$. At the same time, all coordinates of $\hat{x}$ are bounded by 1 in absolute value and $\|A\hat{x}\|_2$ is not much larger than $\|Ax\|_2$.

We start by presenting the partial coloring algorithm and then show how to use it to get the final coloring.

3.1 Partial Coloring

In this section, we present our partial coloring algorithm. The algorithm takes as input an $m \times n$ matrix A and a vector $x \in [-1,+1]^n$. We think of the vector x as a partial coloring. We call a coordinate x_i of x *live* if $|x_i| < 1$ and we let k denote the number of live coordinates in x. For ease of notation, we let $\text{live}_x(i)$ denote the index of the i'th live coordinate in x and we define $\oplus_x : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^n$ as the function such that $a \oplus_x b$ for $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^k$, is the vector obtained from a by adding the i'th coordinate of b to the coordinate of index $\text{live}_x(i)$ in a (where $\text{live}_x(i)$ refers to the i'th live coordinate in x).

Upon termination, the algorithm returns another vector $\gamma \in \mathbb{R}^k$. If we let $\hat{x} = x \oplus_x \gamma$ be the vector in $\mathbb{R}^n$ obtained from x by adding γ_i to $x_{\text{live}_x(i)}$, then the partial coloring algorithm guarantees the following:

1. There are at most $k/2$ live coordinates in $\hat{x}$.
2. For all i, we have $|\hat{x}_i| \leq 1$.
3. $\|A\hat{x}\|_2^2 - \|Ax\|_2^2 = O(m(\text{herdisc}_2(A))^2)$.

Thus upon termination, the new vector $\hat{x}$ has half as many live coordinates, and the discrepancy did not increase by much. In particular the change is related to the hereditary ℓ_2-discrepancy of A.

The main idea in our algorithm is to use the connection between eigenvalues and hereditary ℓ_2-discrepancy that we proved in Corollary 8. Our algorithm proceeds in iterations, where in each step it finds a vector v and adds it to γ. The way we choose v is roughly to find the eigenvectors of A^TA and then pick v orthogonal to the eigenvectors corresponding to the largest eigenvalues. This bounds the difference $\|A(x \oplus_x (\gamma + v))\|_2 - \|A(x \oplus_x \gamma)\|_2$ in terms of the eigenvalues and thus hereditary ℓ_2-discrepancy. At the same time, we use the ideas by Beck and Fiala (and many later papers) where we include constraints forcing v orthogonal to e_i for every coordinate i that is not live. The algorithm is as follows:

PartialColor(A, x):

1. Let k denote the number of live coordinates in x and let C denote the $m \times k$ matrix obtained from A by deleting all columns corresponding to coordinates that are not live.
2. Initialize $\gamma = \mathbf{0} \in \mathbb{R}^k$.
3. Compute an eigendecomposition of C^TC to obtain the eigenvalues $\lambda_1 \geq \cdots \geq \lambda_k \geq 0$ and corresponding eigenvectors $\mu_1, \ldots, \mu_k$.
4. While **True**:
   a. Compute the set S of coordinates i such that $|\gamma_i + x_{\text{live}_x(i)}| = 1$. If $|S| \geq k/2$, **return** γ.
   b. Find a unit vector v orthogonal to all e_j with $j \in S$ and to all μ_i with $i \leq k/4$.
   c. Let $\sigma = -\operatorname{sign}(\langle Ax, A(\mathbf{0} \oplus_x v)\rangle)$. Compute the largest $\beta > 0$ such that all coordinates of $x \oplus_x (\gamma + \sigma\beta v)$ are less than or equal to 1 in absolute value. Update $\gamma \leftarrow \gamma + \sigma\beta v$.

Correctness. We prove that the vector γ returned by the above **PartialColor** algorithm satisfies the three claimed properties. First observe that in every iteration of the while loop, we find a vector v that is orthogonal to e_i whenever $|\gamma_i + x_{\text{live}_x(i)}| = 1$. Hence if $|\gamma_i + x_{\text{live}_x(i)}|$ becomes 1, it never changes again. Moreover, by maximizing β in each iteration, we guarantee that at least one more coordinate satisfies $|\gamma_i + x_{\text{live}_x(i)}| = 1$ after every iteration. Thus the algorithm terminates after at most $k/2$ iterations of the while loop and no coordinate of $x \oplus_x \gamma$ is larger than 1 in absolute value. What remains is to bound $\|A(x \oplus_x \gamma)\|_2^2 - \|Ax\|_2^2$.

Let $v^{(i)}$ denote the vector v found during the i'th iteration of the while loop. Upon termination, we have that $\gamma = \sigma_1\beta_1 v^{(1)} + \cdots + \sigma_r\beta_r v^{(r)}$ where $\sigma_i = -\operatorname{sign}(\langle Ax, v^{(i)}\rangle)$ and each $v^{(i)}$ is orthogonal to $\mu_1, \ldots, \mu_{k/4}$. Thus γ is also orthogonal to $\mu_1, \ldots, \mu_{k/4}$. We therefore have:

$$\begin{aligned}
\|A(x \oplus_x \gamma)\|_2^2 &= \|A(x + (\mathbf{0} \oplus_x \gamma))\|_2^2 \\
&\leq \|Ax\|_2^2 + \|A(\mathbf{0} \oplus_x \gamma)\|_2^2 + 2\langle Ax, A(\mathbf{0} \oplus_x \gamma)\rangle \\
&= \|Ax\|_2^2 + \|C\gamma\|_2^2 + 2\sum_{i=1}^{r}\langle Ax, A(\mathbf{0} \oplus_x \sigma_i\beta_i v^{(i)})\rangle \\
&\leq \|Ax\|_2^2 + \lambda_{k/4}\|\gamma\|_2^2 - 2\sum_{i=1}^{r}\operatorname{sign}(\langle Ax, A(\mathbf{0} \oplus_x v^{(i)})\rangle)\langle Ax, A(\mathbf{0} \oplus_x \beta_i v^{(i)})\rangle \\
&= \|Ax\|_2^2 + \lambda_{k/4}\|\gamma\|_2^2 - 2\sum_{i=1}^{r}\operatorname{sign}(\langle Ax, A(\mathbf{0} \oplus_x v^{(i)})\rangle)^2|\langle Ax, A(\mathbf{0} \oplus_x \beta_i v^{(i)})\rangle| \\
&\leq \|Ax\|_2^2 + \|\gamma\|_\infty^2 k\lambda_{k/4} - 0 \\
&\leq \|Ax\|_2^2 + 4k\lambda_{k/4}.
\end{aligned}$$

We would like to use Corollary 8 to relate $k\lambda_{k/4}$ to the hereditary discrepancy of A. Since C is an $m \times k$ submatrix of A, we have $\operatorname{herdisc}_2(A) \geq \operatorname{herdisc}_2(C)$. Using Corollary 8 we have $\operatorname{herdisc}_2(C) \geq (k/4e)\sqrt{\lambda_{k/4}/mk} = (1/4e)\sqrt{k\lambda_{k/4}/(8\pi)m}$. Hence we conclude that

$$\|A\hat{x}\|_2^2 - \|Ax\|_2^2 \leq 128e^2\pi m(\operatorname{herdisc}_2(A))^2 = O(m(\operatorname{herdisc}_2(A))^2).$$

Running Time. Step 1. of **PartialColor** takes $O(mk)$ time and step 2. takes $O(k)$. Step 3. takes $O(mk^2)$ time to compute C^TC (can be improved via fast matrix multiplication) and $O(k^3)$ time to compute the eigendecomposition. As argued above, each iteration of the while loop increases the size of S by at least one. Hence there are no more than $k/2$ iterations of the loop. Computing S in step (a) takes $O(k)$ time. Finding the unit vector v in step (b) can be done in $O(k^2)$ time as follows: Whenever adding a coordinate i to S, use Gram-Schmidt to compute the normalized (unit-norm) projection $\hat{e}_i$ of e_i onto the orthogonal complement of $\mu_1, \ldots, \mu_{k/4}$ and all previous vectors $\hat{e}_j$. This takes $O(k^2)$ time per i. To find v, sample a uniform random unit vector in $\mathbb{R}^k$ and run Gram-Schmidt to compute its projection onto the orthogonal complement of $\hat{e}_j$ for $j \in S$ and $\mu_1, \ldots, \mu_{k/4}$. The expected length of the projection is $\Omega(1)$ and we can scale it to unit length afterwards. This gives the desired vector. The Gram-Schmidt step takes $O(k^2)$ time. Computing $A(\mathbf{0} \oplus_x v)$ in step (c) takes $O(mk)$ time and computing Ax can be done outside the while loop in $O(mn)$ time. The inner product takes $O(m)$ time to compute. Computing β and adding $\sigma\beta v$ to γ takes $O(k)$ time. Overall, the **PartialColor** algorithm takes $O(mn + mk^2 + k^3)$ time. If Ax is given as argument to the algorithm, the time is further reduced to $O((m+k)k^2)$.

3.2 The Final Algorithm

Now that we have the **PartialColor** algorithm, getting to a low discrepancy coloring is straight forward. Given an $m \times n$ matrix A, we initialize $x \leftarrow \mathbf{0}$. We then repeatedly invoke **PartialColor**(A, x). Each call returns a vector γ. We update $x \leftarrow x + \gamma$ and continue. We stop once there are no live coordinates in x, i.e. all coordinates satisfy $|x_i| = 1$.

In each iteration, the number of live coordinates of i decreases by at least a factor two, and thus we are done after at most $\lg n$ iterations. This means that the final vector x satisfies

$$\begin{aligned}
\|Ax\|_2^2 &\leq \lg n \cdot O(m(\operatorname{herdisc}_2(A))^2) \Rightarrow \\
\|Ax\|_2 &= O(\sqrt{m\lg n} \cdot \operatorname{herdisc}_2(A)) \Rightarrow \\
\operatorname{disc}_2(A, x) &= O(\sqrt{\lg n} \cdot \operatorname{herdisc}_2(A)).
\end{aligned}$$

For the running time, observe that after each call to **PartialColor**, we can compute $A(x+\gamma)$ from Ax in $O(mk)$ time. Thus we can provide Ax as argument to **PartialColor** and thereby reduce its running time to $O((m+k)k^2)$. Since k halves in each iteration, we get a running time of

$$O\left(\sum_{i=1}^{\lg n}(m+n/2^i)(n/2^i)^2\right) = O((m+n)n^2).$$

This concludes the proof of Theorem 9.

4 Experiments

In this section, we present a number of experiments to test the practical performance of our discrepancy minimization algorithm. We denote the algorithm by L2Minimize in the following. We compare it to two base line algorithms Sample and SampleMany. Sample simply picks a uniform random $\{-1,+1\}$ vector as its coloring. SampleMany repeatedly samples a uniform random $\{-1,+1\}$ vector and runs for the same amount of time as L2Minimize. It returns the best vector found within the time limit.

The algorithms were implemented in Python, using NumPy and SciPy for linear algebra operations. All tests were run on a MacBook Pro (15-inch, Late 2013) running macOS Sierra 10.13.3. The machine has a 2 GHz Intel Core i7 and 8GB DDR3 RAM.

We tested the algorithms on three different classes of matrices:

- **Uniform** matrices: Each coordinate is uniform random and independently chosen among -1 and $+1$.
- **2D Corner** matrices: Obtained by sampling two sets $P = \{p_1,\ldots,p_n\}$ and $Q = \{q_1,\ldots,q_m\}$ of n and m points in the plane, respectively. The points are sampled uniformly in the $[0,1]\times[0,1]$ unit square. The resulting matrix has one column per point $p_j \in P$ and one row per point $q_i \in Q$. The entry (i,j) is 1 if p_j is *dominated* by q_i, i.e. $q_i.x > p_j.x$ and $q_i.y > p_j.y$ and it is 0 otherwise. Such matrices are known to have hereditary ℓ_2-discrepancy $O(\lg^{1.5} n)$ [20].
- **2D Halfspace** matrices: Obtained by sampling a set $P = \{p_1,\ldots,p_n\}$ of n points in the unit square $[0,1]\times[0,1]$, and a set Q of m halfspace. Each halfspace in Q is sampled by picking one point a uniformly on either the left boundary of the unit square or on the top boundary, and another point b uniformly on either the right boundary or the bottom boundary of the unit square. The halfspace is then chosen uniformly to be either everything above the line through a,b or everything below it. The resulting matrix has one column per point $p_j \in P$ and one row per halfspace $h_i \in Q$. The entry (i,j) is 1 if p_j is in the halfspace h_i and it is 0 otherwise. Such matrices are known to have hereditary ℓ_2-discrepancy $O(n^{1/4})$ [15].

Each test is run 10 times and the average ℓ_2 discrepancy and average runtime is reported. The running times of the algorithms varied exclusively with the matrix size and not the type of matrix, thus we only show one time column which is representative of all types of matrices. The results are shown in Table 1.

The table clearly shows that L2Minimize gives superior colorings for all types of matrices and all sizes. The tendency is particularly clear on the structured matrices **2D Corner** and **2D Halfspace** where the coloring found by L2Minimize on 10000×10000 matrices is a factor 25-30 smaller than a single round of random sampling (Sample) and a factor 5-7 better than random sampling for as long time as L2Minimize runs (SampleMany).

Table 1 Results of experiments with our L2Minimize algorithm. The Matrix Size column gives the size $m \times n$ of the input matrix. The Disc columns shows $\text{disc}_2(A, x) = \|Ax\|_2/\sqrt{m}$ for the coloring x found by the algorithm on the given type of matrix. Time is measured in seconds. Each entry is the average of 10 executions.

Algorithm	Matrix Size	Disc Uniform	Disc 2D Corner	Disc 2D Halfspace	Time (s)
L2Minimize	200×200	7.2	1.8	1.6	< 1
Sample	200×200	13.8	7.6	11.0	< 1
SampleMany	200×200	11.6	2.3	2.7	< 1
L2Minimize	1000×1000	15.7	1.9	2.3	9
Sample	1000×1000	31.6	16.0	18.3	< 1
SampleMany	1000×1000	28.9	4.9	5.5	9
L2Minimize	4000×4000	31.0	2.1	2.6	717
Sample	4000×4000	63.1	21.0	34.0	< 1
SampleMany	4000×4000	60.3	9.5	10.7	717
L2Minimize	10000×10000	48.3	2.1	3.1	15260
Sample	10000×10000	99.9	51.4	96.8	< 1
SampleMany	10000×10000	96.8	14.2	15.6	15260
L2Minimize	10000×2000	35.9	2.1	2.7	535
Sample	10000×2000	44.7	20.6	24.1	< 1
SampleMany	10000×2000	43.4	6.7	8.0	535
L2Minimize	2000×10000	21.4	1.8	2.0	5809
Sample	2000×10000	99.9	40.8	70.8	< 1
SampleMany	2000×10000	92.2	13.8	16.4	5809

The $O((m+n)n^2)$ running time makes the algorithm practical up to matrices of size about 10000×10000, at which point the algorithm runs for 15260 seconds $\approx$ 4 hours and 15 minutes.

References

1 R. Alexander. Geometric methods in the study of irregularities of distribution. *Combinatorica*, 10(2):115–136, 1990.

2 Wojciech Banaszczyk. Balancing vectors and Gaussian measures of n-dimensional convex bodies. *Random Structures & Algorithms*, 12:351–360, July 1998.

3 Nikhil Bansal. Constructive Algorithms for Discrepancy Minimization. In *Proc. 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 3–10, 2010.

4 Nikhil Bansal, Daniel Dadush, and Shashwat Garg. An Algorithm for Komlós Conjecture Matching Banaszczyk's Bound. In *Proc. 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS'16)*, pages 788–799, 2016.

5 Nikhil Bansal, Daniel Dadush, Shashwat Garg, and Shachar Lovett. The Gram-Schmidt Walk: A Cure for the Banaszczyk Blues. *CoRR*, abs/1708.01079, 2017. `arXiv:1708.01079`.

6 Nikhil Bansal and Shashwat Garg. Algorithmic Discrepancy Beyond Partial Coloring. In *Proc. 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, STOC 2017, pages 914–926, 2017.

7 J. Beck and T. Fiala. Integer-making theorems. *Discrete Applied Mathematics*, 3:1–8, February 1981.

8 Moses Charikar, Alantha Newman, and Aleksandar Nikolov. Tight Hardness Results for Minimizing Discrepancy. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 1607–1614, 2011.

9 Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, 2000.

10 Bernard Chazelle and Alexey Lvov. A Trace Bound for the Hereditary Discrepancy. In *Proc. 16th Annual Symposium on Computational Geometry*, SCG '00, pages 64–69, 2000.

11 Kasper Green Larsen. On Range Searching in the Group Model and Combinatorial Discrepancy. *SIAM Journal on Computing*, 43(2):673–686, 2014.

12 Avi Levy, Harishchandra Ramadas, and Thomas Rothvoss. Deterministic Discrepancy Minimization via the Multiplicative Weight Update Method. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 380–391, 2017.

13 L. Lovász, J. Spencer, and K. Vesztergombi. Discrepancy of Set-systems and Matrices. *European Journal of Combinatorics*, 7(2):151–160, 1986. doi:10.1016/S0195-6698(86)80041-5.

14 Shachar Lovett and Raghu Meka. Constructive Discrepancy Minimization by Walking on the Edges. *SIAM Journal on Computing*, 44(5):1573–1582, 2015.

15 J. Matoušek. Tight Upper Bounds for the Discrepancy of Half-Spaces. *Discrete and Computational Geometry*, 13:593–601, 1995.

16 J. Matousek. *Geometric Discrepancy: An Illustrated Guide*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 1999.

17 Jirí Matoušek and Aleksandar Nikolov. Combinatorial Discrepancy for Boxes via the gamma_2 Norm. In *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34, pages 1–15, 2015.

18 Jiří Matoušek, Aleksandar Nikolov, and Kunal Talwar. Factorization Norms and Hereditary Discrepancy. *CoRR*, abs/1408.1376, 2014. arXiv:1408.1376.

19 Carl D. Meyer, editor. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

20 Aleksandar Nikolov. Tighter bounds for the discrepancy of boxes and polytopes. *Mathematika*, 63:1091–1113, 2017.

21 Joel Spencer. Six standard deviations suffice. *Trans. Amer. Math. Soc.*, 289:679–706, 1985.

Quantum Advantage for the LOCAL Model in Distributed Computing

François Le Gall
Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan

Harumichi Nishimura
Graduate School of Informatics, Nagoya University
Chikusa-ku, Nagoya, Aichi 464-8601, Japan

Ansis Rosmanis
Centre for Quantum Technologies, National University of Singapore
Block S15, 3 Science Drive 2, 117543, Singapore

Abstract

There are two central models considered in (fault-free synchronous) distributed computing: the CONGEST model, in which communication channels have limited bandwidth, and the LOCAL model, in which communication channels have unlimited bandwidth. Very recently, Le Gall and Magniez (PODC 2018) showed the superiority of quantum distributed computing over classical distributed computing in the CONGEST model. In this work we show the superiority of quantum distributed computing in the LOCAL model: we exhibit two computational tasks that can be solved in a constant number of rounds in the quantum setting but require $\Omega(n)$ rounds in the classical (randomized) setting, where n denotes the size of the network.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum computing, distributed computing, LOCAL model

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.49

Acknowledgements FLG was partially supported by the JSPS KAKENHI grants No. 15H01677, No. 16H01705 and No. 16H05853. HN was partially supported by the JSPS KAKENHI grants No. 26247016, No. 16H01705 and No. 16K00015. AR was partially supported by the Singapore Ministry of Education and the National Research Foundation under grant R-710-000-012-135. Part of this work was done while AR was visiting Kyoto University, and AR would like to thank FLG for hospitality.

1 Introduction

Classical distributed computing

A central topic in distributed computing is the study of synchronous network algorithms. Here processors and communication channels are modeled using nodes and edges, respectively, and executions proceed with round-based synchrony, where each node can transfer one message to each adjacent node per round. The main quantity of interest is typically the number of rounds needed to solve a computational task. Two fundamental models considered in the literature are the LOCAL model, introduced by Linial [19, 20], and the CONGEST model, introduced by Peleg [22].

The LOCAL model does not put any limitation on the size of the messages sent at each round, and thus mainly characterizes the locality of the problem considered and the hardness of breaking symmetry between nodes. Obviously all computational problems can be solved with $O(D)$ rounds in the LOCAL model, where D is the diameter of the network, by first

collecting all the information about the network (including the inputs of all nodes) at some node. Many important problems have significantly more efficient algorithms – we refer to [22] for examples and to [8] for a recent classification.

In the CONGEST model, on the other hand, each message has restricted length (the length is typically restricted to $O(\log n)$ bits, where n is the number of nodes in the network). This corresponds to the situation of communication channels with limited bandwidth, in which case congestions can arise. A simple example showing the striking difference between these two models is deciding whether the diameter of the network is 2 or 3. This problem requires $\Theta(n)$ rounds in the CONGEST model [11, 16, 23], while in the LOCAL model it can be trivially solved with a constant number of rounds.

Quantum distributed computing

Quantum versions of both models can be naturally defined by replacing classical channels by quantum channels between the processors (which are now quantum processors, i.e., processors that can process quantum information). Gavoille, Kosowski and Markiewicz [12] first considered quantum distributed computing in the LOCAL model, and showed that for several fundamental problems, such as Graph Coloring or Maximal Independent Set, allowing quantum communication cannot lead to any significant advantage. More recently, Arfaoui and Fraigniaud [2] observed that several lower bound techniques for the classical LOCAL model hold in the quantum model as well.

The power of distributed network computation in the CONGEST model, where each node can send $O(\log n)$ qubits per round to each neighbor, has been first investigated by Elkin, Klauck, Nanongkai and Pandurangan [10]. The main conclusions reached in that paper were that for many fundamental problems in distributed computing, such as computing minimum spanning trees or minimum cuts, quantum communication does not, again, offer significant advantages over classical communication. Recently, Le Gall and Magniez nevertheless showed the superiority of quantum distributed computing in the CONGEST model for a concrete problem [18]: they showed that the diameter of the network can be computed in $\tilde{O}(\sqrt{nD})$ rounds in the quantum setting, where n is the number of nodes of the network and D is the diameter of the network. In comparison, as mentioned above $\Omega(n)$ rounds are necessary in the classical setting even if D is constant. It should be mentioned that from a purely complexity-theoretic perspective most known separations between two-party classical and quantum communication complexities (e.g., separations in the bounded-error setting for the disjointness function [1, 7, 17]) can be converted in a straightforward way into similar separations in the CONGEST model. The contribution of [18] is actually to give a separation for an important problem in distributed computing.

A pressing open question is to understand whether a similar quantum speedup in distributed computing can be obtained in the LOCAL model. The only known gap is a factor of 2: for each integer $t \geq 1$, Gavoille, Kosowski and Markiewicz [12] constructed a computational problem (inspired by the work by Greenberger, Horne and Zeilinger [13]) that can be solved in t rounds in the quantum setting but requires $2t$ rounds in the classical setting.[1] The quantum upper bound comes from the observation that t rounds are enough to create entanglement between two nodes at distance $2t$ from each other. In this perspective, as mentioned in [12], the speed-up factor of 2 may "look like a natural limit". Note that, contrary to

[1] A much larger gap is shown in [12] for the setting where the nodes of the network initially share a globally entangled state. In the present paper, however, we consider the arguably more natural setting where no prior entanglement is allowed.

the CONGEST model, known separations between two-party (or multiparty) quantum and classical communication complexities seem meaningless to prove separations in the LOCAL model due to the unlimited bandwidth between nodes.

Our results

In this work we show the existence of a large gap between the round complexities of quantum and classical (randomized) distributed computation in the LOCAL model.

▶ **Theorem 1.** *There exists a computational problem that can be solved with 2 rounds in the quantum LOCAL model, but requires $\Omega(n)$ rounds in the classical LOCAL model, where n denotes the number of nodes in the network.*

The computational problem we construct to prove Theorem 1 is inspired by a construction from [3], which was initially used to show the non-locality of measurement outcomes of graph states. The same construction was recently also used by Bravyi, Gosset and König [5] to prove their separation between quantum and classical constant-depth circuit complexities. The problem, defined in Section 4, can be informally described as follows: on an n-node ring, the nodes should output one of the possible outcomes that arise when measuring the graph state corresponding to the ring in a basis depending on the input each node receives. We are currently not aware of any applications of Theorem 1 for constructing quantum algorithms for problems of interest to the distributed computing community, but nevertheless consider this result as a valuable proof of concept showing that the quantum LOCAL model can be arbitrarily more powerful than the classical LOCAL model.

The computational problem considered in Theorem 1 is a relation (i.e., for each input there are multiple valid outputs). It is fairly easy to show that for any function (i.e., for each input there is only one valid output at each node) the quantum and classical round complexities are equal in the LOCAL model: we give a proof of this property in Appendix B. We then investigate whether a separation similar to the separation of Theorem 1 can be obtained for a computational problem without input. Such kinds of computational problems (seen as sampling problems or computations of probability distributions) are the main targets of the field of quantum supremacy (see [14] for a recent survey). Indeed, a major open problem left in the work by Bravyi, Gosset and König [5] mentioned above is to prove the superiority of constant-depth quantum circuits for the computation of a probability distribution. We show that in the LOCAL model of distributed computing such a goal can be achieved.

▶ **Theorem 2.** *There exists a sampling problem that can be solved with 2 rounds in the quantum LOCAL model, but requires $\Omega(n)$ rounds in the classical LOCAL model. The classical lower bound holds even for constant-error additive approximation.*

Theorem 2 is proved by considering the same computational problem as used in Theorem 1 but replacing the inputs by random bits. The proof nevertheless requires several adjustments, in particular a careful analysis of the classical randomness shared during the execution of the protocol.

Other relevant works

It is well known that quantum communication can offer significant advantages over classical communication in several settings such as communication complexity or quantum games (see, e.g., [6, 9, 25]). Concerning problems of interest to the distributed computing community,

the main works not already mentioned are quantum algorithms for byzantine agreements [4] and for distributed computing over anonymous networks, and in particular the design of zero-error quantum algorithms for leader election [24] (see also [9]).

2 Preliminaries

2.1 Notations and definitions

Quantum gates

We assume that the reader is familiar with the basis of quantum computation and refer to [21] for a standard reference. We will use the Hadamard gate H and the phase gate S acting on one qubit:

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \mathrm{S} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

where i denotes the imaginary unit of complex numbers. We will also use the CNOT gate acting on two qubits (called the control qubit and the target qubit) that maps the basis state $|a\rangle|b\rangle$, for any two bits $a, b \in \{0, 1\}$, to the state $|a\rangle|a \oplus b\rangle$ where $\oplus$ denotes the exclusive OR. Finally, we will need the following two 2-qubit gates (Controlled-Z and Controlled-S gates):

$$\mathrm{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad \mathrm{CS} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix}.$$

Note that for the gates CZ and CS the order of the qubits the gates act on is unimportant.

Graph-theoretic notation

In this work all the graphs will be undirected and unweighted. For any graph $G = (V, E)$ and any node $u \in V$, we use $N(u)$ to denote the set of neighbors of u.

Graph states

Graph states are a special type of quantum states that are associated with graphs [15]. Let $G = (V, E)$ be any undirected graph. The graph state associated with G is the quantum state on $|V|$ qubits constructed in the following way. Let $\{\mathsf{Q}_u\}_{u \in V}$ denote the $|V|$ registers used to store the qubits of the graph state (each register stores one qubit). First construct the quantum state

$$\bigotimes_{u \in V} |0\rangle_{\mathsf{Q}_u}$$

in these registers. Then apply a Hadamard gate on each register. Finally, for each edge $\{u, v\} \in E$, apply the gate CZ on the pair of registers $(\mathsf{Q}_u, \mathsf{Q}_v)$. The order in which these CZ gates are applied is unimportant, as they all commute.

The total variation distance

Given two probability distributions $p, q \colon X \to [0, 1]$ over a finite set X, the total variation distance (also called statistical distance) between p and q is defined as $\frac{1}{2} \sum_{x \in X} |p(x) - q(x)|$.

2.2 Classical and quantum LOCAL models

In this paper we consider the LOCAL communication model in both the classical and quantum scenarios. The topology of the network is represented by a graph. Executions proceed with round-based synchrony and each node can transfer one message to each adjacent node per round. Initially the nodes of the network share neither any randomness nor, in the quantum scenario, any entanglement.[2] In this paper all the networks are undirected and unweighted. All links and nodes of the network (corresponding to the edges and nodes of the graph, respectively) are reliable and suffer no faults. Each node has a distinct identifier (its size is irrelevant for our purposes). Initially, each node knows nothing about its location in the global topology of the network except the set of edges incident to itself and the number of nodes of the graph.

The processors at each node operate probabilistically in the classical LOCAL model, and they operate quantumly in the quantum LOCAL model. The messages exchanged between them are, respectively, classical and quantum. We do not consider the running time of the processors, as we are only interested in the round complexity. While the classical lower bound of Theorem 2 is proved using a relatively informal definition of the classical LOCAL model, we include its formal definition in Appendix A for completeness.

2.3 The construction from prior works

We now describe the construction introduced in [3], and also used in [5], that shows that non-locality can arise when measuring graph states. For any even integer $d \geq 2$, we define the graph G_d as a ring consisting of $3d$ nodes, and denote the nodes $v_0, v_1, \ldots, v_{3d-1}$ (see Figure 1). It will be convenient to consider this graph as a triangle, with the three nodes v_0, v_d and v_{2d} as corners. We define $V_R = \{v_i \mid i \in \{1, \ldots, d-1\}\}$, $V_B = \{v_i \mid i \in \{d+1, \ldots, 2d-1\}\}$ and $V_L = \{v_i \mid i \in \{2d+1, \ldots, 3d-1\}\}$ as the set of nodes on the right side, bottom side and left side, respectively, of the triangle. We also define V_{even} as the set of all nodes of the graph with even index, and V_{odd} as the set of all nodes with odd index.

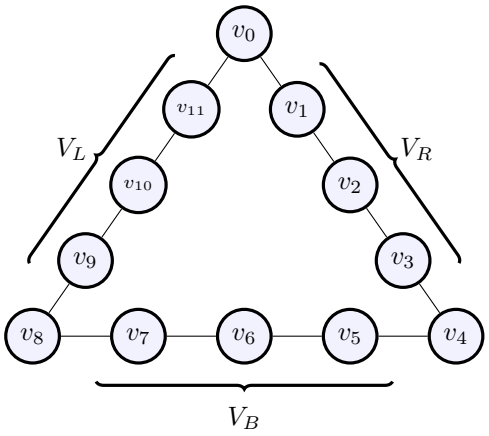


Figure 1 The graph G_d (illustrated for $d = 4$).

Given three bits $b_0, b_1, b_2 \in \{0, 1\}$, consider the process $\mathcal{P}_d(b_0, b_1, b_2)$ described in Figure 2.

[2] The classical lower bound of our first result (Theorem 6) actually holds even if the nodes of the network initially share arbitrary randomness.

1. Create the graph state on the graph G_d.
2. For each $i \in \{0,1,2\}$ apply the quantum gate S^{b_i} to the qubit of node v_{di} (i.e., depending on the value of the three bits b_0, b_1 and b_2, apply either the gate S or the identity gate I on each of three corner nodes v_0, v_d and v_{2d} of the graph).
3. Apply the Hadamard gate H to each qubit of the graph.
4. Measure all qubits in the computational basis. For each $v \in V$, let m_v denote the outcome of the measurement done at node v.

Figure 2 The process $\mathcal{P}_d(b_0, b_1, b_2)$.

From the measurement outcome of the process $\mathcal{P}_d(b_0, b_1, b_2)$, let us define four bits m_E, m_R, m_B and m_L as follows:

$$m_E = \bigoplus_{v \in V_{\text{even}}} m_v, \qquad m_R = \bigoplus_{v \in V_R \cap V_{\text{odd}}} m_v,$$
$$m_B = \bigoplus_{v \in V_B \cap V_{\text{odd}}} m_v, \qquad m_L = \bigoplus_{v \in V_L \cap V_{\text{odd}}} m_v.$$

Refs. [3, 5] characterized which combinations of these four bits can arise as an outcome of the process $\mathcal{P}_d(b_0, b_1, b_2)$:

▶ **Proposition 3.** *([3, 5]) For any bits b_0, b_1, b_2 and any measurement outcome of the process $\mathcal{P}_d(b_0, b_1, b_2)$, the identity $m_R \oplus m_B \oplus m_L = 0$ holds. Additionally, we have:*

$$\begin{cases} m_E = 0 & \text{if } (b_0,b_1,b_2) = (0,0,0), \\ m_E \oplus m_R \oplus m_L = 1 & \text{if } (b_0,b_1,b_2) = (0,1,1), \\ m_E \oplus m_R \oplus m_B = 1 & \text{if } (b_0,b_1,b_2) = (1,0,1), \\ m_E \oplus m_B \oplus m_L = 1 & \text{if } (b_0,b_1,b_2) = (1,1,0). \end{cases}$$

It will be convenient to represent a measurement outcome $\{m_v\}_{v \in V}$ as the binary string $m \in \{0,1\}^{3d}$ where the i-th bit is m_{v_i} for each $i \in \{0, \ldots, 3d-1\}$. We define the *support* of the process $\mathcal{P}_d(b_0, b_1, b_2)$, and denote it $\Lambda_d(b_0, b_1, b_2)$, as the set of all binary strings in $\{0,1\}^{3d}$ corresponding to measurement outcomes arising (with non-zero probability) from the process $\mathcal{P}_d(b_0, b_1, b_2)$.

Finally, our lower bounds will rely on the following lemma, which essentially shows that the quantum correlations from the process $\mathcal{P}_d(b_0, b_1, b_2)$ cannot be simulated classically by local affine functions.

▶ **Lemma 4.** *([3, 5]) Consider any affine function $q_E \colon \{0,1\}^3 \to \{0,1\}$ and any three affine functions $q_R \colon \{0,1\}^2 \to \{0,1\}$, $q_L \colon \{0,1\}^2 \to \{0,1\}$, $q_B \colon \{0,1\}^2 \to \{0,1\}$ such that*

$$q_R(b_0, b_1) \oplus q_B(b_1, b_2) \oplus q_L(b_0, b_2) = 0$$

holds for any $(b_1, b_2, b_3) \in \{0,1\}^3$. Then at least one of the four following equalities does not hold:

$$\begin{aligned} q_E(0,0,0) &= 0, \\ q_E(0,1,1) \oplus q_R(0,1) \oplus q_L(0,1) &= 1, \\ q_E(1,0,1) \oplus q_R(1,0) \oplus q_B(0,1) &= 1, \\ q_E(1,1,0) \oplus q_B(1,0) \oplus q_L(1,0) &= 1. \end{aligned}$$

3 Efficient Construction of Graph States

In this section we consider the construction of graph states in the distributed setting. More precisely, we consider the following problem that we call the *subgraph state construction problem*. The problem is defined on an arbitrary network $G = (V, E)$. Each node $u \in V$ receives a bit $c_u \in \{0, 1\}$ as input. Let $G' = (V', E')$ denote the subgraph of G induced by the node set $V' = \{v \in V \,|\, c_v = 1\}$. The problem asks to create the graph state corresponding to G', shared over the nodes in V': each node $v \in V'$ of the network should own the corresponding 1-qubit register of the graph state (which is the register Q_v in the notations of Section 2.1).

The following theorem shows that this problem can be done efficiently, which is essential for the separation results presented in Sections 4 and 5.

▶ **Theorem 5.** *In the quantum LOCAL model, the subgraph state construction problem can be solved in 2 rounds.*

Proof. The protocol is presented in Figure 3 and illustrated, for a path of two nodes, in Figure 4. This is clearly a 2-round protocol: one round is used at Step 1(c) and one round is used at Step 2(b).

Input: each node $u \in V$ receives a bit c_u

1. Each node $u \in V$ does the following:
   (a) it prepares one 1-qubit register Q_u and, for each neighbor $v \in N(u)$, one 1-qubit register denoted R_u^v (all these registers are initialized to the quantum state $|0\rangle$);
   (b) it applies a Hadamard gate on Q_u, and then a CNOT gate on $(\mathsf{Q}_u, \mathsf{R}_u^v)$ with Q_u as control qubit, for each $v \in N(u)$;
   (c) it sends, for each $v \in N(u)$, the register R_u^v and the bit c_u to node v.
2. Each node $u \in V$ (which now owns the registers Q_u and the registers R_v^u just received) does the following:
   (a) it applies the gate CS to the pair of registers $(\mathsf{Q}_u, \mathsf{R}_v^u)$ for each $v \in N(u)$ such that $c_u \wedge c_v = 1$;
   (b) it sends back the register R_v^u to node v, for each $v \in N(u)$.
3. Each node $u \in V$ (which now owns the registers Q_u and the registers R_u^v) does the following:
   (a) it applies a CNOT gate on $(\mathsf{Q}_u, \mathsf{R}_u^v)$ with Q_u as control qubit, for each $v \in N(u)$;
   (b) it discards the registers R_u^v for all $v \in N(u)$.

Figure 3 The quantum distributed algorithm solving the subgraph state construction problem.

We now prove that the protocol is correct. At the end of Step 1(b), the state of the whole network is:

$$|\varphi\rangle = \bigotimes_{u \in V} \left(\frac{1}{\sqrt{2}} \sum_{j=0}^{1} \left(|j\rangle_{\mathsf{Q}_u} \bigotimes_{v \in N(u)} |j\rangle_{\mathsf{R}_u^v} \right) \right).$$

Let us fix any two nodes u and v such that $\{u, v\} \in E$. The state $|\varphi\rangle$ can be rewritten as

$$\frac{1}{2} \sum_{j=0}^{1} \sum_{k=0}^{1} |j\rangle_{\mathsf{Q}_u} |k\rangle_{\mathsf{Q}_v} |j\rangle_{\mathsf{R}_u^v} |k\rangle_{\mathsf{R}_v^u} |\psi_{j,u,k,v}\rangle |\chi_{u,v}\rangle$$

where $|\chi_{u,v}\rangle$ is a quantum state independent from bits i and j and

$$|\psi_{j,u,k,v}\rangle = \bigotimes_{v' \in N(u)\setminus\{v\}} |j\rangle_{\mathsf{R}_u^{v'}} \bigotimes_{u' \in N(v)\setminus\{u\}} |k\rangle_{\mathsf{R}_v^{u'}}.$$

Note that, for the state $|\varphi\rangle$, applying the gate CS to the pair of registers $(\mathsf{Q}_u, \mathsf{R}_v^u)$ or $(\mathsf{Q}_v, \mathsf{R}_u^v)$ has the same effect as applying it to the pair of registers $(\mathsf{Q}_u, \mathsf{Q}_v)$, yet the former can be done locally. Thus, when node u applies the gate CS to the pair of registers $(\mathsf{Q}_u, \mathsf{R}_v^u)$ and node v applies the gate CS to the pair of registers $(\mathsf{Q}_v, \mathsf{R}_u^v)$, the state $|\varphi\rangle$ is mapped to the quantum state

$$\mathrm{CZ}_{(\mathsf{Q}_u,\mathsf{Q}_v)}|\varphi\rangle,$$

where $\mathrm{CZ}_{(\mathsf{Q}_u,\mathsf{Q}_v)}$ denotes the gate CZ applied to the pair of registers $(\mathsf{Q}_u, \mathsf{Q}_v)$. Since $c_u \wedge c_v = 1$ if and only if $\{u, v\} \in E'$, at the end of Step 2, the whole state of the network is

$$\left(\prod_{\{u,v\}\in E'} \mathrm{CZ}_{(\mathsf{Q}_u,\mathsf{Q}_v)} \right) |\varphi\rangle.$$

Step 3(a) disentangles the registers R_u^v for all $\{u, v\} \in E$, restoring each of them to state $|0\rangle$. Therefore, discarding the registers R_u^v at Step 3(b) does not introduce decoherence in the remaining qubits and, at the end of Step 3, we obtain the desired graph state shared by the nodes in V'. ◀

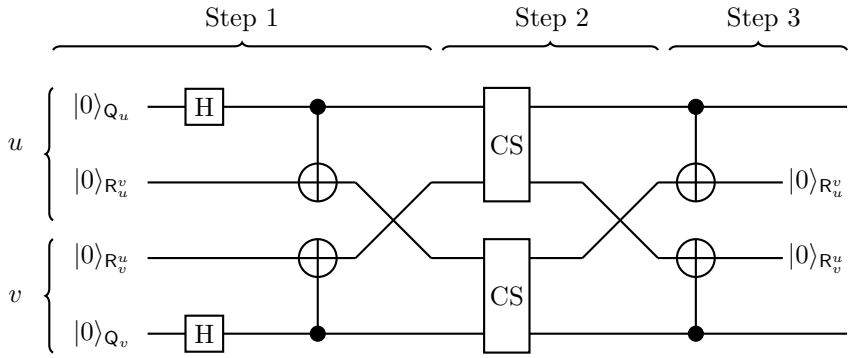


Figure 4 Our protocol illustrated for a 2-path graph $G = (V, E)$ with $V = \{u, v\}$, $E = \{\{u, v\}\}$ and $c_u = c_v = 1$ (the classical messages are omitted from the figure). The global state after Step 1, 2 and 3 is, respectively, $\frac{1}{2}(|0000\rangle + |0101\rangle + |1010\rangle + |1111\rangle)$, $\frac{1}{2}(|0000\rangle + |0011\rangle + |1100\rangle - |1111\rangle)$ and $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$, where the order of qubits is as they appear in the circuit from the top to the bottom.

4 Separation between the Classical and Quantum LOCAL Models

In this section we prove Theorem 1.

For any even integer $d \geq 2$, recall the network $G_d = (V, E)$ defined in Section 2.3, where $V = \{v_0, \ldots, v_{3d-1}\}$. In this section we will consider the network $\mathcal{G}_d$ with node set $V \cup \{w_0, w_1, w_2\}$ and edge set $E \cup \{\{v_0, w_0\}, \{v_d, w_1\}, \{v_{2d}, w_2\}\}$, which is obtained from G_d by adding one node to each corner (see Figure 5).

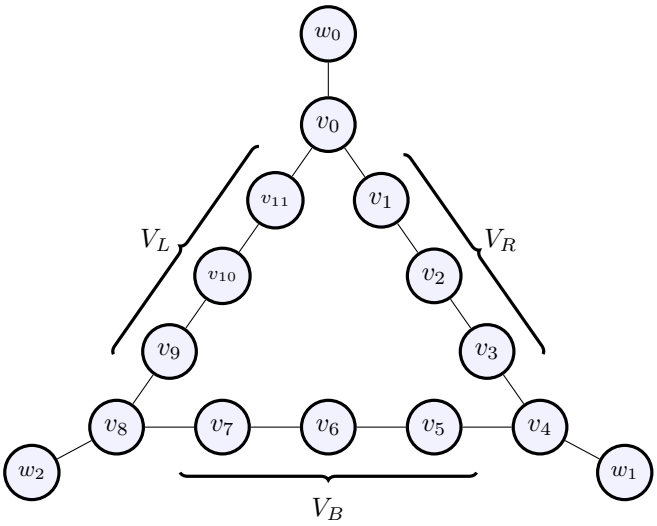


Figure 5 The network $\mathcal{G}_d$ considered to prove the separation (illustrated for $d = 4$).

We now describe the computational problem used to prove our separation. The network considered is $\mathcal{G}_d$, for any even integer $d \geq 2$. The input consists of three bits b_0, b_1 and b_2: node w_0 is given b_0, node w_1 is given b_1, and node w_2 is given b_2 (the other nodes have no input). The output is defined as follows: for each $i \in \{0, 1, \ldots, 3d-1\}$, the node v_i should output one bit x_i. The nodes w_0, w_1 and w_2 do not output anything. The output can thus be seen as a binary string $(x_0, \ldots, x_{3d-1})$ of length $3d$. We say that this string is *valid* if it is in the set $\Lambda_d(b_0, b_1, b_2)$.

The following theorem shows an upper bound on the complexity of this problem in the quantum LOCAL model and a lower bound in the classical LOCAL model.

▶ **Theorem 6.** *There exists a 2-round quantum algorithm that always outputs a valid string. For any integer $T \leq d/2$, no T-round classical algorithm can output a valid string with probability greater than $7/8$ on all inputs $(b_0, b_1, b_2) \in \{0,1\}^3$, even if arbitrary prior randomness is allowed.*

Proof. The considered computational problem can easily be solved in two rounds in the quantum setting by implementing the following process.

▶ **Process 1.** *The nodes of the network first apply the 2-round algorithm of Theorem 5 with input $c_{w_0} = c_{w_1} = c_{w_2} = 0$ and $c_v = 1$ for each $v \in V$. This constructs the graph state over the subgraph G_d of $\mathcal{G}_d$. Moreover, for each $i \in \{0, 1, 2\}$, the node w_i concurrently sends its input b_i to its neighbor v_{di} (the messages can be appended to the messages of the algorithm of Theorem 5). Finally, the nodes of V implement Steps 2–4 of the process $\mathcal{P}_d(b_0, b_1, b_2)$, which can be done without communication, and output their measurement outcomes.*

Note that implementing Process 1 requires each node to know whether it is an input node (w_0, w_1 or w_2), a corner node on the ring (v_0, v_d or v_{2d}) or a non-corner node on the ring (all the other nodes). This is not a problem since each node knows its degree and the type of the nodes depends only on their degrees: the nodes w_0, w_1 and w_2 are the nodes of degree 1, the nodes v_0, v_d and v_{2d} are the nodes of degree 3, and all the other nodes have degree 2.

We now show the classical lower bound, which uses the same argument as in [3] and holds even if the nodes of the network share prior randomness. Consider any classical distributed algorithm $\mathcal{A}$ and fix its randomness r (the string r represents both the shared

prior randomness and the random bits used by the algorithm). This defines a deterministic algorithm that we denote $\mathcal{A}(r)$. Let us write $q_v(b_0, b_1, b_2)$ the bit output at node v by $\mathcal{A}(r)$, for each $v \in V$. Let us define

$$q_E(b_0,b_1,b_2) = \bigoplus_{v\in V_{\text{even}}} q_v(b_0,b_1,b_2), \qquad q_R(b_0,b_1,b_2) = \bigoplus_{v\in V_R\cap V_{\text{odd}}} q_v(b_0,b_1,b_2),$$
$$q_B(b_0,b_1,b_2) = \bigoplus_{v\in V_B\cap V_{\text{odd}}} q_v(b_0,b_1,b_2), \qquad q_L(b_0,b_1,b_2) = \bigoplus_{v\in V_L\cap V_{\text{odd}}} q_v(b_0,b_1,b_2).$$

Assume that the algorithm uses at most $d/2$ rounds. Then, for each $v \in V$, $q_v(b_0, b_1, b_2)$ depends only on one of the bits b_0, b_1, b_2. Since all single-input Boolean functions are affine and so are their sums, q_E, q_R, q_B and q_L are affine functions of b_0, b_1, b_2. Moreover, q_R can only depend on b_0 and b_1, q_B can only depend on b_1 and b_2, and q_L can only depend on b_0 and b_2. From Proposition 3 and Lemma 4 we get that, at least for one choice of $(b_0, b_1, b_2) \in \{0,1\}^3$, the output of $\mathcal{A}(r)$ is not a valid string (i.e., does not correspond to a possible measurement outcome of the process $\mathcal{P}_d(b_0, b_1, b_2)$). A simple counting argument then shows that there exists at least one choice of (b_0, b_1, b_2) for which the original randomized protocol $\mathcal{A}$ fails to output a valid string with probability at least $1/8$. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. Theorem 6 implies that any classical algorithm that outputs a valid string with probability greater than $7/8$ requires a number of rounds linear in the size of the network (since d is a linear function of the size of network $\mathcal{G}_d$).

We now show how to reduce the success probability from $7/8$ to an arbitrary small value: for any constant $\varepsilon > 0$ we construct a new computational problem, which can still be solved in two rounds in the quantum setting, such that any classical algorithm solving this problem with probability at least ε requires a number of rounds linear in the size of the network. Let k be an integer. The problem considered is simply k independent copies of the problem considered so far: the network considered has $3k(d+1)$ nodes and consists of k copies of the network $\mathcal{G}_d$. Each copy receives three bits and outputs a string of $3d$ bits. The output of the whole network is correct if the strings output by each copy are all valid. This problem can obviously be solved using two rounds in the quantum setting by constructing the graph state over the whole network. Theorem 6 implies that for any integer $T \leq d/2$, no T-round classical algorithm can give a correct output with probability greater than $(7/8)^k$ on all inputs, even if arbitrary prior randomness is allowed. Setting $k = \Theta(\log(1/\varepsilon))$ concludes the proof. ◀

5 Separation for a Distribution

In this section we prove Theorem 2. The idea is to convert the relation of the previous section into a distribution by requiring that each input is taken uniformly at random (and requiring that the three nodes with an input output their inputs as well).

Recall Process 1 in the proof of Theorem 6. There, the actions of every node of $\mathcal{G}_d$ depend only on the degree of the node, namely, whether its degree is 1, 2 or 3. The same is true for the 2-round sampling protocol in the quantum LOCAL model described below, which also uses the same network $\mathcal{G}_d$. Therefore, for notational convenience, let us assume that every node knows its global location in $\mathcal{G}_d$.

Consider the probability distribution Γ_d generated by the following 2-round quantum protocol. First, for each $i \in \{0,1,2\}$, the node w_i chooses an unbiased random bit b_i. Then Process 1 is implemented, at the end of which, as specified, nodes $u \in V$ each return one bit. Meanwhile, the nodes w_0, w_1, w_2 output, respectively, b_0, b_1, b_2.

Theorem 2 immediately follows from the following result.

▶ **Theorem 7.** *Every $T \leq d/4$ round algorithm on $\mathcal{G}_d$ in the classical LOCAL model generates a probability distribution that is at least $1/11$ away from Γ_d in the total variation distance.*

Proof. The proof proceeds as follows. Starting from the classical LOCAL model, we present a series of increasingly powerful models on the network $\mathcal{G}_d$. Each model receives no input and returns one bit per node. Then we show that the last, the most powerful among these models cannot generate a probability distribution that has a total variation distance less than $1/11$ to Γ_d.

Consider the classical LOCAL model on the network $\mathcal{G}_d$. We assume that the randomness of each node stems from a finite random bit string that it receives as an input, and all further operations of the node are deterministic (see Appendix A.1 for technical details). We now present a series of steps where each step either strengthens the model or maintains its power while making it easier to analyze.

1. We assume that all the nodes know their location in the global topology.
2. We allow certain nodes to share randomness. In particular, for each $i \in \{0,1,2\}$, let V_i be the set consisting of w_i and all the nodes $u \in V$ at distance at most T away from w_i. And let $V_\perp = V \setminus (V_0 \cup V_1 \cup V_2)$. We assume that, for $i \in \{0,1,2,\perp\}$, all nodes within V_i share randomness, namely, they all start with the same random string Q_i, which we think of as a random variable.

Here it is worth pausing the model-strengthening steps to note that, in a T-round protocol, the bit b_i output by the node w_i depends only on Q_i, thus we may write it as a function $b_i(Q_i)$. Let p_i be the probability that $b_i = 1$. If there exists $i \in \{0,1,2\}$ with $p_i \notin [5/11, 6/11]$, then the marginal distribution over b_i is already at total variation distance greater than $1/11$ away from the corresponding marginal distribution in Γ_d, and the whole distributions (Γ_d and the one generated by the classical protocol) can be only even farther apart. Thus let us assume that $p_i \in [5/11, 6/11]$ for all $i \in \{0,1,2\}$. Since Q_0, Q_1, Q_2 are independent, each $(b_0, b_1, b_2) \in \{0,1\}^3$ is output with probability at least $(5/11)^3 > 1/11$.

3. For $i \in \{0,1,2\}$, let B_i be a random variable that takes value 1 with probability p_i and value 0 with probability $1 - p_i$. For both $\beta \in \{0,1\}$, let Q_i^β be a random variable that equals each value q of Q_i such that $b_i(q) = \beta$ with probability $\Pr[Q_i = q] / \Pr[B_i = \beta]$. We replace the shared randomness Q_i by (Q_i^0, Q_i^1, B_i) – each of the three variables being independent – with an extra requirement that the node w_i always outputs B_i. This is clearly without loss of power, because we can recover Q_i as $Q_i^{B_i}$, for which $b_i(Q_i) = B_i$.
4. We share all the randomness except B_0, B_1, B_2 among all the nodes. More precisely, we assume that all nodes start with the randomness $r = (Q_\perp, Q_0^0, Q_0^1, Q_1^0, Q_1^1, Q_2^0, Q_2^1)$. In addition, for each $i \in \{0,1,2\}$, nodes in V_i start with an additional random bit B_i and we preserve the requirement that w_i must output B_i.

Now we need to show that the final model cannot generate a probability distribution that has a total variation distance at most $1/11$ to Γ_d. Note that, at the beginning of the protocol, the value B_i is only known to nodes at distance at most $T - 1$ away from v_{di}, and, after the protocol, it can be known only to nodes at distance $2T - 1 < d/2$ away from v_{di}. In particular, at the end of the protocol, each node of the network will know no more than one of the values B_0, B_1, B_2. All other communicated information is useless, as, aside from B_0, B_1, B_2, all other randomness is global.

The remainder of the proof is almost equivalent to that of the classical lower bound in Theorem 6, with the sole difference of the counting argument: instead of each choice of (b_0, b_1, b_2) being given with probability exactly $1/8$, now each choice of (b_0, b_1, b_2) is given with probability at least $1/11$. ◀

References

1 Scott Aaronson and Andris Ambainis. Quantum Search of Spatial Regions. *Theory of Computing*, 1(1):47–79, 2005. doi:10.4086/toc.2005.v001a004.

2 Heger Arfaoui and Pierre Fraigniaud. What can be computed without communications? *SIGACT News*, 45(3):82–104, 2014. doi:10.1145/2670418.2670440.

3 Jonathan Barrett, Carlton M. Caves, Bryan Eastin, Matthew B. Elliott, and Stefano Pironio. Modeling Pauli measurements on graph states with nearest-neighbor classical communication. *Physical Review A*, 75:012103, 2007. doi:10.1103/PhysRevA.75.012103.

4 Michael Ben-Or and Avinatan Hassidim. Fast quantum byzantine agreement. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 481–485, 2005. doi:10.1145/1060590.1060662.

5 Sergey Bravyi, David Gosset, and Robert König. Quantum advantage with shallow circuits. *Science*, 362(6412):308–311, 2018. doi:10.1126/science.aar3106.

6 Anne Broadbent and Alain Tapp. Can quantum mechanics help distributed computing? *SIGACT News*, 39(3):67–76, 2008. doi:10.1145/1412700.1412717.

7 Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. Classical Communication and Computation. In *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 63–68, 1998. doi:10.1145/276698.276713.

8 Yi-Jun Chang and Seth Pettie. A Time Hierarchy Theorem for the LOCAL Model. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, pages 156–167, 2017. doi:10.1109/FOCS.2017.23.

9 Vasil S. Denchev and Gopal Pandurangan. Distributed quantum computing: a new frontier in distributed systems or science fiction? *SIGACT News*, 39(3):77–95, 2008. doi:10.1145/1412700.1412718.

10 Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. Can quantum communication speed up distributed computation? In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, pages 166–175, 2014. doi:10.1145/2611462.2611488.

11 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1150–1162, 2012. doi:10.1137/1.9781611973099.91.

12 Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. What Can Be Observed Locally? In *Proceedings of the 23rd International Symposium on Distributed Computing*, pages 243–257, 2009. doi:10.1007/978-3-642-04355-0_26.

13 Daniel M. Greenberger, Michael A. Horne, and Anton Zeilinger. Going beyond Bell's Theorem. In *Bell's Theorem, Quantum Theory, and Conceptions of the Universe*, volume 37 of *Fundamental Theories of Physics*, pages 69–72. Springer, Dordrecht, 1989. doi:10.1007/978-94-017-0849-4_10.

14 Aram W. Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549:203–209, 2017. doi:10.1038/nature23458.

15 Marc Hein, Jens Eisert, and Hans J. Briegel. Multiparty entanglement in graph states. *Physical Review A*, 69:062311, June 2004. doi:10.1103/PhysRevA.69.062311.

16 Stephan Holzer and Roger Wattenhofer. Optimal Distributed All Pairs Shortest Paths and Applications. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, pages 355–364, 2012. doi:10.1145/2332432.2332504.

17 Peter Høyer and Ronald de Wolf. Improved Quantum Communication Complexity Bounds for Disjointness and Equality. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 299–310, 2002. doi:10.1007/3-540-45841-7_24.

18 François Le Gall and Frédéric Magniez. Sublinear-Time Quantum Computation of the Diameter in CONGEST Networks. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2018. doi:10.1145/3212734.3212744.

19 Nathan Linial. Distributive Graph Algorithms-Global Solutions from Local Data. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 331–335, 1987. doi:10.1109/SFCS.1987.20.

20 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

21 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2011. doi:10.1017/CBO9780511976667.

22 David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

23 David Peleg, Liam Roditty, and Elad Tal. Distributed Algorithms for Network Diameter and Girth. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, pages 660–672, 2012. doi:10.1007/978-3-642-31585-5_58.

24 Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Exact Quantum Algorithms for the Leader Election Problem. *ACM Transactions on Computation Theory*, 4(1):1:1–1:24, 2012. doi:10.1145/2141938.2141939.

25 Ronald de Wolf. Quantum communication and complexity. *Theoretical Computer Science*, 287(1):337–353, 2002. doi:10.1016/S0304-3975(02)00377-8.

A Technical Definition of the Classical LOCAL Model

We formalize a T-round classical LOCAL network as follows. We model each node $u \in V$ as a special Turing machine with a work tape, a message tape $M_{u,v}$ for each neighbor $v \in N(u)$, and a read-only random tape. Initially, the work tape contains the input of u (if there is any), the message tapes are blank, and the random tape is initialized to unbiased random bits, independent from one another and from the content of other tapes.

The set of states of each Turing machine is a disjoint union $S_0 \cup \ldots \cup S_T \cup \{q_{\text{fin}}\}$, with one designated "starting" state $q_t \in S_t$ for each $t \in \{0, \ldots, T\}$. The state q_{fin} is the final state, and, for convenience, we define $q_{T+1} = q_{\text{fin}}$. The Turing machine starts in q_0, and, for every $t \in \{0, \ldots, T\}$, we require that a state in S_t can only transition into a state in $S_t \cup \{q_{t+1}\}$. In addition, we require that the transition from S_t to q_{t+1} occurs with probability 1, regardless of the content of the work and the message tapes when the Turing machine first enters q_t.

We formalize the exchange of messages as follows. In round $t \in \{0, \ldots, T\}$, all Turing machines start in their corresponding state q_t and run until they all have reached their corresponding state q_{t+1}. Then, if $t < T$, the configuration of message tapes $M_{u,v}$ and $M_{v,u}$ are swapped for every $\{u, v\} \in E$, and all Turing machines start round $t + 1$. Otherwise, if $t = T$, the work tape of $u \in V$ contains the output of that node.

A.1 Restriction to finite and initial randomness

In the proof of Theorem 2, we are essentially assuming that the random tapes are of finite length. That is without loss of generality because, given any protocol on a finite network and any $\epsilon > 0$, there exists a positive integer L such that, with probability at least $1 - \epsilon$, no Turing machine of the protocol ever visits more than L cells of its random tape. Thus, since ϵ can be chosen arbitrarily small, we can assume all random tapes to be of some finite length L. Via similar reasoning, we can assume that all the randomness is provided at the beginning of the protocol, instead of fresh randomness being provided at each round.

B The Case of Functions

A well-known fact in classical distributed computing is that randomness does not help when computing functions in the LOCAL model. In this appendix we show that this argument extends to the quantum case: we prove that any T-round quantum protocol computing a function can be converted into a T-round classical protocol computing the same function.

Suppose, in the LOCAL model, we have a T-round quantum protocol $\mathcal{P}$ with the network structure given by a graph $G = (V, E)$. And suppose that $\mathcal{P}$ computes some function $f\colon D \to \Sigma^{|V|}$, where Σ is the input-output alphabet and $D \subseteq \Sigma^{|V|}$. More precisely, we assume that, for every input $x \in D$, with probability strictly larger than $1/2$ all nodes $u \in V$ output $f(x)_u$.

For a node $u \in V$ and an integer $i \geq 0$, let the *i-neighborhood* of u, denoted $N_i(u)$, be the set of nodes in V at distance at most i away from $u \in V$. And, for an input $x \in D$, let $x_{u,i}$ denote the restriction of x to $N_i(u)$.

▷ Claim 8. For every $x \in D$ and every $u \in V$, the output of node u is a random variable $O_u(x)$ whose probability distribution depends only on $x_{u,T}$. (This holds true even in a more powerful model where nodes are allowed to share any entanglement prior to receiving the input.)

Since the quantum protocol $\mathcal{P}$ computes f, for every $x \in D$ and every $u \in V$, the random variable $O_u(x)$ takes the value $f(x)_u$ with probability larger than $1/2$. Now consider the following classical T-round *deterministic* protocol: each node $u \in V$ collects the inputs from nodes in its T-neighborhood, which suffices to locally reproduce $O_u(x)$, and then it outputs the most probable value of $O_u(x)$. The correctness of this protocol follows from Claim 8.

Proof of Claim 8. For $t \in \{0, 1, \ldots, T\}$, let ρ_t be the reduced density state of the $(T-t)$-neighborhood of u after t rounds of communication. By induction, we argue that the states $\rho_0, \rho_1, \ldots, \rho_T$ – which we can think of forming the past light cone of ρ_T – all depend only on $x_{u,T}$, and no values of x outside $N_T(u)$. As the base case, it clearly holds for ρ_0 (even in the presence of prior entanglement). For the inductive step, let us assume that, for some $t \geq 0$, ρ_t depends only on $x_{u,T}$. Then the reduced density state of the $(T-t)$-neighborhood of u just before the $(t+1)$-th round of communication depends only on $x_{u,T}$. In that round of communication, nodes in the $(T-t-1)$-neighborhood of u receive messages only from within the $(T-t)$-neighbourhood of u, and thus the state ρ_{t+1} also depends only on $x_{u,T}$. When ρ_T, the final state of the node u, is measured, the probabilities of various outcomes are completely determined by ρ_T. Hence, these probabilities depend only on $x_{u,T}$. ◁

Lifting Theorems for Equality

Bruno Loff
INESC-TEC and University of Porto, Porto, Portugal
bruno.loff@gmail.com

Sagnik Mukhopadhyay
Computer Science Institute of Charles University, Prague, Czech Republic
sagnik@kam.mff.cuni.cz

Abstract

We show a *deterministic simulation (or lifting) theorem* for composed problems $f \circ \mathsf{Eq}_n$ where the inner function (the gadget) is Equality on n bits. When f is a total function on p bits, it is easy to show via a rank argument that the communication complexity of $f \circ \mathsf{Eq}_n$ is $\Omega(\deg(f) \cdot n)$. However, there is a surprising counter-example of a partial function f on p bits, such that any completion f' of f has $\deg(f') = \Omega(p)$, and yet $f \circ \mathsf{Eq}_n$ has communication complexity $O(n)$. Nonetheless, we are able to show that the communication complexity of $f \circ \mathsf{Eq}_n$ is at least $D(f) \cdot n$ for a complexity measure $D(f)$ which is closely related to the AND-query complexity of f and is lower-bounded by the logarithm of the leaf complexity of f. As a corollary, we also obtain lifting theorems for the set-disjointness gadget, and a lifting theorem in the context of parity decision-trees, for the NOR gadget.

As an application, we prove a tight lower-bound for the deterministic communication complexity of the communication problem, where Alice and Bob are each given p-many n-bit strings, with the promise that either all of the strings are distinct, or all-but-one of the strings are distinct, and they wish to know which is the case. We show that the complexity of this problem is $\Theta(p \cdot n)$.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Communication complexity; Theory of computation → Oracles and decision trees

Keywords and phrases Communication complexity, Query complexity, Simulation theorem, Equality function

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.50

Related Version `https://eccc.weizmann.ac.il/report/2018/175/`

Funding *Bruno Loff*: The research leading to these results has received funding from the Foundation for Science and Technology (FCT), Portugal, grant number SFRH/BPD/116010/2016. This work is partially funded by the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT as part of project UID/EEA/50014/2013.
Sagnik Mukhopadhyay: Most of the work is done while the author was a post-doctoral researcher at KTH Royal Institute of Technology, Stockholm. The author is now supported by European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 616787.

Acknowledgements We are thankful to Suhail Sherif, Mark Vinyals, and Susanna de Rezende for many helpful discussions, and Or Meir for pointing out an important bug in an earlier draft of the paper. We also thank the anonymous referees whose insights improved the paper by a substantial amount. We owe an extraordinary debt to Arkadev Chattopadhyay, an outstanding companion of many tea-break conversations on the subject of this paper.

1 Introduction

In the same paper of Karchmer and Wigderson [40], where the notion of formula depth was shown to be equivalent to the communication complexity of their since-homonymous games, was also the first proof separating monotone NC_2 from monotone NC_1. Although not formulated explicitly in this way, their separation of these two circuit classes can be nowadays be presented as a two-part argument: (I) one first shows that the monotone Karchmer–Wigderson game for connectivity on $n^{\Theta(1)}$-node graphs is equivalent to a composition problem in communication complexity, namely $\mathsf{Switch}_n \circ \mathsf{Ind}_n$, the composition of the Switch relation on n bits with the Indexing gadget on $\log n$ bits (given to Alice) and n bits (given to Bob); and (II) one then shows lower-bounds for $\mathsf{Switch}_n \circ \mathsf{Ind}_n$ by lifting an $\Omega(\log n)$ adversarial lower-bound against decision trees trying to solve the Switch_n relation, into an $\Omega((\log n)^2)$ adversarial lower-bound against communication protocols for $\mathsf{Switch}_n \circ \mathsf{Ind}_n$. Formally, a composed function $f \circ g$ consisting of f on p-bits and g on n bits is defined on $p \cdot n$ bit long input $x = \langle x_1, \cdots, x_p \rangle$ (where each x_i is n bit long) as follows: $(f \circ g)(x) = f(g(x_1), \cdots, g(x_p))$.

Their seminal paper led to the following general approach for proving lower-bounds against a given complexity measure. One first (I) finds a composed problem $f \circ g$ whose communication complexity is upper-bounded by the given complexity measure, and (II) one then proves a lower-bound for the communication complexity of $f \circ g$ by arguing that a lower-bound for f in a simple model (such as decision trees) will *lift* to a lower-bound against protocols for $f \circ g$.

Complexity theory has profited greatly from this approach. It appears in the celebrated Raz-McKenzie separation of the monotone NC hierarchy, [57] but also in the best known lower-bounds on monotone formula depth and monotone span programs [59, 54]. Several lower-bounds on the length of proofs in various proof systems were first established using this approach [14, 54, 19], and it is the only known way of proving various separations between complexity classes in communication complexity [27, 26, 25, 24, 23, 68]. It may even be used for proving lower-bounds against data-structure schemes [13], and lower-bounds on the extension complexity of linear programs [42, 46, 22].

Owing partly to this long list of discoveries, and partly to the Karchmer-Raz-Wigderson approach [39] for proving lower-bounds against (non-monotone) NC_1 [30, 17, 21, 15], the lower-bounds community developed a specific interest in understanding the computational complexity of composition, and devoted a large effort to understanding composition problems. Under this heading we should include Sherstov's pattern matrix method [61], and the closely related block-composition method of Shi and Zhu [65], which were developed further in [10, 47, 11, 63, 64, 55], and resulted in many different applications. The problem of understanding the communication complexity of XOR functions [56, 66, 31] is another example of a composition problem, and particularly pertinent to our case since Equality is itself an XOR function, $\mathsf{Eq}_n = \mathsf{NOR}_n \circ \mathsf{XOR}_2$. It is conjectured that the communication complexity of a composition $g \circ \mathsf{XOR}_2$ is approximately equal to the parity decision-tree complexity of g, and in fact this has been shown to hold up to a polynomial if g is a total function [31]. From this conjectured connection, it would follows that the communication complexity of a composition with Equality, $f \circ \mathsf{Eq}_n$, should equal the parity decision-tree complexity of the composition with the NOR function, $f \circ \mathsf{NOR}_n$.

Work on the direct-sum and direct-product problems [35, 2, 29, 33, 16, 53, 34, 36, 1, 5, 6, 7, 4, 41, 32] is also a study of composition, where the outer function f in $f \circ g$ is the hardest possible: the identity function; even this case remains unsolved in various settings.

The complexity of composition is a difficult problem – not just because, generally speaking, lower-bounds are hard to establish, but also because the composition of two hard problems is sometimes not as hard as one may expect: sometimes there is a "collapse" of hardness. A classic example is the case of direct sum in communication complexity: a near-perfect direct sum result holds in the non-deterministic case [49, 38], but fails to hold in the deterministic model [52, 18], and is still an open problem in the randomized model. The following recent example is also of great interest. In the case of deterministic decision-trees, the depth-complexity of $f \circ g$ is the product of the complexities of f and g; this both intuitive and easy to establish, and holds whether f is a total function, a partial function, or relation of any kind. But already if we look into *randomized* decision-trees, Gavinsky *et al.* [20] and Sanyal [60] show that the depth-complexity of the composition $f \circ g$ will be as high as the product of the complexity of f with the *square-root* of the complexity of g; and, surprisingly, [20] exhibit a *relation* f and a function g for which this bound is tight. This "collapse" of hardness when composing relations or partial functions seems to make such problems difficult to understand. As we will see, composition with Equality provides another instance of this phenomenon.

1.1 A tea-break puzzle

Alice and Bob, two renowned complexity theorists, get together during the conference's tea break: *Communication complexity is the most successful area in complexity theory* – Alice says – *at least the natural examples of functions are really well understood.* Bob raises his eyebrows – *do you mean total functions, like Equality, or partial functions, like Gap-Hamming-distance?* – *Both* – replies Alice – *Equality has been well understood since the invention of the field [70], and even Gap-Hamming-Distance is at this point understood for every gap – the constant gap case is a simple result [69], and even* $\frac{1}{\sqrt{n}}$ *fraction gap was eventually understood [9, 67, 62].*

Ok – Bob replied, wryly – *how about the "*$n, (n-1)$*-Equality-Gap"? Suppose you are given* p*-many* n*-bit strings* $x_1, \ldots, x_p$*, and I am given* $y_1, \ldots, y_p$*, and we are promised that either all of the* (x_i, y_i) *pairs are different or exactly one of the* (x_i, y_i) *pairs is equal... show me that we need to communicate* $\Omega(n \cdot p)$ *bits in order to know which is the case...*

Alice thinks for a while – *I know, we can do it via a rank argument. Your "*$n, (n-1)$*-Equality-Gap" function is the composition* $\mathsf{F} \circ \mathsf{Eq}_n$*, where* F *is the partial function which is* 1 *on the all* 0 *string and* 0 *on the strings of Hamming weight* 1*, and* Eq_n *is Equality on* n *bits. The decision tree complexity of* F *is* $\Omega(p)$ *which can be seen by a simple adversarial argument, and by the connection between degree and decision tree complexity [8, 51], we can show that any completion* F' *of* F *has degree* $\Omega(p^{1/3})$*. Also,* Eq_n *has rank* 2^n*, so the rank of the communication matrix of* $\mathsf{F}' \circ \mathsf{Eq}_n$ *is* $2^{\Omega(p^{1/3}n)}$ *(see Lemma 6), and hence the communication complexity is* $\Omega(p^{1/3}n)$*. This is not tight, but it's close to what you want.*

Bob nods – *Your argument holds true, but it only implies that any protocol for* $\mathsf{F}' \circ \mathsf{Eq}_n$ *needs* $\Omega(n \cdot p^{1/3})$ *bits. However, even though a protocol for* $\mathsf{F} \circ \mathsf{Eq}_n$ *does give you a completion of the partial communication matrix for* $\mathsf{F} \circ \mathsf{Eq}_n$*, this completion does not need to be in the composed form* $\mathsf{F}' \circ \mathsf{Eq}_n$ *where* F' *is a completion of* F*. So you did not answer my question, not even if I disregard the polynomial loss...*

At this point Alice does not know what to answer, and rightly so. We will see below an example of a p-bit partial function f, such that any completion of f must have degree $\Omega(p^{1/3})$, and yet the communication complexity of $f \circ \mathsf{Eq}_n$ is $O(n)$, instead of $\Omega(n \cdot p^{1/3})$, which is what one would expect from a rank-degree argument. The protocol that shows this will precisely take advantage of the fact that a completion of $f \circ g$ does not have to be of the form $f' \circ \mathsf{Eq}_n$ for some completion of f' of f. We will also show that such a counter-example does not exist if f is a partial function.

A solution to Bob's tea-break puzzle appears as Corollary 18, in page 10. Using our lifting theorem (Theorem 13, page 9) the desired tight lower-bound of $\Omega(n \cdot p)$ is a 2-line argument.

Interestingly, the counter-example provided below the problem of distinguishing the case when all of the (x_i, y_i) pairs are equal, from the case when all but one of the (x_i, y_i) pairs are equal, so it is strongly related to the example Alice and Bob were discussing above. However, the communication complexity of the example is $\Omega(p \cdot n)$, but the communication complexity of the counter-example is only $O(n)$.

1.2 Composition with Equality

In this work, we answer a question pertaining to the communication complexity of composition of Boolean relations with the Equality gadget. Before stating the question and our main results, we explain the context surrounding this question. We begin with some definitions.

- Define the "Switch" relation: $\mathsf{Switch}_p = \{(z, i) \in \{0,1\}^p \times \{0, 1, \ldots, p\} \mid z_i = 1, z_{i+1} = 0\}$, where we use $z_0 = 1$ and $z_{p+1} = 0$, *i.e.*, we are given p bits and wish to find a "switching point", a position i where a 1-bit flips into a 0-bit. If $z = 0^p$ we must output $i = 0$ and if $z = 1^p$ we must output $i = p$.
- Let $\mathsf{Ind}_n : [n] \times \{0,1\}^n \to \{0,1\}$ denote the two-player Indexing function on n-bits, so that $\mathsf{Ind}_n(x, y) = y_x$.
- Then $\mathsf{Switch}_p \circ \mathsf{Ind}_n$ denotes the composed Boolean relation:

$$\mathsf{Switch}_p \circ \mathsf{Ind}_n = \{(\bar{x}; \bar{y}; i) \in [n]^p \times (\{0,1\}^n)^p \times \{0, 1, \ldots, p\} \mid (y_i)_{x_i} = 1, (y_{i+1})_{x_{i+1}} = 0\}.$$

- Let $\mathsf{Eq}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ denote two-player Equality on n-bits, so that $\mathsf{Eq}_n(x, y) = 1$ iff $x = y$.
- Then $\mathsf{Switch}_p \circ \mathsf{Eq}_n$ denotes the composed Boolean relation:

$$\mathsf{Switch}_p \circ \mathsf{Eq}_n = \{(\bar{x}; \bar{y}; i) \in (\{0,1\}^n)^p \times (\{0,1\}^n)^p \times \{0, 1, \ldots, p\} \mid x_i = y_i, x_{i+1} \neq y_{i+1}\}.$$

- Let $F \subseteq \mathcal{A} \times \mathcal{B} \times \mathcal{C}$ be a relation. The *deterministic communication complexity of* F, $\mathsf{D}^{cc}(F)$, is the minimum communication cost of a protocol for solving the communication problem where Alice is given $a \in \mathcal{A}$, Bob is given $b \in \mathcal{B}$, and they wish to find c such that $(a, b, c) \in F$, whenever one such c exists (see [45], Chapter 5).
- Let $f \subseteq \{0,1\}^p \times \mathcal{C}$ be a relation. The *deterministic query complexity of* f, $\mathsf{D}^{dt}(f)$, is the minimum number of queries made by a deterministic decision-tree which, given query access to $z \in \{0,1\}^p$, finds a $c \in \mathcal{C}$ such that $(z, c) \in F$, whenever one such c exists.

In STOC'88, Karchmer and Wigderson [40] presented a proof that connectivity is not in monotone NC_1. At the heart of their result was an argument which may be reinterpreted as a proof of the following theorem:

▶ **Theorem 1** (Karchmer and Wigderson, [40]). $\mathsf{D}^{cc}(\mathsf{Switch}_p \circ \mathsf{Ind}_n) = \Omega((\log n) \cdot \log p)$.

In Structures'91, the conference now known as CCC, Grigni and Sipser [28] provided an alternative proof that connectivity is not in monotone NC_1. Their proof uses Eq in place of Ind, and this allows for a simpler argument:

▶ **Theorem 2** (Grigni and Sipser, [28]). $\mathsf{D}^{cc}(\mathsf{Switch}_p \circ \mathsf{Eq}_n) = \Omega(n \cdot \log p)$.

It is not hard to see that Theorem 2 implies Theorem 1, by reducing $\mathsf{Eq}_{\log n}$ to Ind_n. Later, in FOCS'97, Raz and McKenzie [57] separated the entire monotone NC hierarchy. At the heart of their proof was an argument for a vast generalization of Theorem 1:

▶ **Theorem 3** (Raz and McKenzie, [57]). *For any Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$, whenever $n \geq p^{20}$, $\mathsf{D}^{cc}(f \circ \mathsf{Ind}_n) = \Omega\left((\log n) \cdot \mathsf{D}^{dt}(f)\right)$.*

Theorem 3 was not stated with such generality in [57], but appeared in this form in a recent work of Göös, Pitasi and Watson [26]. Theorem 3 has been the basis of several papers [26, 12, 44].

Knowing the above history, one naturally comes to the question of whether one can prove a similar generalization for Grigni and Sipser's Theorem 2, i.e., whether we can prove the conjecture:

▶ **Conjecture 4.** *For any Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$, $\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) = \Omega\left(n \cdot \mathsf{D}^{dt}(f)\right)$.*

Very general lifting theorems may be proven using rank arguments, and the current state of the art [59, 54] is a lifting of the *Nullstellensatz degree* of any CNF-relation[1] f to the rank of $f \circ g$, which works for a large class of gadgets g having a certain algebraic property[2]. The equality gadget does possess the required property, however our lower-bound technique will work for any relation, and not just CNF-relations.

In the case when f is a total function, however, there is an *ad-hoc* degree-to-rank lifting theorem which works for the equality gadget, and which is in the same spirit as [59, 54]. It uses the following characterization:

▶ **Proposition 5** ([3]). *If h is a Boolean function and F is the communication matrix of $h \circ \mathsf{XOR}_2$, then $\mathsf{rank}(F) = \|h\|_0$.*

Above, $\mathsf{rank}(F)$ is the real rank of the communication matrix of F, and $\|h\|_0$ is the Fourier sparsity (the number of non-zero Fourier coefficients) of h. We can view $f \circ \mathsf{Eq}_n$ as an XOR_2 function, $f \circ \mathsf{NOR}_n \circ \mathsf{XOR}_2$. The following observation is easy to prove, but the proof is omitted due to space constraints (see the ECCC version [48] for the proof).

▶ **Lemma 6.** *For every $f : \{+1,-1\}^p \to \{+1,-1\}$ with $\deg(f) \geq 1$, and every $g : \{+1,-1\}^n \to \{+1,-1\}$, we have $\|f \circ g\|_0 \geq (\|g\|_0 - 1)^{\deg(f)}$.*

Lemma 6 implies that $\|f \circ \mathsf{NOR}_n\|_0 = \Omega(2^{\deg(f)\cdot n})$, since $\|\mathsf{NOR}_n\|_0 = 2^n$. By the rank-lower bound for communication complexity, we thus have $\mathsf{D}^{cc}(f \circ \mathsf{Eq}) \geq \Omega(\deg(f) \cdot n)$. Now we can use the following connection between $\deg(f)$ and $\mathsf{D}^{dt}(f)$, which improves upon a theorem of Nisan and Smolensky theorem [8].

▶ **Proposition 7** ([51]). *, $\deg(f) = \Omega(\mathsf{D}^{dt}(f)^{1/3})$.*

Combining the three above facts, we get that when f is a total Boolean function, then $\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) = \Omega(\mathsf{D}^{dt}(f)^{1/3} \cdot n)$. This easy-to-prove result is similar to Conjecture 4, except for the $1/3$ loss in the exponent, and works for all total functions. But surprisingly, when allow f to be a partial function, Conjecture 4 is false! The following counter-example was given to us by Arkadev Chattopadhyay, Suhail Sherif, and Mark Vinyals. Let $f \subseteq \{0,1\}^p \times \{0,1\}$ be the relation

$$f = \{(z,1) \mid |z| = p \text{ or } |z| < p-1\} \cup \{(z,0) \mid |z| = p-1 \text{ or } |z| < p-1\},$$

[1] A CNF-relation $f_\phi \subseteq \{0,1\}^n \times [m]$ is defined for a given unsatisfiable CNF ϕ on n variables and m clauses, by $(x,i) \in f_\phi$ if x falsifies the i-th clause. Such relations appear prominently in the study of monotone Karchmer–Wigderson games.

[2] These results are explained in Robert Robere's excellent PhD thesis [58]. The mentioned algebraic property appears in Section 5.1.

i.e., we are given a Boolean string $z \in \{0,1\}^p$, and wish to distinguish the case when z has Hamming weight p from the case when z has Hamming weight $p-1$. It is easy to show that $\mathsf{D}^{dt}(f) \geq p$: an adversary keeps answering 1 to all queries, and $f(z)$ will remain unknown until the very last query. This adversary also shows that $\mathsf{D}^{dt}(f') \geq p$ for any "completion" of f, i.e. any total function $f' : \{0,1\}^p \to \{0,1\}$ which agrees with f on the inputs with Hamming weight p or $p-1$; and hence $\deg(f') = \Omega(p^{1/4})$ for any such f', by Proposition 7. So one might mistakenly hope, like Alice did in Section 1.1, that a rank/degree argument would serve to prove a lower-bound of $\Omega(p^{1/4} \cdot n)$ for $f \circ \mathsf{Eq}_n$.

However, a protocol for $f \circ \mathsf{Eq}_n(x_1, \ldots, x_p; y_1, \ldots, y_p)$ may proceed as follows. Think of each of Alice and Bob's inputs for $f \circ \mathsf{Eq}_n$ as a matrix with p rows and n columns. Then let $a \in \{0,1\}^n$ be the XOR of each column of Alice's input, and $b \in \{0,1\}^n$ be the XOR of each column of Bob's input. Then Alice sends a to Bob, and Bob replies whether $a = b$. Now, if every x_i equals the corresponding y_i, then clearly $a = b$; and if every x_i equals y_i, except for a single value of $i \in [p]$, then there must exist a $j \in [n]$ such that $a_j \neq b_j$. It then holds that $\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) \leq n + 1$, and so Conjecture 4 is false. Remarkably, this seems to suggest that rank/degree arguments will fail to hold.

This counter-example also shows that $\mathsf{Eq}_{\log n}$ behaves differently from Ind_n, when used as the inner function in a composition – indeed Theorem 3 implies that $\mathsf{D}^{cc}(f \circ \mathsf{Ind}_n) \geq p \log n$, which is strictly higher when $p = \omega(1)$. The difference between Equality and Indexing may be further explained with the help of a recent paper of Chattopadhyay, Koucký, and the authors [12]. There it is shown that a theorem like Conjecture 4 will hold for any inner function g, in place of Eq_n, which admits certain *hitting distributions*[3]. As it turns out, all gadgets for which we could prove a deterministic simulation theorem, namely, Indexing [57], Inner-product and gap-Hamming [12], and several others [44], all admit such hitting distributions. But it may be seen that although Equality has a 0-hitting distribution, it fails to have any 1-hitting distribution.

The existence of such a counter-example was surprising to us, because in the case of the Switch relation, the Karchmer–Wigderson theorem and Grigni–Sipser theorem behave the same way (by lifting a decision-tree adversary for the Switch relation). The main purpose of this work was to understand what is happening.

1.3 Almost Conjecture 4

We will be able to prove a simulation theorem for composition with Equality, but for a notion different than decision-tree depth. In order to avoid long preliminaries for now, we postpone the full list of our results until the end of Section 2. However, one of our results is sufficiently close to what was already discussed, that it may be easily stated in the present section, and may thus serve as motivation for the remainder.

For a given relation $f \subseteq \{0,1\}^p \times \mathcal{C}$, let $\mathsf{L}^{dt}(f)$ denote the smallest number of leaves of any deterministic decision-tree which, given query access to $z \in \{0,1\}^p$, finds a $c \in \mathcal{C}$ such that $(z,c) \in F$, whenever such a c exists. Notice that $\mathsf{D}^{dt}(f) \geq \log \mathsf{L}^{dt}(f)$, and so if Conjecture 4 were true, a consequence would be that $\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) = \Omega\left(n \cdot \log \mathsf{L}^{dt}(f)\right)$. The following theorem, thus, may be considered a weak variant of Conjecture 4:

[3] A (δ, h)-hitting rectangle-distribution (for $\delta \in (0,1)$ and $h \in \mathbb{N}$) is a distribution over rectangles such that a random rectangle from this distribution will intersect any 2^{-h}-large rectangle with probability $\geq 1 - \delta$. By a Boolean function g having (δ, h)-hitting monochromatic rectangle-distributions, we mean that there are two (δ, h)-hitting rectangle-distributions σ_0 and σ_1, such that σ_c only samples rectangles which are c-monochromatic with respect to g.

▶ **Theorem 8** (Lifting for $\log \mathsf{L}^{dt}$). *For any Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$, whenever $n \geq 100 \cdot \log p$,*

$$\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) = \Omega\left(n \cdot \frac{\log \mathsf{L}^{dt}(f)}{\log p}\right).$$

1.4 Organization

In Section 2 we state the definitions required to understand the statements of our results, and then state all our results in full; in this section we give the first new concept required by our results, namely the notion of 0-query complexity. In Section 3, we introduce the combinatorial invariants required to prove our main result, including the notion of *thickness*, which comes from Raz and McKenzie [57, 26, 12], but also the notion of *square*, which is the second new concept required by our proofs. In Section 4 we prove a *projection lemma* – the crucial lemma required to prove the simulation theorem – which is then proven in Section 5.

2 Preliminaries, and precise statements of our results

In this section we provide basic notations and precise statements of all our results.

We will assume the reader is familiar with various basic concepts pertaining to complexity of Boolean functions, namely: decision trees, query complexity, leaf complexity, protocol trees, communication complexity, combinatorial rectangles, and Fourier analysis of Boolean functions. See [45, 37] for reference.

We will be studying the decision-tree complexity of relations. A *Boolean relation* f is a subset of $\{0,1\}^p \times \mathcal{C}$ where $\mathcal{C}$ is a finite set; associated with f is the search problem where we are given a string $z \in \{0,1\}^p$, and wish to find an element $c \in \mathcal{C}$ such that $(z,c) \in \mathcal{C}$, if such an element exists.[4] If to each z corresponds exactly one c, we call f a *total Boolean function*.

For a given Boolean relation, we let $\mathsf{D}^{dt}(f)$, called the *query complexity of* f, be the minimum height of T, taken over deterministic decision-trees T which solve the search problem associated with f. We let $\mathsf{L}^{dt}(f)$, called the *leaf complexity of* f, be the minimum number of leaves of T, again taken over deterministic decision-trees T which solve the search problem associated with f.

We will also be interested in the communication complexity of relations. A *two-player relation* F is a subset $F \subseteq \mathcal{A} \times \mathcal{B} \times \mathcal{C}$ where $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are finite sets; associated with F is the communication problem where Alice is given $a \in \mathcal{A}$, Bob is given $b \in \mathcal{B}$, and they wish to find $c \in \mathcal{C}$ such that $(a,b,c) \in F$, if one such c exists. If $g \subseteq \mathcal{A} \times \mathcal{B} \times \{0,1\}$ is a two-player relation such that to each pair $(a,b) \in \mathcal{A} \times \mathcal{B}$ corresponds exactly one $c \in \{0,1\}$ with $(a,b,c) \in g$, we call g a *gadget*. The Equality and Indexing function defined in page 4 are examples of gadgets. A third example is the *Set-disjointness function* $\mathsf{Disj}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, where $\mathsf{Disj}_n(x,y) = 0$ iff $x_i = y_i = 1$ for some $i \in [n]$.

For a given two-player relation $F \subseteq \mathcal{A} \times \mathcal{B} \times \mathcal{C}$, we let $\mathsf{D}^{cc}(F)$, called the *communication complexity of* F, be the height of the shortest deterministic protocol-tree for solving the communication problem associated with F.

[4] Although when considering functions the difference between a total function and a partial function (a promise problem) is very important, this distinction is irrelevant when thinking more generally about relations, at least in computational models which are guaranteed to produce an output. Indeed, a partial Boolean relation $f \subseteq \{0,1\}^n \times \mathcal{C}$ may be replaced by the total Boolean relation $f' = f \cup \{(x,c) \in \{0,1\}^n \times \mathcal{C} \mid (x,c') \notin f \text{ for any } c' \in \mathcal{C}\}$, meaning if the input is outside the promise we allow the algorithm to output anything.

The *composition* of a Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$ with a gadget $g : \mathcal{A} \times \mathcal{B} \to \{0,1\}$ is the two-player relation $f \circ g \subseteq \mathcal{A}^p \times \mathcal{B}^p \times \mathcal{C}$, given by

$$f \circ g = \{(a_1, \ldots, a_p; b_1, \ldots, b_p; c) \mid (g(a_1, b_1) \ldots g(a_p, b_p), c) \in f\}.$$

The following definition is crucial to our result and, to our knowledge, has not been used prior to this work:

▶ **Definition 9.** *Given a deterministic decision-tree T over $\{0,1\}^p$, the* 0-depth of T *is the maximum number of queries which are answered* 0*, in any root-to-leaf path of T. The* 0-query complexity *of f, denoted $\mathsf{D}_0^{dt}(f)$, to be the smallest* 0*-depth of T, taken over deterministic decision-trees T which solve the search problem associated with f.*

It is unusual to make a query complexity notion depend on the specific outcome of the queries, instead of just the number of queries. However, the above notion is closely related to a notion analogous to parity decision-trees. Indeed, we may define AND decision-trees to be like parity decision-trees, but where the algorithm is allowed the query an AND of the input bits, instead of a parity of the input bits:

▶ **Definition 10.** *An* AND *decision-tree over $\{0,1\}^p$ is a rooted tree where each internal node v is labeled by a set of variables $Q_v \subseteq [p]$ and each edge is labeled* 0 *or* 1*. As in the case of deterministic decision-tree, the execution of T on an input $z \in \{0,1\}^p$ traces a path in this tree: at each internal node v the execution is given the value of the conjunction $q = \bigwedge_{i \in Q_v} z_i$, and follows the edge labeled q into one of v's children. With each node v of the tree we may associate the set $S_v \subseteq \{0,1\}^p$ of those inputs whose execution follows the path down to the node v; the set S_v is given by a system of conjunctive equations.*

An AND *decision-tree over $\{0,1\}^p$ is said to solve the search problem associated with a Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$ if, for every leaf v, there exists a choice of $c \in \mathcal{C}$ such that $(z, c) \in f$ for every $z \in S_v$.*

Then, the AND-query complexity *of f, denoted $\mathsf{D}_{\mathsf{AND}}^{dt}(f)$, is defined as the minimum depth of T, taken over* AND *decision-trees T which solve the search problem associated with f.*

We are then able to establish the following relationship:

▶ **Lemma 11.** *Let $f \subseteq \{0,1\}^p \times \mathcal{C}$ be any Boolean relation. Then*

$$\mathsf{D}_{\mathsf{AND}}^{dt}(f) \geq \mathsf{D}_0^{dt}(f) \geq \frac{\mathsf{D}_{\mathsf{AND}}^{dt}(f)}{\lceil \log(p+1) \rceil}$$

Since these measures are within a $\log p$ factor of each other, it is possible to think of the more natural $\mathsf{D}_{\mathsf{AND}}^{dt}(f)$ as a proxy for $\mathsf{D}_0^{dt}(f)$. The proof is simple, but is omitted due to space constraints (it appears in the full version of the paper [48]).

There is also a simple relation between 0-query complexity and leaf complexity. If a decision-tree over p bits never makes more than d zero-queries, each root-to-leaf path may be specified by the positions of the 0-answers along that path, so there are fewer than $\binom{p}{\leq d} \leq 2^{(d+1)\log p}$ leaves. Hence it follows:

▶ **Lemma 12.** *Let $f \subseteq \{0,1\}^p \times \mathcal{C}$ be any Boolean relation. Then*

$$\mathsf{D}_0^{dt}(f) \geq \frac{\log \mathsf{L}^{dt}(f)}{\log p} - 1.$$

If $\log \mathsf{L}^{dt}(f) = \Omega(p)$, we have $\binom{p}{\leq d} \geq 2^{\Omega(H_2(d/p) \cdot p)}$, and so $\mathsf{D}_0^{dt}(f) = \Omega(p)$ also.

Lifting theorems for Equality. Our main result is a simulation theorem which lifts 0-query complexity of a Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$ to the communication complexity $f \circ \mathsf{Eq}_n$:

▶ **Theorem 13** (Lifting for D_0^{dt})**.** *Let $f \subseteq \{0,1\}^p \times \mathcal{C}$ be any Boolean relation. Then, whenever $n \geq 100 \cdot \log p$,*

$$\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) = \Omega\left(n \cdot \mathsf{D}_0^{dt}(f)\right).$$

The proof of Theorem 13 uses the notion of thickness from Raz-McKenzie [57], and a new invariant, called a *square*, which is inspired by Grigni-Sipser [28]. These notions are presented in Section 3.

Our proof is similar in flavor to Or Meir's lower-bound for the direct-sum of the universal relation [50], although for that problem a rank argument will work [43].[5]

▶ Remark 14. It is not hard to verify that $\mathsf{D}_0^{dt}(f) = 1$ when f is the counter-example to Conjecture 4, which we described in Section 1.2: a decision tree for f queries coordinates one at a time until it finds the first 0. Then it follows from Theorem 13 that the protocol for $f \circ \mathsf{Eq}_n$ appearing in page 5 is optimal, up to constant factors.

Theorem 8 follows from Theorem 13 and Lemma 12. Theorem 13 and Lemma 11 give us the following:

▶ **Corollary 15** (Lifting for $\mathsf{D}_{\mathsf{AND}}^{dt}$)**.** *Let $f \subseteq \{0,1\}^p \times \mathcal{C}$ be any Boolean relation. Then, whenever $n \geq 100 \cdot \log p$, $\mathsf{D}^{cc}(f \circ \mathsf{Eq}_n) = \Omega\left(n \cdot \frac{\mathsf{D}_{\mathsf{AND}}^{dt}(f)}{\log p}\right)$.*

Lifting theorems for Set-disjointness. By a simple reduction, we are also able to show the first lifting theorem known for set-disjointness. Indeed, we may reduce an instance of Eq_n to an instance of Disj_{2n}. Alice maps each of her bits x_i into the pair of bits $a_i = (1 - x_i)x_i$, and Bob maps each of his bits y_i into $b_i = y_i(1 - y_i)$; it now holds that $x_i = y_i$ iff a_i and b_i are disjoint, and hence $\mathsf{Eq}_n(x, y) = \mathsf{Disj}_{2n}(a, b)$. As a corollary, we find:

▶ **Corollary 16** (Lifting for disjointness)**.** *Let $f \subseteq \{0,1\}^p \times \mathcal{C}$ be a Boolean relation and $n \geq 100 \cdot \log p$. Then $\mathsf{D}^{cc}(f \circ \mathsf{Disj}_n) = \Omega\left(n \cdot \mathsf{D}_0^{dt}(f)\right)$.*

Naturally, Theorem 8 and Corollary 15 will hold for Set-disjointness.

Lifting theorems for parity decision-trees. A composition with Equality, $f \circ \mathsf{Eq}_n$, is a XOR function $f \circ \mathsf{NOR}_n \circ \mathsf{XOR}_2$. It is well known and easy to see that $\mathsf{D}^{cc}(F \circ \mathsf{XOR}_2) \leq \mathsf{D}_{\oplus}^{dt}(F)$ [31], where $\mathsf{D}_{\oplus}^{dt}(F)$ is the parity-query complexity of F. Hence a consequence of our lifting theorem for Equality in communication complexity is also a lifting theorem for the NOR function, with respect to parity decision-trees:

▶ **Corollary 17.** *For any Boolean relation $f \subseteq \{0,1\}^p \times \mathcal{C}$, whenever $n \geq 100 \cdot \log p$, $\mathsf{D}_{\oplus}^{dt}(f \circ \mathsf{NOR}_n) = \Omega\left(n \cdot \mathsf{D}_0^{dt}(f)\right)$.*

It may be seen that $\mathsf{D}_0^{dt}(f)$ cannot be replaced by $\mathsf{D}^{dt}(f)$, by the same counter-example f of page 5.

[5] Or Meir's proof is similar to what one would obtain if one were to carry out our proof when f is the identity function, so our technique can be seen as a generalization of Meir's. Of course in our case composition with identity would be just a larger equality, so the lower-bound follows trivially, whereas in the case of the universal relation the result is not trivial.

A solution to the tea-break puzzle. A lifting theorem such as Theorem 13 is a powerful tool for proving lower-bounds in communication complexity. The theorem is very general and many such results may be proven, but let us here give an example of lower-bound for a concrete problem in communication complexity.

Consider the Bob's example $\mathsf{F} \circ \mathsf{Eq}_n$ from the tea-break puzzle where Alice and Bob are each given p-many n-bit strings, with the promise that either all strings are different, or exactly one pair of strings is equal, and they wish to know which is the case.

We have $\mathsf{F}(z) = 1$ when its input, z, has Hamming weight 0, and $\mathsf{F}(z) = 0$ when z has Hamming weight 1. This is a partial function, so we may not use Lemma 6 to prove a lower-bound on it. However (this is the two-line proof): an adversary may answer 0 $p-1$ times before fixing $\mathsf{F}(z)$; hence $\mathsf{D}_0^{dt}(\mathsf{F}) \geq p-1$, and it follows immediately from Theorem 13:

▶ **Corollary 18.** *Whenever* $n \geq 100 \cdot \log p$, $\mathsf{D}^{cc}(\mathsf{F} \circ \mathsf{Eq}_n) = \Omega(n \cdot p)$.

To the best of our knowledge, there is currently no other way to establish this lower-bound.

3 Thickness and squares

Notation . If p is a natural number, we write $[p]$ for the set $\{1, \ldots, p\}$. For sets A and B, we use $A \to B$ to denote the set of total functions from A to B. We write $f : A \to B$ to mean $f \in (A \to B)$. We also use B^A to denote the set of total functions from A to B, but in this case we think of them as A-indexed sequences of elements from B, and if we first write $f \in B^A$, instead of $f : A \to B$, we will later write f_a instead of $f(a)$. If $f : A \to B$ (or $f \in B^A$) and $A' \subseteq A$, then $f\big|_{A'}$ is the restriction of f to A'. A disjoint union is denoted by $\uplus$, i.e. $A \uplus B$ denotes the union of two disjoint sets A and B.

We will look at sets $A \subseteq (\{0,1\}^n)^{[p]}$, and we will often want to think of some set of coordinates $I \subseteq [p]$ as being *alive*, and the corresponding complement $D = [p] \setminus I$ will be the set of *dead* coordinates. We will be working with partial assignments of elements from $(\{0,1\}^n)^{[p]}$, which can be encoded as total functions from I to $\{0,1\}^n$. Hence the following two definitions will be helpful.

▶ **Definition 19** (Join). *Let* $n \geq 1$ *and* $p \geq 2$ *be integers,* $\varnothing \neq I \subsetneq [p]$ *and* $D = [p] \setminus I$.
If $s' \in (\{0,1\}^n)^I$ *and* $s'' : (\{0,1\}^n)^D$, *then their* join $s' \times s'' \in (\{0,1\}^n)^{[p]}$ *is given by:*

$$(s' \times s'')_i = \begin{cases} s'_i & \text{if } i \in I \\ s''_i & \text{if } i \in D. \end{cases}$$

This notation is extended to subsets of $(\{0,1\}^n)^I$ *and* $(\{0,1\}^n)^D$ *in the natural way.*
If $i \in I \subseteq [p]$, $s' \in \{0,1\}^n$ *and* $s'' \in (\{0,1\}^n)^{I \setminus \{i\}}$, *then their* join at i *is the sequence* $s' \times_i s'' \in (\{0,1\}^n)^I$ *with* $(s' \times_i s'')_i = s'$, *and* $\forall j \in I \setminus \{i\}$ $(s' \times_i s'')_j = s''_j$.

▶ **Definition 20.** *Let* $n \geq 1$ *and* $p \geq 2$ *be integers,* $I \subseteq [p]$, $i \in I$ *and* $S \subseteq (\{0,1\}^n)^I$. *We define the* projections*:* $S_i = \{s_i \mid s \in S\} \subseteq \{0,1\}^n$ *and* $S_{\neq i} = \{s\big|_{I \setminus \{i\}} \mid s \in S\} \subseteq (\{0,1\}^n)^{I \setminus \{i\}}$.

Likewise if $\varnothing \neq E \subset I$, *we define* $S_E = \{s\big|_E \mid s \in S\} \subseteq (\{0,1\}^n)^E$ *and, for each* $s'' \in (\{0,1\}^n)^{I \setminus E}$, *the extensions of* s'' *in* S *is the set* $\mathsf{Ext}_S(s'') = \{s' \in (\{0,1\}^n)^E \mid s' \times s'' \in S\}$.

For a subset $U \subseteq \{0,1\}^n$, *the* restriction of S to U at coordinate i *is the set* $S^{i,U} = \{s \in S \mid s(i) \in U\}$. *We will also write* $S^{i,U}_{\neq i}$ *for the set* $(S^{i,U})_{\neq i}$ *(i.e. we first restrict the* i*-th coordinate then project onto the remaining coordinates in* I*):* $S^{i,U}_{\neq i} = \{s\big|_{I \setminus \{i\}} \mid s \in S, s_i \in U\}$.

3.1 Thickness and its properties

The notion of *thickness* was first used by Raz and McKenzie in [57], and is by now a well-known notion. But whereas previously the notion of thickness was only looked at with respect to all coordinates simultaneously, we will be interested in the notion of thickness with respect to a subset of coordinates. This difference is non-essential, and all the relevant properties are proven mutatis mutandis. Due to space constraints, the proofs are omitted (but appear in the full version of the paper [48]).

▶ **Definition 21** (Aux graph, average and min-degrees). *Let $n \geq 1, p \geq 2$ be integers, $I \subseteq [p]$, and $S \subseteq (\{0,1\}^n)^I$. For each $i \in I$, the aux graph $G(S,i)$ is the bipartite graph with left-side vertices S_i, right-side vertices $S_{\neq i}$ and edges corresponding to the set S, i.e., (s', s'') is an edge iff $s' \times_i s'' \in S$.*

We define the average degree of $G(S,i)$ to be the average right-degree: $d_{\text{avg}}(S,i) = \frac{|S|}{|S_{\neq i}|}$, and the min-degree of $G(S,i)$, to be the minimum right-degree: $d_{\min}(S,i) = \min_{s'' \in S_{\neq i}} |\mathsf{Ext}_S(s'')|$.

▶ **Definition 22** (Thickness and average thickness). *Let $n \geq 1, p \geq 2$ be integers, $\varnothing \neq F \subseteq I \subseteq [p]$, and $S \subseteq (\{0,1\}^n)^I$. Then S is called τ-*thick on F *if $d_{\min}(S,i) \geq \tau \cdot 2^n$ for all $i \in F$. (By convention an empty set S is τ-thick.) Similarly, S is called φ-*average-thick on F *if $d_{\text{avg}}(S,i) \geq \varphi \cdot 2^n$ for all $i \in F$. For $p = 1$, set S is τ-thick if $|S| \geq \tau \cdot 2^n$.*

We will need the following two lemmas. The proofs are similar to the analogous lemmas in [26].

▶ **Lemma 23** (Average thickness implies thickness). *Let $n \geq 1, p \geq 2$ be integers, $\varnothing \neq F \subseteq I \subseteq [p]$, and $S \subseteq (\{0,1\}^n)^I$. If S is φ-average-thick on F, then for every $\delta \in (0,1)$ there is a subset $S' \subseteq S$ which is $\frac{\delta}{p}\varphi$-thick on F and has $|S'| \geq (1-\delta) \cdot |S|$.*

A recent example by Kozachinskiy [44] shows that the $\frac{1}{p}$ loss in Lemma 23 is needed. This loss is the core reason why we need the gadget to have size $n = \Omega(\log p)$ in Theorem 13.

▶ **Lemma 24.** *Let $n \geq 1, p \geq 2$ be integers, $i \in F \subseteq I \subseteq [p]$, and $S \subseteq (\{0,1\}^n)^I$ be τ-thick on F. Then for any set $U \subseteq \{0,1\}^n$, $S^{i,U}_{\neq i}$ will also be τ-thick on $F \setminus \{i\}$, and $S^{i,U}_{\neq i}$ will be empty iff $U \cap S_i$ is empty.*

3.2 Squares

We will be interested in rectangles $R = A \times B$, where A, B both are subsets of $(\{0,1\}^n)^{[p]}$, and which have a certain "square-like" structure. Such a "square-like" rectangle appears in our proofs, and will always be a sub-rectangle of the rectangle induced by a protocol.

A "square-like" rectangle $R = A \times B$, is one for which we have a set $I \subseteq [p]$ of *live* coordinates, with a corresponding set $D = [p] \setminus I$ of *dead* coordinates, and also a family $S \subseteq (\{0,1\}^n)^I$, for which one can do the following:

$$\begin{gathered}\text{For any } s \in S, \text{ there exist } \alpha(s), \beta(s) \in (\{0,1\}^n)^D, \text{ such that} \\ A \text{ is exactly the set of all } s \times \alpha(s) \text{ and } B \text{ is exactly the set of all } s \times \beta(s), \\ \text{and, furthermore, } \alpha(s)_i \neq \beta(t)_i \text{ for every } s \in S, t \in S, i \in D.\end{gathered}$$

i.e., given any s in S, which is a way of filling the live coordinates, there are two ways of filling the dead coordinates, $\alpha(s)$ and $\beta(s)$, such that the various $s \times \alpha(s)$ will be Alice's side of the rectangle, and the various $s \times \beta(s)$ will be Bob's side of the rectangle; furthermore, $\alpha(s)_i \neq \beta(t)_i$ always holds. We will call such a configuration a *square*:

▶ **Definition 25** (Square). *A* square *is a tuple* $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ *where:*

- $n \geq 1$, $p \geq 2$ *are integers;*
- $R = A \times B$ *where* $A, B \subseteq (\{0,1\}^n)^{[p]}$*;*
- $\varnothing \neq I \subseteq [p]$ *is a non-empty set of so-called* live *coordinates, and*
- $D = [p] \setminus I$ *is the corresponding set of* dead *coordinates;*
- $S \subseteq (\{0,1\}^n)^I$*;*
- $\alpha : S \to (\{0,1\}^n)^D$ *and* $\beta : S \to (\{0,1\}^n)^D$ *are such that* $A = \{s \times \alpha(s) \mid s \in S\}$ *and* $B = \{s \times \beta(s) \mid s \in S\}$*;*
- *for every* $s \in S$, $t \in S$, $i \in D$, *we have* $\alpha(s)_i \neq \beta(t)_i$.

▶ **Definition 26.** *The* density *of square* $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ *is given by* $\mathsf{Density}(\mathcal{S}) = \frac{|S|}{2^{n|I|}}$.

▶ **Definition 27.** *We say a square* $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ *is* τ*-thick on* $F \subseteq I$ *if* S *is* τ*-thick on* F*, and is* φ*-average-thick on* F *if* S *is* φ*-average-thick on* F.

One may justify the name *square* by the observation that a square $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ induces a bijection between A and B, where $s \times \alpha(s) \in A$ corresponds to $s \times \beta(s) \in B$.

4 The projection lemma

The main technical lemma of our simulation theorem is a *projection lemma*, which allow us to constrain coordinates of a square while preserving thickness, in such a way that $\alpha(s)_i \neq \beta(t)_i$ always holds.

▶ **Lemma 28.** *Let* $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ *be a square and* $\tau, \varphi \in [0,1]$ *be real numbers. Suppose that* $p \leq \frac{1}{12} \cdot 2^{\tau \cdot 2^n}$*. Suppose also that* $\mathcal{S}$ *is* τ*-thick, but not* φ*-average-thick, on* $F \subseteq I$.

Then, for any $z \in \{0,1\}^F$*, there exists a non-empty set* $E = E(z) \subseteq F$ *such that, letting* $E_0 = \{i \in E \mid z_i = 0\}$*, we may construct a square* $\mathcal{S}' = \mathcal{S}'(z) = \langle n, p, R' = A' \times B', I', S', \alpha', \beta' \rangle$*, where:*

- **(i)** $A' \subseteq A$ *and* $B' \subseteq B$*;*
- **(ii)** $I' = I \setminus E_0$*;*
- **(iii)** $\mathsf{Density}(\mathcal{S}') \geq (\frac{1}{2\varphi})^{|E_0|} \cdot \mathsf{Density}(\mathcal{S})$*; and*
- **(iv)** $\mathcal{S}'$ *is* $\frac{1}{2}\varphi$*-average-thick on* $F \setminus E$.

Furthermore, the set $E = E(z) \subseteq F$ *is obtained by a query procedure on the string* z*, and is exactly the set of positions queried by this procedure.*

Proof. We will explain the projection procedure in three steps. The entire procedure is achieved by running Procedure 1, 2 and 3 one after another (see below).

To begin with, S is not φ-average-thick on F, and so we are assured we will add at least one coordinate to E. Every time we add an index i to E we have, immediately prior to this, that $\frac{|S_{I \setminus E_0}|}{|S_{I \setminus (E_0 \cup \{i\})}|} \leq \varphi \cdot 2^n$, and hence $|S_{I \setminus (E_0 \cup \{i\})}| \geq \frac{|S_{I \setminus E_0}|}{\varphi \cdot 2^n}$. This means that if $z_i = 0$ and we then add i to E_0, we will have $|S_{I \setminus E_0}|$ grow by a factor of $\varphi \cdot 2^n$. By the end of this process, $S_{I \setminus E_0}$ must be φ-average-thick on $F \setminus E$ (otherwise we would add another coordinate to E), and furthermore $|S_{I \setminus E_0}| \geq \frac{|S|}{(\varphi \cdot 2^n)^{|E_0|}}$, which is to say

$$\frac{|S_{I \setminus E_0}|}{2^{|I \setminus E_0| \cdot n}} \geq \frac{1}{\varphi^{|E_0|}} \cdot \frac{|S|}{2^{|I| \cdot n}}. \tag{$*$}$$

This will later ensure our density increase.

▶ **Procedure 1.** Choosing E.

- *We start by letting* $E = \varnothing$.
- *As long as* $S_{I\setminus E_0}$ *is not* φ*-average-thick on* $F \setminus E$*, there exists some* $i \in F \setminus E$ *such that*

$$\frac{|S_{I\setminus E_0}|}{|S_{I\setminus(E_0\cup\{i\})}|} \leq \varphi \cdot 2^n.$$

- *We will then add* i *to* E *and query* z_i *(to know if* $i \in E_0$ *or not).*

▶ **Procedure 2.** Choosing $W = (U_i, V_i)_{i\in E_0}$, X and Y.

- *Independently for each* $i \in E_0$*, choose a partition* $\{0,1\}^n = U_i \uplus V_i$*, so that each string* $x \in \{0,1\}^n$ *is placed in* U_i *with probability* $\frac{1}{2}$*, and is placed in* V_i *otherwise. Let us use* $W = (U_i, V_i)_{i\in E_0}$ *to denote all the partitions chosen in this step.*
- *Now let us start by letting* $X = Y = S$.
- *Then for each index* $i \in E_0$ *in turn, we change* X *to* $X^{i,U_i}_{\neq i}$ *and change* Y *to* $Y^{i,V_i}_{\neq i}$.

Now consider the Procedure 2. At the end of its execution, we have both $X, Y \subseteq S_{I\setminus E_0}$. Now we may ask how much of $S_{I\setminus E_0}$ survived inside both X and Y. Let us first consider the difficult case when $|E_0| \geq 1$. We make the following claim:

▷ Claim 29. If $|E_0| \geq 1$, then for some choice of the partitions $(U_i, V_i)_{i\in E_0}$ we will have $|X \cap Y| \geq \frac{1}{2} \cdot |S_{I\setminus E_0}|$.

Before proving this claim, let us see why it is enough to give us our new square $\mathcal{S}'$. Let $U \subseteq (\{0,1\}^n)^{E_0}$ be the product of the various U_i sets, for $i \in E_0$, and likewise let $V \subseteq (\{0,1\}^n)^{E_0}$ be the product of the various V_i sets, for $i \in E_0$. The square $\mathcal{S}'$ is chosen thus:

▶ **Procedure 3.** Choosing the square $\mathcal{S}'$.

- *We set* $S' = X \cap Y$.
- *For each* $s' \in S'$*, we choose a string* $u(s') \in U \cap \mathsf{Ext}_S(s') \subseteq (\{0,1\}^n)^{E_0}$*; such a* $u(s')$ *exists because of how* X *was constructed; letting* $s = s' \times u(s') \in S$*, for each* $i \in [p] \setminus I' = ([p] \setminus I) \cup E_0$*, set*

$$\alpha'(s')_i = \begin{cases} u(s')_i & \text{if } i \in E_0, \\ \alpha(s)_i & \text{if } i \in [p] \setminus I. \end{cases}$$

- *We proceed symmetrically to choose* $\beta'(s')$.
- A' *and* B' *are simply the images of* S' *under* α' *and* β'.

For any $s \in S$, $t \in S$ and $i \in ([p] \setminus I) \cup E_0$, we have $\alpha(s)_i \neq \beta(t)_i$. This follows, on coordinates $i \in E_0$ because U_i and V_i are disjoint, and on coordinates $i \in [p] \setminus I$ because square $\mathcal{S}$ has the same property for α and β.

Properties (i) and (ii) are by construction. Property (iii) is a calculation using Claim 29 and $(*)$:

$$\mathsf{Density}(\mathcal{S}') = \frac{|S'|}{2^{|I'|\cdot n}} \underset{\text{Claim 29}}{\geq} \frac{\frac{1}{2} \cdot |S_{I\setminus E_0}|}{2^{|I\setminus E_0|\cdot n}} \underset{\text{Using } (*)}{\geq} \frac{1}{2} \cdot \frac{1}{\varphi^{|E_0|}} \cdot \frac{|S|}{2^{|I|\cdot n}} = \frac{1}{2} \cdot \frac{1}{\varphi^{|E_0|}} \cdot \mathsf{Density}(\mathcal{S}).$$

Now Property (iii) follows using the fact that $|E_0| \geq 1$. Property (iv) follows by Claim 29, because $S_{I \setminus E_0}$ is φ-average-thick on $F \setminus E$, and S' is a subset of $S_{I \setminus E_0}$ with $|S'| \geq \frac{1}{2} \cdot |S_{I \setminus E_0}|$.

In the simple case when $|E_0| = 0$, we have $X = Y = S$, and so we set $\mathcal{S}'$ to be exactly $\mathcal{S}$. Properties (i) and (ii) are easy to check, and Property (iii) is trivial, and property (iii) holds even without the $1/2$ factor loss, by our choice of E.

Now to prove Claim 29. Let $\delta = 2^{-\tau \cdot 2^n}$. Let us think of a matrix M where the rows are indexed by the various possible $s' \in S_{I \setminus E_0}$ and the columns are indexed by the different possible choices $W = (U_i, V_i)_{i \in E_0}$. The entry $M(s', W)$ equals 1 if $s' \in X$, where X is obtained from S and $(U_i)_{i \in E_0}$ by Procedure 2. In other words, again denoting by U the product of the various sets U_i, we have $M(s', W) = 1$ iff $U \cap \mathsf{Ext}_S(s') \neq \varnothing$.

Now fix some $s' \in S_{I \setminus E_0}$, and let us estimate the probability that $M(s', W) = 1$, i.e. that $s' \in X$, over the randomized choice of W. At the beginning of Procedure 2, we have $X = S$, and X is τ-thick on F. Then for each index $i \in E_0 \subseteq F$ in turn, we will change X to $X^{i,U_i}_{\neq i}$. Before we do this for the first time, s' will have at least one extension $s \in \mathsf{Ext}_X(s') \subseteq (\{0,1\}^n)^{E_0}$; at this point X is τ-thick on F, and so, taking any extension $s'' \in \mathsf{Ext}_{X_{\neq i}}(s') \subseteq (\{0,1\}^n)^{E \setminus \{i\}}$, there will be at least $\tau \cdot 2^n$ strings $s''' \in \{0,1\}^n$ such that $(s' \times s'') \times_i s''' \in S$. Each of these strings s''' is placed in U_i with probability $1/2$; hence the probability that $(s' \times s'') \in X^{i,U_i}_{\neq i}$ is at least $1 - 2^{-\tau \cdot 2^n} = 1 - \delta$, i.e., some extension s'' of s' survived with at least $1 - \delta$ probability over the choice of this first U_i. By Lemma 24, changing X to $X^{i,U_i}_{\neq i}$ gives us a set which is again thick on $F \setminus \{i\}$. Hence we may apply the same reasoning to the next index in E_0.

Changing X in this way $|E_0|$ times, we conclude that, in the end, $\Pr[M(s', W) = 1] = \Pr[s' \in X] \geq (1 - \delta)^{|E_0|} \geq 1 - |E_0|\delta$, where the probability is with respect to the distribution of W given by the above process. Now call a certain choice of W X-good if the W-column of M has at least a $1 - 3|E_0|\delta$ fraction of the rows $s' \in S_{I \setminus E_0}$ with $M(s', W) = 1$. Then, by a standard averaging argument, we must have $\Pr[W \text{ is } X\text{-good}] > 1/2$ (where again the probability is with respect to the distribution of W).

Arguing in the same way with respect to Y, we conclude that the probability that W is Y-good will also be more than $\frac{1}{2}$. Hence there must exist a choice of W which is both X-good and Y-good. For this choice of W we will have both $|X|, |Y| \geq (1 - 3|E_0|\delta)|S_{I \setminus E_0}|$, and given that $X, Y \subseteq S_{I \setminus E_0}$, this implies that $|X \cap Y| \geq (1 - 6|E_0|\delta) \cdot |S_{I \setminus E_0}| \geq (1 - 6p\delta) \cdot |S_{I \setminus E_0}|$. This is at least $\frac{1}{2}|S_{I \setminus E_0}|$ by our assumed bound on p. The claim is thus proven. ◀

▶ **Lemma 30.** *Let $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ be a square which is τ-thick on $F \subseteq I$, and let $z \in \{0,1\}^p$ be such that $z_i = 1$ for every $i \in I \setminus F$, and $z_i = 0$ for every $i \in [p] \setminus I$. Then there exists some $(x, y) \in A \times B$ with $\mathsf{Eq}^p(x, y) = z$.*

Proof. This is proven very similarly to Lemma 28. Instead of using Procedure 1 to choose E and E_0, we choose them directly based on z.

If there are no $i_0 \in I$ with $z_{i_0} = 0$, then any $s \in S$ will give $\mathsf{Eq}^p(s \times \alpha(s), s \times \beta(s)) = z$. Otherwise, let $E = F \setminus \{i_0\}$, so that $E_0 = \{i \in I \mid i \neq i_0, z_i = 0\}$. We may then use Procedure 2 to construct sets X and Y such that $X, Y \subseteq S_{I \setminus E_0}$. Note now that Claim 29 will still hold, because it only requires that S be thick on F. We may then use Procedure 3 to construct $\mathcal{S}'$, and Properties (i) and (ii) will hold as before. $\mathcal{S}'$ is a square on coordinates $I \setminus E_0 = \{i \mid z_i = 1\} \cup \{i_0\}$. By Lemma 24, we know that $\mathcal{S}'$ is τ-thick on $\{i_0\}$ and thus there are two strings $s \in S'$ and $t \in S'$ with $s_{i_0} \neq t_{i_0}$ but $s_i = t_i$ for all $i \in I' \setminus \{i_0\}$. Then $x = s \times \alpha(s)$ and $y = t \times \beta(t)$ give us $\mathsf{Eq}^p(x, y) = z$. ◀

5 Lifting 0-query complexity

We now prove our main simulation theorem (Theorem 13). Suppose $p \leq 2^{n/100}$, and let us fix $\tau = 2^{-n/10}$ and $\varphi = 2^{-n/20}$. Suppose we are given a C-bit communication protocol π for $f \circ \mathsf{Eq}_n$. We will then construct a decision-tree τ for f. On input $z \in \{0,1\}^p$, τ will find a leaf v of the protocol-tree of π, such that the associated rectangle R_v has some $(x,y) \in R_v$ with $\mathsf{Eq}^p(x,y) = z$. The label of such a leaf then equals $f(\mathsf{Eq}^p(x,y)) = f(z,)$. We now present an informal description of τ, and in Algorithm 1 below we provide pseudocode for τ. We will then show that the algorithm for τ is correct, i.e. that it is always able to find such a leaf v, and then show that the number of 0-queries that τ makes is $O(\frac{C}{n})$, which completes the proof of Theorem 13.

Given an input $z \in \{0,1\}^p$, τ starts traversing a path from the root of the protocol tree of π. A variable v is maintained, indicating the node of the protocol tree of π which is the current-node during the ongoing simulation; associated with v is the rectangle R_v of inputs which cause the protocol to reach node v. The decision-tree τ, when traversing node v, maintains a rectangle $R = A \times B$ and a square $\mathcal{S} = \langle n, p, R = A \times B, I = F \cup O, S, \alpha, \beta \rangle$, such that R is a sub-rectangle of R_v. The set F corresponds to coordinates of the input z that were not queried yet, and O is set of coordinates i which have been queried and found to have $z_i = 1$. Throughout the execution of the algorithm, it is maintained as an invariant that the square $\mathcal{S}$ is τ-thick in the coordinates F. At the beginning, $I = F = [p]$, $O = \varnothing$, and $A = B = (\{0,1\}^n)^{[p]}$, so the invariant is trivially true.

In each iteration of the simulation, the algorithm checks whether $\mathcal{S}$ is φ-average-thick on F. If this fails to hold, the algorithm will use the projection lemma (Lemma 28) and change $\mathcal{S}$ to ensure this requirement, as follows. Using Procedure 1 of Lemma 28, it chooses the set $E \subseteq F$; this requires querying z_i for $i \in E$, and gives us the set $E_0 \subseteq E$ of coordinates where $z_i = 0$, and the set $E_1 = E \setminus E_0$ of coordinates where $z_i = 1$. The algorithm then uses Procedure 3 of Lemma 28 to construct a square $\mathcal{S}'$. Lemma 28 guarantees that $\mathcal{S}'$ is $\frac{\varphi}{2}$-average-thick on $F \setminus E$, and that $\mathsf{Density}(\mathcal{S}')$ grows by a factor of $(2\varphi)^{-|E_0|}$. If E_0 is non-empty, i.e. if we have made some 0 queries, the density will grow significantly; otherwise the density will not change. The algorithm proceeds with $\mathcal{S} = \mathcal{S}'$, $I = I \setminus E_0$, $O = O \cup E_1$, and $F = F \setminus E$.

Now the algorithm is promised to have a square $\mathcal{S}$ which is at least $\frac{1}{2}\varphi$-average-thick. The algorithm then proceeds to a child v_c of v which has at least $1/2$ fraction of the density of $\mathcal{S}$, as follows. Suppose Alice communicated in v, and for each $c \in \{0,1\}$, let $R_{v_c} = A_{v_c} \times B_{v_c}$ be the rectangle which π associates with v_c. We then fix a choice $c \in \{0,1\}$ such that $|R \cap R_{v_c}| \geq |R|/2$. Now consider the set $S' = \{s \in S \mid s \times \alpha(s) \in A_{v_c}\}$. This set is still $\frac{1}{4}\varphi$-average-thick. We may then apply Lemma 23, with $\delta = \frac{1}{2}$, to S', which gives us a subset $S'' \subseteq S'$ which is τ-thick on F. The new square $\mathcal{S}$ is then given by restricting α and β to the set S''. By changing $\mathcal{S}$ in this way, we have preserved a $\frac{1}{4}$ fraction of the density.

Eventually, when we reach a leaf node v of the protocol tree, we are left with a square $\mathcal{S}$ which is τ-thick on F. The algorithm outputs the labeling of R_v in π, and we will now argue that this must equal $f(z)$.

Correctness. Because π correctly solves $f \circ \mathsf{Eq}^n$, then for each leaf v of π we have $(x,y,\pi(v)) \in f$ for all $(x,y) \in R_v$; the rectangle R obtained at the termination of Algorithm 1 is a sub-rectangle of R_v for a leaf of π, hence $(x,y,\pi(v)) \in f$ for all $(x,y) \in R$. On the other hand, we have preserved a square $\mathcal{S} = \langle n, p, R = A \times B, I, S, \alpha, \beta \rangle$ which is τ-thick on $F \subseteq I$, and such that $z_i = 1$ for every $i \in O = I \setminus F$, and $z_i = 0$ for every $i \in [p] \setminus I$. Then Corollary 30 tells us that some pair $(x,y) \in R$ is such that $\mathsf{Eq}^p(x,y) = z$; hence $(z, \pi(v)) \in f$.

Algorithm 1 Decision-tree procedure τ.

Input: $z \in \{0,1\}^p$
Output: $f(z)$

Initialization: Set v to be the root of the protocol tree for π, $I = F = [p]$, $O = \varnothing$, $\mathcal{S} = \langle n, p, R, I, S, \alpha, \beta \rangle$, where $R = A \times B$, $A = B = S = (\{0,1\}^n)^{[p]}$, and α, β are the empty functions.
while v is not a leaf **do**
  if $\mathcal{S}$ is not φ-average-thick on F **then**
    Use Lemmas 28 and 23, to get $E \subseteq F$, $E_0 \subseteq E$, and
      a square $\mathcal{S}' = \langle n, p, R', I \setminus E_0, S', \alpha', \beta' \rangle$, such that
      (1) $\mathcal{S}'$ is τ-thick on $F \setminus E$, and
      (2) $\mathsf{Density}(\mathcal{S}') \geq \frac{1}{4} \frac{1}{\varphi^{|E_0|}} \mathsf{Density}(\mathcal{S})$.
    Update $\mathcal{S} = \mathcal{S}'$, $O = O \cup (E \setminus E_0)$,
      $F = F \setminus E$, $I = I \setminus E_0$.
  end if
  ▷ *At this point $\mathcal{S}$ is at least $\frac{1}{2}\varphi$-average-thick on F.*
  Choose $c \in \{0,1\}$ such that $|R \cap R_{v_c}| \geq \frac{1}{2}|R|$.
  Using Lemma 23, choose $\mathcal{S}' = \langle n, p, R', I, S', \alpha', \beta' \rangle$, such that
    (1) $R' \subseteq R \cap R_{v_c}$
    (2) $\mathsf{Density}(\mathcal{S}') \geq \frac{1}{4}\mathsf{Density}(\mathcal{S})$.
    (3) $\mathcal{S}'$ is τ-thick on F.
  Update $\mathcal{S} = \mathcal{S}'$, and $v = v_c$.
end while
Output $\pi(v)$.

Number of queries. In each time when the simulation goes down the protocol tree of π, $\mathsf{Density}(\mathcal{S})$ drops by a factor of at most $\frac{1}{4}$ and hence, in total, by a factor of 4^{-C}. For each set E of queries that the algorithm makes in a round, the density of the current square increases by a factor of $(2\varphi)^{-|E_0|}$ – this is Property (iii) of Lemma 28. So, if Q_0 is the total number of queries which the algorithm makes, and which are answered 0, then the total gain in $\mathsf{Density}(\mathcal{S})$ is at least $(2\varphi)^{-Q_0}$. Since the density can be at most 1, we have, $4^{-C} \cdot (2\varphi)^{-Q_0} \leq 1$, and so $Q_0 \leq \frac{-2C}{\log(2\varphi)} = \frac{2C}{\frac{n}{20}-1} = O\left(\frac{C}{n}\right)$. This concludes the proof. ◀

References

1 Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. *SIAM Journal on Computing*, 42(3):1327–1363, 2013.

2 Paul Beame, Toniann Pitassi, Nathan Segerlind, and Avi Wigderson. A direct sum theorem for corruption and the multiparty NOF communication complexity of set disjointness. In *Proceedings of the 20th CCC*, pages 52–66, 2005.

3 Anna Bernasconi and Bruno Codenotti. Spectral Analysis of Boolean Functions as a Graph Eigenvalue Problem. *IEEE Transactions on Computers*, 48(3):345–351, 1999.

4 Mark Braverman and Anup Rao. Information Equals Amortized Communication. *IEEE Transactions on Information Theory*, 60(10):6058–6069, 2014.

5 Mark Braverman, Anup Rao, Omri Weinstein, and Amir Yehudayoff. Direct Product via Round-Preserving Compression. In *Proceedings of the 40th ICALP*, pages 232–243, 2013.

6 Mark Braverman, Anup Rao, Omri Weinstein, and Amir Yehudayoff. Direct Products in Communication Complexity. In *Proceedings of the 54th FOCS*, pages 746–755, 2013.

7 Joshua Brody, Harry Buhrman, Michal Koucký, Bruno Loff, Florian Speelman, and Nikolay Vereshchagin. Towards a reverse Newman's theorem in interactive information complexity. In *Proceedings of the 28th CCC*, pages 24–33, 2013.

8 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

9 Amit Chakrabarti and Oded Regev. An Optimal Lower Bound on the Communication Complexity of Gap-Hamming-Distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012.

10 Arkadev Chattopadhyay. Discrepancy and the Power of Bottom Fan-in in Depth-three Circuits. In *Proceedings of the 48th FOCS*, pages 449–458, 2007.

11 Arkadev Chattopadhyay and Anil Ada. Multiparty Communication Complexity of Disjointness. Technical Report TR08-002, ECCC, 2008.

12 Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation Theorems via Pseudorandom Properties. *CoRR*, abs/1704.06807, 2017.

13 Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation beats richness: new data-structure lower bounds. In *Proceedings of the 50th STOC*, pages 1013–1020. ACM, 2018.

14 Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. How Limited Interaction Hinders Real Communication. In *Proceedings of the 56th FOCS*, 2016.

15 Irit Dinur and Or Meir. Toward the KRW Composition Conjecture: Cubic Formula Lower Bounds via Communication Complexity. *Computational Complexity*, 27(3):375–462, 2018.

16 Andrew Drucker. Improved direct product theorems for randomized query complexity. *Computational Complexity*, 21(2):197–244, 2012.

17 Jeff Edmonds, Russell Impagliazzo, Steven Rudich, and Jirí Sgall. Communication complexity towards lower bounds on circuit depth. *Computational Complexity*, 10(3):210–246, 2001.

18 Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized Communication Complexity. *SIAM Journal on Computing*, 24(4):736–750, 1995.

19 Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. In *Proceedings of the 50th STOC*, pages 902–911, 2018.

20 Dmitry Gavinsky, Troy Lee, and Miklos Santha. On the randomised query complexity of composition. *CoRR*, abs/1801.02226, 2018.

21 Dmitry Gavinsky, Or Meir, Omri Weinstein, and Avi Wigderson. Toward better formula lower bounds: an information complexity approach to the KRW composition conjecture. In *Proceedings of the 46th STOC*, pages 213–222, 2014.

22 Mika Göös, Rahul Jain, and Thomas Watson. Extension complexity of independent set polytopes. *SIAM Journal on Computing*, 47(1):241–269, 2018.

23 Mika Göös, TS Jayram, Toniann Pitassi, and Thomas Watson. Randomized Communication versus Partition Number. In *Proceedings of the 44th ICALP*, 2017.

24 Mika Göös, Pritish Kamath, Toniann Pitassi, and Thomas Watson. Query-to-communication Lifting for P^{NP}. In *Proceedings of the 32nd CCC*, 2017.

25 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. In *Proceedings of the 47th STOC*, pages 257–266. ACM, 2015.

26 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *Proceedings of the 56th FOCS*, 2015.

27 Mika Göös, Toniann Pitassi, and Thomas Watson. The landscape of communication complexity classes. *Computational Complexity*, pages 1–60, 2015.

28 Michelangelo Grigni and Michael Sipser. Monotone Separation of Logspace from NC. In *Proceedings of the 6th Annual Structure in Complexity Theory Conference*, pages 294–298, 1991.

29 Prahladh Harsha, Rahul Jain, David McAllester, and Jaikumar Radhakrishnan. The communication complexity of correlation. In *Proceedings of the 22nd CCC*, pages 10–23, 2007.

30 Johan Håstad and Avi Wigderson. Composition of the Universal Relation. In *Proceedings of the DIMACS Workshop*, pages 119–134, 1990.

31 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of Protocols for XOR Functions. *SIAM Journal on Computing*, 47(1):208–217, 2018.

32 Rahul Jain. New strong direct product results in communication complexity. *Journal of the ACM*, 62(3):20, 2015.

33 Rahul Jain, Hartmut Klauck, and Ashwin Nayak. Direct product theorems for classical communication complexity via subdistribution bounds. In *Proceedings of the 40th STOC*, pages 599–608, 2008.

34 Rahul Jain, Attila Pereszlényi, and Penghui Yao. A direct product theorem for the two-party bounded-round public-coin communication complexity. In *Proceedings of the 53rd FOCS*, pages 167–176, 2012.

35 Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. A direct sum theorem in communication complexity via message compression. In *Proceedings of the 20th ICALP*, pages 300–315, 2003.

36 Rahul Jain and Penghui Yao. A strong direct product theorem in terms of the smooth rectangle bound. *CoRR*, abs/1209.0263, 2012.

37 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*. Springer, 2012.

38 Mauricio Karchmer, Eyal Kushilevitz, and Noam Nisan. Fractional Covers and Communication Complexity. *SIAM Journal on Discrete Mathematics*, 8(1):76–92, 1995.

39 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-Logarithmic Depth Lower Bounds Via the Direct Sum in Communication Complexity. *Computational Complexity*, 5(3/4):191–204, 1995.

40 Mauricio Karchmer and Avi Wigderson. Monotone Circuits for Connectivity Require Super-Logarithmic Depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, 1990.

41 Iordanis Kerenidis, Sophie Laplante, Virginie Lerays, Jérémie Roland, and David Xiao. Lower bounds on information complexity via zero-communication protocols and applications. *SIAM Journal on Computing*, 44(5):1550–1572, 2015.

42 Pravesh K Kothari, Raghu Meka, and Prasad Raghavendra. Approximating rectangles by juntas and weakly-exponential lower bounds for LP relaxations of CSPs. In *Proceedings of the 49th STOC*, pages 590–603. ACM, 2017.

43 Alexander Kozachinskiy. Comment on Meir's paper The Direct Sum of Universal Relations. Available at the address `https://eccc.weizmann.ac.il/report/2017/128/comment/1/download/`.

44 Alexander Kozachinskiy. From Expanders to Hitting Distributions and Simulation Theorems. In *Proceedings of the 43rd MFCS*, pages 4:1–4:15, 2018.

45 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

46 James Lee, Raghu Meka, and Thomas Vidick. (Less) heavy lifting: from conic junta degree to non-negative rank. Presented in the workshop *Hardness Escalation in Communication Complexity and Query Complexity*, FOCS 2017.

47 Troy Lee, Adi Shraibman, and Robert Spalek. A Direct Product Theorem for Discrepancy. In *Proceedings of the 23rd CCC*, pages 71–80, 2008.

48 Bruno Loff and Sagnik Mukhopadhyay. Lifting Theorems for Equality. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:175, 2018.

49 László Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

50 Or Meir. The direct sum of universal relations. *Information Processing Letters*, 136:105–111, 2018.

51 Gatis Midrijanis. Exact quantum query complexity for total Boolean functions. *CoRR*, abs/quant-ph/0403168, 2004.

52 Alon Orlitsky. Worst-case interactive communication. I. Two messages are almost optimal. *IEEE Transactions on Information Theory*, 36(5):1111–1126, 1990.

53 Denis Pankratov. Direct sum questions in classical communication complexity. Master's thesis, University of Chicago, 2012.

54 Toniann Pitassi and Robert Robere. Lifting nullstellensatz to monotone span programs over any field. In *Proceedings of the 50th STOC*, pages 1207–1219, 2018.

55 Anup Rao and Amir Yehudayoff. Simplified Lower Bounds on the Multiparty Communication Complexity of Disjointness. In *Proceedings of the 30th CCC*, pages 88–101, 2015.

56 Ran Raz. Fourier analysis for probabilistic communication complexity. *Computational Complexity*, 5(3-4):205–221, 1995.

57 Ran Raz and Pierre McKenzie. Separation of the Monotone NC Hierarchy. *Combinatorica*, 19(3):403–435, 1999.

58 Robert Robere. *Unified Lower Bounds for Monotone Computation*. PhD thesis, University of Toronto, 2018.

59 Robert Robere, Toniann Pitassi, Benjamin Rossman, and Stephen A Cook. Exponential lower bounds for monotone span programs. In *Proceedings of the 57th FOCS*, pages 406–415, 2016.

60 Swagato Sanyal. A Composition Theorem via Conflict Complexity. *CoRR*, abs/1801.03285, 2018. `arXiv:1801.03285`.

61 Alexander A Sherstov. The pattern matrix method. *SIAM Journal on Computing*, 40(6):1969–2000, 2011.

62 Alexander A. Sherstov. The Communication Complexity of Gap Hamming Distance. *Theory of Computing*, 8(1):197–208, 2012.

63 Alexander A. Sherstov. The multiparty communication complexity of set disjointness. In *Proceedings of the 44th STOC*, pages 525–548, 2012.

64 Alexander A. Sherstov. Communication lower bounds using directional derivatives. In *Proceedings of the 45th STOC*, pages 921–930, 2013.

65 Yaoyun Shi and Yufan Zhu. Quantum communication complexity of block-composed functions. *Quantum Information & Computation*, 9(5):444–460, 2009.

66 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier Sparsity, Spectral Norm, and the Log-Rank Conjecture. In *Proceedings of the 54th FOCS*, pages 658–667, 2013.

67 Thomas Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the Gap-Hamming-Distance problem. *Chicago Journal of Theoretical Computer Science*, 2013, 2013.

68 Thomas Watson. A $\mathsf{ZPP}^{\mathsf{NP}}$ Lifting Theorem. *Unpublished preprint*, 2017.

69 David P. Woodruff. *Efficient and private distance approximation in the communication and streaming models*. PhD thesis, Massachusetts Institute of Technology, 2007.

70 Andrew Chi-Chih Yao. Some Complexity Questions Related to Distributive Computing (Preliminary Report). In *Proceedings of the 11h STOC*, pages 209–213, 1979.

Car-Sharing on a Star Network: On-Line Scheduling with k Servers

Kelin Luo
School of Management, Xi'an Jiaotong University, Xi'an, China
luokelin@stu.xjtu.edu.cn

Thomas Erlebach
Department of Informatics, University of Leicester, Leicester, United Kingdom
te17@leicester.ac.uk

Yinfeng Xu
School of Management, Xi'an Jiaotong University, Xi'an, China
yfxu@xjtu.edu.cn

Abstract

We study an on-line scheduling problem that is motivated by applications such as car-sharing for trips between an airport and a group of hotels. Users submit ride requests, and the scheduler aims to accept requests of maximum total profit using k servers (cars). Each ride request specifies the pick-up time, the pick-up location, and the drop-off location, where one of the two locations must be the airport. A request must be submitted a fixed amount of time before the pick-up time. The scheduler has to decide whether or not to accept a request immediately at the time when the request is submitted (booking time). In the unit travel time variant, the travel time between the airport and any hotel is a fixed value t. We give a 2-competitive algorithm for the case in which the booking interval (pick-up time minus booking time) is at least t and the number of servers is even. In the arbitrary travel time variant, the travel time between the airport and a hotel may have arbitrary length between t and Lt for some $L \geq 1$. We give an algorithm with competitive ratio $O(\log L)$ if the number of servers is at least $\lceil \log L \rceil$. For both variants, we prove matching lower bounds on the competitive ratio of any deterministic on-line algorithm.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Car-Sharing System, On-Line Scheduling, Competitive Analysis, Star Network

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.51

Funding *Kelin Luo*: This work was partially supported by the China Postdoctoral Science Foundation (Grant No. 2016M592811), and the China Scholarship Council (Grant No. 201706280058).

1 Introduction

In a car-sharing system, customers can hire a car from a company for a period of time. They can pick up a car in one location, drive it to another location, and return it there. Customer requests for car bookings arrive over time, and the decision about each request must be made immediately, without knowledge of future requests. The goal is to maximize the profit obtained from satisfied requests. We refer to this problem as the *car-sharing problem*. Similar problems arise in car rental or taxi dispatching. In this paper, we consider the setting where all car booking requests are for travel between a central location (e.g., an airport, shopping mall or central business district) and one of a group of nearby locations (e.g., hotels, or residential areas), but can be in either direction. The connections between the central location and the nearby locations can therefore be viewed as a star network.

The car-sharing problem bears some resemblance to interval scheduling, but in addition the pick-up and drop-off locations play an important role: The server (car) that serves a request must be at the pick-up location at the start time of the request and will be at the drop-off location at the end time of the request.

A server can serve two consecutive requests, where the pick-up time of the second is no earlier than the drop-off time of the first, only if the drop-off location of the first request is the same as the pick-up location of the second request, or if there is enough time to travel between the two locations otherwise. We allow *empty movements*, i.e., a server can be moved from one location to another while not serving a request. Such empty movements could be implemented by having company staff drive a car from one location to another, or in the future by self-driving cars.

1.1 Related work

On-line car-sharing problem. The car-sharing problem has been studied in several previous papers. In [9], we considered the special case with two locations and a single server, considering both fixed booking times and variable booking times, and presented tight results for the competitive ratio. The optimal competitive ratio was shown to be 2 for fixed booking times and 3 for variable booking times. In [10], we dealt with the car-sharing problem with two locations and two servers, considering only the case of fixed booking times, and showed that the optimal competitive ratio is 2. In [11], we studied the car-sharing problem with two locations and k servers, where k can be arbitrarily large. We considered both fixed booking times and variable booking times. The results showed that, surprisingly, 3 servers (in one case) and 5 servers (in another case) already allow us to get the best competitive ratio, and no improvement is possible with more servers. In contrast to the previous work on car-sharing that has only considered two locations, in this paper we study the car-sharing problem for fixed booking time in the setting with k servers and $m + 1$ locations that are arranged in a star network.

Off-line car-sharing problem. Böhmová et al. [4] showed that if all customer requests for car bookings are known in advance, the problem of maximizing the number of accepted requests is solvable in polynomial time. Furthermore, they considered the problem variant with two locations where each customer requests two rides (in opposite directions) and the scheduler must accept either both or neither of the two. They proved that this variant is NP-hard and APX-hard. In contrast to their work, we consider the on-line version of the problem with $m + 1$ locations.

On-line dial-a-ride problem. A closely related problem is the on-line dial-a-ride problem (OLDARP). Versions of OLDARP with the objective of serving all requests while minimizing the makespan [1, 3] or the maximum flow time [7] have been widely studied in the literature. Versions of OLDARP where not all requests need to be served include the setting where each request must be served before its deadline or rejected [12], and the setting with a given common time limit where the goal is to maximize the revenue from requests served before the time limit [6]. In OLDARP, transportation requests between locations in a metric space arrive over time, but typically it is assumed that requests want to be served "as soon as possible" rather than at a specific time as in our problem.

On-line interval scheduling problem. The on-line car-sharing problem can be interpreted as a variation of the on-line interval scheduling problem. If all the pick-up and drop-off locations are the same, the car-sharing problem becomes an on-line interval scheduling

problem. Lipton and Tomkins [8] defined the basic on-line interval scheduling problem: intervals with a given length are presented in the order of their start time and the scheduler aims to accept intervals of maximum total length. The scheduler has to decide whether to accept each interval before the next interval is presented and ensure that no pair of accepted intervals overlap. They showed that no (randomized) algorithm can achieve competitive ratio $O(\log \Delta)$ (where Δ denotes the ratio between the longest and the shortest interval, and Δ is unknown to the algorithm), and gave an $O((\log \Delta)^{1+\varepsilon})$-competitive randomized algorithm.

1.2 Problem description and preliminaries

We consider a setting with $m+1$ locations in a star network and denote the central location by 0 and the other locations by i for $i \in \{1, 2, \ldots, m\}$. There are k servers, denoted by $S = \{s_1, s_2, \ldots, s_k\}$, that are initially located at 0. We assume that $m \geq 2$ since, if $m = 1$, the problem turns into the car-sharing problem between two locations that has been studied before [9, 10, 11]. The length of the edge between 0 and i, for $1 \leq i \leq m$, is denoted by $d(0, i) = d(i, 0)$. The travel time from 0 to i, $i \in \{1, 2, \ldots, m\}$, is $d(0, i) \cdot t$, where t is a fixed positive constant, and is the same as the travel time from i to 0, $d(i, 0) \cdot t$. In the variant with unit travel times, all edges have length 1 and the travel time between 0 and i is t for all $i \in \{1, 2, \ldots, m\}$ (see Fig. 1 for an example). In the variant with arbitrary travel times, we only assume that the edge lengths satisfy $1 \leq d(0, i) \leq L$ for all $1 \leq i \leq m$ (Fig. 2 shows an example).

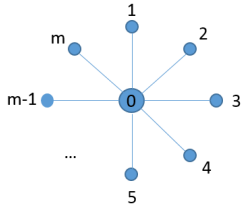


Figure 1 Unit travel times.

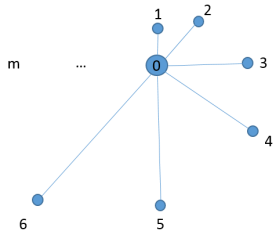


Figure 2 Arbitrary travel times.

Let R denote a sequence of requests that are released over time. The requests with the same release time are released one by one in arbitrary order. The i-th request is denoted by $r_i = (\tilde{t}_{r_i}, t_{r_i}, p_{r_i}, \dot{p}_{r_i})$ and specifies the *booking time* or *release time* $\tilde{t}_{r_i}$, the *start time* or *pick-up time* t_{r_i}, the pick-up location $p_{r_i} \in \{0, 1, \ldots, m\}$, and the drop-off location $\dot{p}_{r_i} \in \{0, 1, \ldots, m\}$. We also say that request r_i *drops off* at $\dot{p}_{r_i}$. We require $\dot{p}_{r_i} \neq p_{r_i}$ and $\min\{p_{r_i}, \dot{p}_{r_i}\} = 0$, i.e., for all requests $r_i \in R$, either the pick-up location p_{r_i} or the drop-off location $\dot{p}_{r_i}$ is 0. We assume that the *booking interval* $t_{r_i} - \tilde{t}_{r_i}$ is equal to a fixed value a for all requests $r_i \in R$. For the variant with unit travel times (resp. the variant with arbitrary travel times), if r_i is accepted, a server must pick up the customer at p_{r_i} at time t_{r_i} and drop off the customer at $\dot{p}_{r_i}$ at time $\dot{t}_{r_i} = t_{r_i} + t$ (resp. at time $\dot{t}_{r_i} = t_{r_i} + d(p_{r_i}, \dot{p}_{r_i}) \cdot t$), the *end time* (or *drop-off time*) of the request.

Each server can only serve one request at a time. If two requests are such that they cannot both be served by one server, we say that the requests are *in conflict*. For the variant with unit travel times (resp. arbitrary travel times), serving a request r_i yields profit $P_{r_i} = r$ (resp. $P_{r_i} = d(p_{r_i}, \dot{p}_{r_i}) \cdot r$). An empty movement has no cost. We denote the requests accepted by an algorithm by R'. The i-th request in R', in order of request start times, is denoted by r'_i. We denote the profit of serving the requests in R' by $P_{R'} = \sum_{i=1}^{|R'|} P_{r'_i}$. The goal of the car-sharing problem is to accept a set of requests R' that maximizes the profit $P_{R'}$.

We use *kSmL-U* to refer to the problem with k servers and $m+1$ locations with unit travel times. The problem variant with arbitrary travel times is called the *kSmL-A* problem. For the kSmL-A problem, we assume that $a \geq 2Lt$, where Lt is the travel time of the longest edge of the star. This ensures that any free server always has enough time to travel from its current location to the pick-up location of a newly accepted request. We do not require that the algorithm assigns an accepted request to a server immediately, provided that it ensures that one of the k servers will serve the request.

We forbid "unprompted" moves, i.e., the algorithm is allowed to make an empty move to another location only if it does so in order to serve a request that was accepted before the current time and whose pick-up location is the other location. If the length of the booking interval (recall that the booking interval is the interval between booking time and start time) is greater than the maximum travel time of any two locations in the two problems defined above, we observe that there is never a need for a server to make an unprompted movement. Therefore, if $a \geq 2t$ for kSmL-U or $a \geq 2Lt$ for kSmL-A, whether or not we forbid unprompted movements affects neither lower bounds nor the algorithm performance.

The performance of an algorithm for kSmL-U or kSmL-A is measured using competitive analysis [5]. For any request sequence R, let P_{R^A} denote the objective value produced by an on-line algorithm A, and P_{R^*} that obtained by an optimal scheduler OPT that has full information about the request sequence in advance. Like for the algorithm, we also require that OPT does not make unprompted moves, i.e., OPT is allowed to make an empty move starting at time t_0 with some server s_j from location p to location q only if there is an accepted request r_i assigned to s_j with $\tilde{t}_{r_i} \leq t_0$, $p_{r_i} = q$ and $t_{r_i} \geq t_0 + d(p,q) \cdot t$. The competitive ratio of A is defined as $\rho_A = \sup_R \frac{P_{R^*}}{P_{R^A}}$. We say that A is ρ-competitive if $P_{R^*} \leq \rho \cdot P_{R^A}$ for all request sequences R. Let ON be the set of all on-line algorithms for a problem. We only consider deterministic algorithms in this paper. A value β is a *lower bound* on the best possible competitive ratio if $\rho_A \geq \beta$ for all A in ON.

The asymptotic competitive ratio (asymptotic performance ratio) of A is defined to be $\rho'_A = \lim_{n\to\infty} \sup_R \{\frac{P_{R^*}}{P_{R^A}} | P_{R^*} = n\}$. A value β' is a *lower bound* on the best possible asymptotic competitive ratio if $\rho'_A \geq \beta'$ for all A in ON. We write $\mathbb{N} = \{0, 1, 2, \ldots\}$.

1.3 Paper outline

Table 1 Lower and upper bounds on the competitive ratio for the kSmL problem.

Problem	Booking constraint	Lower bound	Upper bound
kSmL-U	$a < t$	$\frac{k}{\lfloor k/m \rfloor}$	$\frac{k}{\lfloor k/m \rfloor}$
kSmL-U	$t \leq a < 2t$	2	2 (for even k)
kSmL-U	$a \geq 2t$	$2 - \frac{1}{2m-1}$	2 (for even k)
kSmL-A	$a \geq 2Lt$	$\Omega(\log L)$	$O(\log L)$ (for $k \geq \log L$)

In Section 2, we present lower bounds on the competitive ratio for the kSmL-U problem. In Section 3, we propose two greedy algorithms, the m-partition greedy algorithm and the bi-partition greedy algorithm, that achieve the best possible competitive ratio for kSmL-U for different ranges of a. In Section 4, we study kSmL-A and give an algorithm with competitive ratio $O(\log L)$ and show that no deterministic on-line algorithm can achieve competitive ratio smaller than $\Omega(\log L)$. Section 5 concludes the paper. An overview of our results is shown in Table 1. All our lower bounds hold even in the seemingly simpler case where the start time of every request is a multiple of t.

2 Lower bounds for kSmL-U

In this section, we present lower bounds for the kSmL-U problem. We use ALG to denote any on-line algorithm and OPT to denote an optimal scheduler. The set of requests accepted by ALG is denoted by R', and the set of requests accepted by OPT by R^*.

▶ **Theorem 1.** *For $a < t$, no deterministic on-line algorithm for kSmL-U can achieve asymptotic competitive ratio smaller than $\frac{k}{\lfloor k/m \rfloor}$.*

Proof. Consider a sequence of requests that consists of γ phases where phase i, for $1 \leq i \leq \gamma$, consists of l_i groups of requests, with each group consisting of k identical requests. Let $\sigma(u, v)$ be the number of request groups that the adversary has released by the time when the requests in phase u, group v have just been released, i.e., $\sigma(u, v) = \sum_{i=1}^{u-1} l_i + v$.

The adversary releases requests based on the release rule for kSmL-U shown in Algorithm 1.

Algorithm 1 Release Rule for kSmL-U with $a < t$.

Initialization: The adversary presents the requests in phase 1 group 1: k copies of the request $(\nu \cdot t - a, \nu \cdot t, 0, 1)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \geq 0$.
$i = 1$, $j = 1$. Let l be a large, positive, odd integer.
While $i \leq m$ do
  if $j < l$ then
    if $|R'_{i,j}| > \lfloor k/m \rfloor$ then
      $l_i = j$, $i = i + 1$, $j = 1$ and the adversary releases the requests in $R_{i,j}$;
    else if $|R'_{i,j}| \leq \lfloor k/m \rfloor$ then
      $j = j + 2$ and the adversary releases the requests in $R_{i,j-1}$ and $R_{i,j}$;
  if $j \geq l$ then
      break.
Output: $\gamma = i$ and $l_i = j$.
(1) Let $R_{i,j}$ denote the set of requests in phase i group j. If $i > 1$ and $j = 1$, $R_{i,j}$ consists of k copies of the request $(\tilde{t}_{r_{\sigma(i-1,l_{i-1})k}} + t, t_{r_{\sigma(i-1,l_{i-1})k}} + t, 0, i)$; if $i > 0$, $j > 1$ and $j = 2e$ where $e \in \mathbb{N}$, $R_{i,j}$ consists of k copies of the request $(\tilde{t}_{r_{\sigma(i,j-1)k}} + t, t_{r_{\sigma(i,j-1)k}} + t, i, 0)$; if $i > 0$, $j > 1$ and $j = 2e + 1$ where $e \in \mathbb{N}$, $R_{i,j}$ consists of k copies of the request $(\tilde{t}_{r_{\sigma(i,j-1)k}} + t, t_{r_{\sigma(i,j-1)k}} + t, 0, i)$.
(2) Let $R'_{i,j}$ denote the set of requests accepted by ALG in phase i group j.

We make four observations:

(a) For each $i < \gamma$, $l_i < l$. This holds because, as soon as j reaches value l, the While-loop is exited and γ is set to i.

(b) For each $i \leq \gamma$, ALG accepts no more than $\lfloor k/m \rfloor (l_i - 1)$ requests in total among the requests in phase i excluding the requests in phase i group l_i. This can be seen as follows: The algorithm accepts at most $\lfloor k/m \rfloor$ requests from phase i group j for any odd j, $j < l_i$. Moreover, the total number of requests from phase i group j for all even j together cannot be larger than the total number of requests from phase i group j for all odd j, $j < l_i$.

(c) ALG accepts no more than k requests in total among the requests in phase 1 group l_1, phase 2 group l_2, ..., phase γ group l_γ. This holds because any server accepting a request in phase i group l_i will remain at i and not be able to serve any further requests.

(d) ALG accepts more than $(\lfloor k/m \rfloor + 1)(\gamma - 1)$ requests in total among the requests in phase 1 group l_1, phase 2 group l_2, ..., phase $\gamma - 1$ group $l_{\gamma-1}$. This holds because the algorithm accepts strictly more than $\lfloor k/m \rfloor$ requests in each of these $\gamma - 1$ groups.

According to (d), more than $(\lfloor k/m \rfloor + 1) \cdot (\gamma - 1)$ servers are not in location 0 or γ when the requests in phase γ are released. These servers cannot accept requests in phase γ because the release time of a request is too late for such a server to be able to serve it with empty movement. If $\gamma = m$, $k - (\lfloor k/m \rfloor + 1)(\gamma - 1) \leq \lfloor k/m \rfloor$, and hence the adversary stops to release requests in phase m only after group l. Therefore, $l_\gamma = l$ no matter whether $\gamma < m$ or $\gamma = m$.

By (b) and (c), we have that ALG accepts no more than $(\lfloor k/m \rfloor \sum_{i=1}^{\gamma}(l_i - 1)) + k$ requests.

OPT accepts all the requests except the requests in phase 1 group l_1, phase 2 group l_2, ..., phase $\gamma - 1$ group $l_{\gamma-1}$. We have $P_{R^*} = (kr \sum_{i=1}^{\gamma-1}(l_i - 1)) + krl$. Since $P_{R'} \leq kr + \lfloor k/m \rfloor \sum_{i=1}^{\gamma}(l_i - 1)r$ and $\lim_{l\to\infty} \inf_{l_i \leq l} \frac{kl + k\sum_{i=1}^{\gamma-1}(l_i-1)}{k + \lfloor k/m \rfloor \sum_{i=1}^{\gamma}(l_i-1)} = \frac{k}{\lfloor k/m \rfloor}$, where the infimum is taken over all possible values of l_i for $1 \leq i \leq \gamma - 1$, we get $\lim_{P_{R^*}\to\infty} P_{R^*}/P_{R'} \geq \frac{k}{\lfloor k/m \rfloor}$. ◀

▶ **Theorem 2.** *For $t \leq a < 2t$, no deterministic on-line algorithm for kSmL-U can achieve asymptotic competitive ratio smaller than 2.*

Proof. Let l be a large, positive integer. Consider a sequence of requests that consists of 2 phases where phase i, for $i = 1, 2$, consists of l_i groups of requests, with each group consisting of k identical requests. Let $R_{i,j}$ denote the set of requests in phase i group j. Initially, the adversary releases $R_{i,j}$ with $i = 1$ and $j = 1$, consisting of k copies of the request $r_1 = (\nu \cdot t - a, \nu \cdot t, 0, 1)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \geq 0$. Let $R'_{i,j}$ denote the set of requests accepted by ALG in phase i group j. The adversary releases further requests based on the following rules: If $|R'_{1,j}| \leq \frac{k}{2}$ and $j < l$, let $j = j + 1$ and release $R'_{1,j}$ consisting of k copies of the request $(\tilde{t}_{r_1} + 2(j-1)t, t_{r_1} + 2(j-1)t, 0, 1)$; otherwise, set $l_1 = j$ and stop to release requests in phase 1. Note that either $l_1 = l$, or $l_1 < l$ and $|R'_{1,l_1}| > \frac{k}{2}$. We distinguish two cases.

Case 1: $l_1 = l$. Observe that $|R'_{1,j}| \leq \frac{k}{2}$ for all $1 \leq j < l_1$. In this case, OPT accepts all requests in $R_{1,j}$ for all $1 \leq j \leq l$. We have $P_{R^*} = l \cdot kr$ and $P_{R'} = \sum_{j=1}^{l_1} |R'_{1,j}|r \leq \frac{k}{2} \cdot (l-1)r + kr$, and hence $\lim_{P_{R^*}\to\infty} P_{R^*}/P_{R'} \geq 2$.

Case 2: $l_1 < l$ and $|R'_{1,l_1}| > \frac{k}{2}$. Observe that $|R'_{1,j}| \leq \frac{k}{2}$ for all $1 \leq j < l_1$. The adversary then releases $R_{2,j}$ for all $1 \leq j \leq l^2$, where each $R_{2,j}$ consists of k copies of the request $(\tilde{t}_{r_1} + 2(l_1 - 1 + j)t, t_{r_1} + 2(l_1 - 1 + j)t, 2, 0)$. Observe that $|R'_{1,l_1}|$ servers of the algorithm are in location 1 when the requests in $R_{2,j}$ for all $1 \leq j \leq l^2$ are released. The release time of a request with pick-up location 2 is too late for a server in location 1 to be able to serve it with empty movement because the travel time between location 1 and the pick-up location 2 is $2t$, which is greater than the booking interval a. From this it follows that the $|R'_{1,l_1}|$ servers of ALG cannot accept any requests in phase 2. OPT accepts all requests in phase 2. We have $P_{R^*} \geq l^2 \cdot kr$. Since $P_{R'} \leq \sum_{j=1}^{l_1-1} |R'_{1,j}|r + |R'_{1,l_1}|r + (k - |R'_{1,l_1}|)l^2 r \leq \frac{k}{2} \cdot (l^2 + l_1 - 1)r + |R'_{1,l_1}|r$, we have $\lim_{P_{R^*}\to\infty} P_{R^*}/P_{R'} \geq 2$. ◀

▶ **Theorem 3.** *For $a \geq 2t$, no deterministic on-line algorithm for kSmL-U can achieve competitive ratio smaller than $2 - \frac{1}{2m-1}$. Furthermore, if $k < 2(m-1)$, no deterministic on-line algorithm for kSmL-U can achieve competitive ratio smaller than 2.*

Proof. The adversary releases a number of request sequences. We use k_i $(0 \leq k_i \leq k)$ to denote the number of requests that ALG accepts from the i^{th} request sequence.

Initially, the adversary releases the 1^{st} request sequence, consisting of k copies of the request $(\nu \cdot t - a, \nu \cdot t, 0, 1)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \geq 0$. There are two options that the adversary can adopt:

Option 1. The adversary does not release any more requests. In this case, OPT accepts all requests in the 1^{st} request sequence. We have $P_{R^*} = k \cdot r$ and $P_{R'} = k_1 \cdot r$, and hence $P_{R^*}/P_{R'} \geq \frac{k}{k_1}$.

Option 2. The adversary releases the 2^{nd} request sequence, consisting of k copies of the request $(\tilde{t}_{r_1}, t_{r_1}, 1, 0)$, and then releases $m-1$ further request sequences (from the 3^{rd} request sequence to the $(m+1)^{th}$ request sequence), where the i^{th} $(3 \leq i \leq m+1)$ request sequence consists of k copies of the request $(\tilde{t}_{r_1} + t, t_{r_1} + t, 0, i-1)$. Then, the adversary releases the $(m+2)^{th}$ request sequence, consisting of k copies of the request $(\tilde{t}_{r_1} + 2t, t_{r_1} + 2t, \varrho - 1, 0)$ where $\varrho = \arg\min\{k_i, 3 \leq i \leq m+1\}$. Since the requests in the 1^{st} request sequence are in conflict with the requests in the i^{th} $(2 \leq i \leq m+2)$ request sequence, k_1 servers of ALG accept at most one request each. OPT accepts all requests in the 2^{nd} request sequence, the ϱ^{th} request sequence and the $(m+2)^{th}$ request sequence, i.e., $P_{R^*} = 3kr$.

Observe that $k_\varrho \leq \frac{k-k_1}{m-1}$. Furthermore, the requests in the $(m+2)^{th}$ request sequence are in conflict with the requests in the i^{th} $(3 \leq i \leq m+1$ and $i \neq \varrho)$ request sequence. Therefore, at most k_ϱ servers accept requests both in the $(m+2)^{th}$ request sequence and the i^{th} $(3 \leq i \leq m+1)$ request sequence. From this it follows that k_ϱ servers accept at most three requests each (in the 2^{nd}, the ϱ^{th}, and the $(m+2)^{th}$ request sequence) and the remaining servers of ALG, i.e., $k - k_1 - k_\varrho$ servers, each accept at most two requests (in the 2^{nd} and the i^{th} request sequence where $3 \leq i \leq m+1$ and $i \neq \varrho$). Thus, we have $P_{R'} \leq k_1 r + 2(k-k_1)r + k_\varrho \cdot r$ and hence $P_{R'} \leq k_1 r + 2(k-k_1)r + \frac{k-k_1}{m-1} \cdot r$. Since $P_{R^*} = 3kr$, $P_{R^*}/P_{R'} \geq \frac{3k}{2k-k_1+(k-k_1)/(m-1)}$.

If we choose the option (from Option 1 and Option 2) that maximizes $\frac{P_{R^*}}{P_{R'}}$, we have $\frac{P_{R^*}}{P_{R'}} \geq \max\{\frac{k}{k_1}, \frac{3k}{2k-k_1+(k-k_1)/(m-1)}\}$. As k_1 increases, $\frac{k}{k_1}$ decreases and $\frac{3k}{2k-k_1+(k-k_1)/(m-1)}$ increases. Since $\frac{3k}{2k-k_1+(k-k_1)/(m-1)} = \frac{k}{k_1} = 2 - \frac{1}{2m-1}$ when $k_1 = \frac{2m-1}{4m-3} \cdot k$, we have $P_{R^*}/P_{R'} \geq 2 - \frac{1}{2m-1}$. The claimed lower bound of $2 - \frac{1}{2m-1}$ follows.

Furthermore, if $k < 2(m-1)$, we can argue as follows. If $k_1 \leq \frac{k}{2}$, we choose Option 1 and get $P_{R^*}/P_{R'} = \frac{k}{k_1} \geq 2$. If $k_1 > \frac{k}{2}$, then $k_\varrho \leq \frac{k-k_1}{m-1} < 1$ and hence $k_\varrho = 0$. We choose Option 2 and have $P_{R'} \leq k_1 r + 2(k-k_1)r$ and thus $P_{R^*}/P_{R'} \geq \frac{3k}{2k-k_1} > 2$. The claimed lower bound of 2 follows. ◀

3 Upper bounds for kSmL-U

In this section, we prove the upper bounds for the kSmL-U problem. Denote the requests accepted by OPT by $R^* = \{r_1^*, r_2^*, \ldots, r_{|R^*|}^*\}$ and the requests accepted by an algorithm by $R' = \{r_1', r_2', \ldots, r_{|R'|}'\}$, indexed in the order in which they are released (and hence also in order of non-decreasing pick-up times).

Let $R^*(e,p,d)$ denote the set of requests in R^* which start at time e at location p and drop off at location d. Observe that $\forall e,p,d$, $|R^*(e,p,d)| \leq k$. Let $R^*(p,d)$ denote the set of requests in R^* which start at location p and drop off at location d. Furthermore, let $R^*(e,0,X)$ denote the set of requests in R^* which start at time e at location 0, i.e., $R^*(e,0,X) = \bigcup_{d=1}^{m} R^*(e,0,d)$, and let $R^*(e,X,0)$ denote the set of requests in R^* which start at time e and drop off at location 0, i.e., $R^*(e,X,0) = \bigcup_{d=1}^{m} R^*(e,d,0)$. Similarly, define $R^*(0,X) = \bigcup_{d=1}^{m} R^*(0,d)$ and $R^*(X,0) = \bigcup_{d=1}^{m} R^*(d,0)$. The subsets $R'(e,p,d)$, $R'(p,d)$, $R'(e,0,X)$, $R'(e,X,0)$ $R'(0,X)$ and $R'(X,0)$ of R' are defined analogously.

3.1 Upper bound for $0 \leq a < t$

We propose an m-Partition Greedy Algorithm (m-PGA) for the kSmL-U problem when $0 \leq a < t$, shown in Algorithm 2. The k servers are divided into m groups $S_1, S_2, \ldots, S_m$. From group S_1 to group S_{m-1}, each group has $\lfloor k/m \rfloor$ servers; group S_m has $k - \lfloor k/m \rfloor (m-1) \geq \lceil k/m \rceil$ servers. The servers in group S_i, $1 \leq i \leq m$, only serve requests whose pick-up or drop-off location is i.

Algorithm 2 m-Partition Greedy Algorithm (m-PGA).

Input: k servers, requests arrive over time.

Step: When request r_i arrives, if it is acceptable to a server in group $S_{g(r_i)}$, where $g(r_i) = \max\{p_{r_i}, \dot{p}_{r_i}\}$, assign it to that server; otherwise, reject it.

(1) $r^n_{i,j}$ denotes the newest request which is assigned to s_j before r_i is released. Set $\dot{p}_{r^n_{i,j}} = 0$ and $\dot{t}_{r^n_{i,j}} = 0$ if server s_j has not accepted any request before r_i is released.

(2) r_i is acceptable to a server s_j ($s_j \in S$) if and only if $p_{r_i} = \dot{p}_{r^n_{i,j}}$ and $t_{r_i} \geq \dot{t}_{r^n_{i,j}}$.

We refer to the servers of m-PGA as $s'_1, s'_2, \ldots, s'_k$, and the servers of OPT as $s^*_1, s^*_2, \ldots, s^*_k$. For an arbitrary request sequence $R = (r_1, r_2, \ldots, r_n)$, note that we have $t_{r_i} \leq t_{r_{i+1}}$ for $1 \leq i < n$ because $t_{r_i} - \tilde{t}_{r_i} = a$ is fixed.

▶ **Observation 4.** m-PGA only accepts requests without empty movement because the release time of a request is too late for a server to be able to serve it with empty movement in kSmL-U with $a < t$. Therefore, each m-PGA server accepts requests with alternating pick-up location, starting with a request with pick-up location 0.

▶ **Observation 5.** OPT only accepts requests without empty movement because the release time of a request is too late for a server to be able to serve it with empty movement in kSmL-U with $a < t$. Therefore, each OPT server accepts requests starting with a request with pick-up location 0, and any two consecutive requests accepted by a server of OPT have the following property: the drop-off location of the first request is the pick-up location of the second request.

For simplification of the analysis, we suppose that for each d, $1 \leq d \leq m$, OPT has k separate servers for serving requests for travel between location 0 and location d, and those k servers only serve such requests. This simplification does not decrease the profit gained by OPT. In this way we can analyse the requests for travel between location 0 and location d for different d independently. In the following, we focus on an arbitrary value of d and assume that S_d contains $\lfloor k/m \rfloor$ servers.

The analysis of the algorithm is divided into two parts. First, we reassign the requests in $R'(0,d)$, $R'(d,0)$, $R^*(0,d)$ and $R^*(d,0)$ by repeated application of two reassignment rules, and then we show that the profit accrued by the algorithm is within a certain factor of the profit accrued by OPT.

Suppose OPT accepts k^*_0 requests that start at location 0 and drop off at location d, i.e., $R^*(0,d) = \{r^{*0}_1, r^{*0}_2, \ldots, r^{*0}_{k^*_0}\}$, and OPT accepts k^*_1 requests that start at location d and drop off at location 0, i.e., $R^*(d,0) = \{r^{*d}_1, r^{*d}_2, \ldots, r^{*d}_{k^*_1}\}$. The subsets $R'(0,d) = \{r'^0_1, r'^0_2, \ldots, r'^0_{k'_0}\}$ and $R'(d,0) = \{r'^d_1, r'^d_2, \ldots, r'^d_{k'_1}\}$ of R' are defined analogously.

Reassignment Rule 1. Consider the case that requests r^{*0}_o and r^{*d}_o are both assigned to the same server for $o < i$ and r^{*0}_i and r^{*d}_i are assigned to different servers. Suppose r^{*0}_i is assigned to s^*_j and r^{*d}_i is assigned to s^*_l where $l \neq j$. We reassign r^{*d}_i to server s^*_j,

reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to s_j^* (denote the set of these requests by $\Re_j$) to server s_l^*, and reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to s_l^* (denote them by $\Re_l$) to server s_j^*.

As each server accepts requests with alternating pick-up location, starting with a request with pick-up location 0, we have $\dot{t}_{r_i'^0} \leq t_{r_i'^d}$ (for all $i \leq k_1'$) and $\dot{t}_{r_i^{*0}} \leq t_{r_i^{*d}}$ (for all $i \leq k_1^*$). Thus, for $i \leq k_1^*$, r_i^{*0} and r_i^{*d} are not in conflict, and hence reassigning r_i^{*d} to server s_j^* is valid. Furthermore, any two consecutive requests in $\Re_l$ are not in conflict, so reassigning all requests of $\Re_l$ to server s_j^* is valid. Observe that server s_l^* is at location d at time $t_{r_i^{*d}}$. Because the first request in $\Re_j$, say, request x, has pick-up location d and starts after $t_{r_i^{*d}}$, reassigning request x to server s_l^* is valid. As any two consecutive requests in $\Re_j$ are not in conflict, reassigning all requests of $\Re_j$ to server s_l^* is valid. From this it follows that $R^*(0,d)$ and $R^*(d,0)$ are still a valid solution with the same profit after the reassignment.

Reassignment Rule 2. Consider the case that requests r_o^{*0} and r_o^{*d} are both assigned to the server $s_{o \bmod k}$ for $o < i$ and r_i^{*0} and r_i^{*d} are not assigned to the server $s_{i \bmod k}$ (the case where r_i^{*d} does not exist can be handled similarly). Suppose r_i^{*0} and r_i^{*d} are assigned to s_j^*, $j \neq i \bmod k$. We reassign r_i^{*0} and r_i^{*1} to server $s_{i \bmod k}$, reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to s_j^* (denote the set of these requests by $\Re_j$) to server $s_{i \bmod k}^*$, and reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to $s_{i \bmod k}^*$ (denote them by $\Re_{i \bmod k}$) to server s_j^*.

Since the requests r_o^{*0} and r_o^{*d} are both assigned to the server $s_{o \bmod k}$ for $o < i$, the last request r_l accepted by $s_{i \bmod k}^*$ whose pick-up time is earlier than $t_{r_i^{*0}}$ ends not later than the last request accepted by s_j^* whose pick-up time is earlier than $t_{r_i^{*0}}$ if $j \neq i \bmod k$. Reassigning all requests of $\Re_j$ to server $s_{i \bmod k}^*$ is valid. Because the first request in $\Re_{i \bmod k}$ accepted by $s_{i \bmod k}^*$ starts later than $t_{r_i^{*0}}$ and starts at location $p_{r_i^{*0}}$, and any two consecutive requests in $\Re_{i \bmod k}$ are not in conflict, reassigning all requests of $\Re_{i \bmod k}$ to server s_j^* is valid. From this it follows that $R^*(0,d)$ and $R^*(d,0)$ are still a valid solution with the same profit after the reassignment.

Similarly, we reassign the requests in $R'(0,d)$ and $R'(d,0)$ based on the above process until both requests $r_i'^0$ and $r_i'^d$ are assigned to $s'_{i \bmod \lfloor k/m \rfloor}$ for $i \leq k_1'$. Note that this reassignment does not affect the validity of $R'(0,d)$ and $R'(d,0)$, and $P_{R'(0,d)}$ and $P_{R'(d,0)}$ do not change.

▶ **Theorem 6.** *m-PGA is $\frac{k}{\lfloor k/m \rfloor}$-competitive for kSmL-U if $0 \leq a < t$.*

Proof. We bound the competitive ratio of m-PGA by analyzing the requests between 0 and d for each d independently. As $\bigcup_{d=1}^{m} R'(0,d) \cup \bigcup_{d=1}^{m} R'(d,0) = R'$ and $\bigcup_{d=1}^{m} R^*(0,d) \cup \bigcup_{d=1}^{m} R^*(d,0) = R^*$, it is clear that, for any $\alpha \geq 1$, $P_{R^*}/P_{R'} \leq \alpha$ follows if we can show that $P_{R^*(0,d)}/P_{R'(0,d)} \leq \alpha$ and $P_{R^*(d,0)}/P_{R'(d,0)} \leq \alpha$ for all d, $1 \leq d \leq m$. Consider an arbitrary value of d from now on.

The proof uses similar ideas to the one used for the case $a < t$ of car-sharing between two locations (Theorem 2 in [9] and Theorem 2 in [10]), but there one can easily show that R^* can be transformed into R' without reducing its profit, whereas we encounter the additional difficulty that m-PGA has only $\lfloor k/m \rfloor$ servers while OPT has k servers.

Let $\bar{R}^*(0,d)$ (resp. $\bar{R}^*(d,0)$) be the subset of $R^*(0,d)$ (resp. $R^*(d,0)$) that contains the requests from the $(ik+1)^{th}$ to the $(ik+\lfloor k/m \rfloor)^{th}$, for all $0 \leq i \leq \frac{|R^*(0,d)|}{k}$ (resp. $0 \leq i \leq \frac{|R^*(d,0)|}{k}$). In other words, $\bar{R}^*(0,d)$ and $\bar{R}^*(d,0)$ contain the requests that are accepted by the first $\lfloor k/m \rfloor$ servers of OPT. Suppose the first $\lfloor k/m \rfloor$ servers of OPT accept k_0^* requests

that start at location 0 and drop off at location d, i.e., $\bar{R}^*(0,d) = \{r_1^{*0}, r_2^{*0}, \ldots, r_{k_0^*}^{*0}\}$, and the first $\lfloor k/m \rfloor$ servers of OPT accept k_1^* requests that start at location d and drop off at location 0, i.e., $\bar{R}^*(d,0) = \{r_1^{*d}, r_2^{*d}, \ldots, r_{k_1^*}^{*d}\}$. Suppose m-PGA accepts k_0 requests that start at location 0 and drop off at location d, i.e., $R'(0,d) = \{r_1'^0, r_2'^0, \ldots, r_{k_0}'^0\}$, and m-PGA accepts k_1 requests that start at location d and drop off at location 0, i.e., $R'(d,0) = \{r_1'^d, r_2'^d, \ldots, r_{k_1}'^d\}$. We claim that $\bar{R}^*(0,d)$ (resp. $\bar{R}^*(d,0)$) can be transformed into $R'(0,d)$ (resp. $R'(d,0)$) without reducing its profit, thus showing that $P_{\bar{R}^*(0,d)} \leq P_{R'(0,d)}$ (resp. $P_{\bar{R}^*(d,0)} \leq P_{R'(d,0)}$), and hence $P_{R^*(0,d)} \leq \frac{k}{\lfloor k/m \rfloor} P_{R'(0,d)}$ (resp. $P_{R^*(d,0)} \leq \frac{k}{\lfloor k/m \rfloor} P_{R'(d,0)}$).

By Observation 5, when $\bar{R}^*(d,0)$ consists of w requests, $\bar{R}^*(0,d)$ consists of at least w requests and of at most $w + \lfloor k/m \rfloor$ requests, i.e., $k_1^* \leq k_0^* \leq k_1^* + \lfloor k/m \rfloor$.

As m-PGA accepts the request r_γ which is the first acceptable request that starts at location 0 and the request r_δ which is the first acceptable request that starts at location d (r_δ is the first request in R that starts at location d and starts after $\dot{t}_{r_\gamma}$), it is clear that $t_{r_1'^0} \leq t_{r_1^{*0}}$ and $t_{r_1'^d} \leq t_{r_1^{*d}}$. If $r_1'^0 \neq r_1^{*0}$, we can replace r_1^{*0} by $r_1'^0$ in $\bar{R}^*(0,d)$, and if $r_1'^d \neq r_1^{*d}$, we can replace r_1^{*d} by $r_1'^d$ in $\bar{R}^*(d,0)$. Similarly, if $i \leq \lfloor k/m \rfloor$, request $r_i'^0$ (resp. $r_i'^d$) starts earlier than request r_i^{*0} (resp. r_i^{*d}). Otherwise, server s_i' accepts request r_i^{*0} and r_i^{*d} instead of $r_i'^0$ and $r_i'^d$. If $r_i'^0 \neq r_i^{*0}$, we can replace r_i^{*0} by $r_i'^0$ in $\bar{R}^*(0,d)$, and if $r_1'^d \neq r_i^{*d}$, we can replace r_i^{*d} by $r_i'^d$ in $\bar{R}^*(d,0)$.

Now assume, that the first i ($i \geq \lfloor k/m \rfloor$) requests in $\bar{R}^*(0,d)$ are identical to the first i requests in $R'(0,d)$, and the first i requests in $\bar{R}^*(d,0)$ are identical to the first i requests in $R'(d,0)$ where $1 \leq i \leq k_1^*$. Note that server $s'_{(i+1) \bmod \lfloor k/m \rfloor}$ and $s^*_{i+1 \bmod \lfloor k/m \rfloor}$ are at location 0 at time $\dot{t}_{r'_{(i+1)-\lfloor k/m \rfloor}}$. If there are two requests r_{i+1}^{*0} and r_{i+1}^{*d} accepted by server $s^*_{(i+1) \bmod \lfloor k/m \rfloor}$, there must also be two requests $r_{i+1}'^0$ and $r_{i+1}'^d$ accepted by $s'_{(i+1) \bmod \lfloor k/m \rfloor}$ and request $r_{i+1}'^0$ (resp. $r_{i+1}'^d$) starts earlier than request r_{i+1}^{*0} (resp. r_{i+1}^{*d}). If $r_{i+1}'^0 \neq r_{i+1}^{*0}$, we can replace r_{i+1}^{*0} by $r_{i+1}'^0$ in $\bar{R}^*(0,d)$, and if $r_1'^d \neq r_{i+1}^{*d}$, we can replace r_{i+1}^{*d} by $r_{i+1}'^d$ in $\bar{R}^*(d,0)$. If there are no such requests r_{i+1}^{*0} and r_{i+1}^{*d} accepted by server $s^*_{(i+1) \bmod \lfloor k/m \rfloor}$, then $i+1 > k_1^*$, and hence it follows that $\bar{R}^*(d,0)$ is identical to $R'(d,0)$ (or $R'(d,0)$ even contains additional requests).

If $k_0^* = k_1^*$, the claim thus follows. If $k_0^* \neq k_1^*$ ($k_0^* - k_1^* = \tau$ where $1 \leq \tau \leq \lfloor k/m \rfloor$), then $\bar{R}^*(d,0)$ is already identical to $R'(d,0)$, and the first k_1^* requests of $\bar{R}^*(0,d)$ are already identical to the first k_1^* requests of $R'(0,d)$ by the argument above. Observe that server s_j' and s_j^*, $1 \leq j \leq \lfloor k/m \rfloor$, are at location 0 at time $\dot{t}_{r'_{k_1^*+j-\lfloor k/m \rfloor}}$. If there is a request $r_{k_1^*+o}^{*0}$ ($1 \leq o \leq \tau$) accepted by server s_j^*, there must also be a request $r_{k_1^*+o}'^0$ accepted by server s_j' and it starts no later than request $r_{k_1^*+o}^{*0}$. If $r_{k_1^*+o}'^0 \neq r_{k_1^*+o}^{*0}$, we can replace $r_{k_1^*+o}^{*0}$ by $r_{k_1^*+o}'^0$ in $\bar{R}^*(0,d)$ making $\bar{R}^*(0,d)$ identical to $R'(0,d)$. If there is no request $r_{k_1^*+o}^{*0}$ accepted by server s_j^*, then $k_1^* + o > k_0^*$, and hence it follows that $\bar{R}^*(0,d)$ is identical to $R'(0,d)$ (or $R'(0,d)$ even contains additional requests). As $\bar{R}^*(d,0)$ is already identical to $R'(d,0)$, we have that $\bar{R}^*(0,d) \cup \bar{R}^*(d,0)$ is identical to $R'(0,d) \cup R'(d,0)$ (or $R'(0,d) \cup R'(d,0)$ even contains additional requests). ◀

3.2 Upper bound for $a \geq t$

We propose a Bi-Partition Greedy Algorithm (Bi-PGA) for the kSmL-U problem when $a \geq t$, shown in Algorithm 3. We assume that $k \geq 2$. The k servers are divided into two groups: a group S^c of $\lfloor k/2 \rfloor$ servers and a group S^n of $\lceil k/2 \rceil$ servers. The $\lfloor k/2 \rfloor$ servers in S^c serve requests that start at location 0, and the $\lceil k/2 \rceil$ servers in S^n serve requests that drop off at location 0.

Algorithm 3 Bi-Partition Greedy Algorithm (Bi-PGA).

Input: k servers, requests arrive over time.

Step: When request r_i arrives, if $p_{r_i} = 0$ and r_i is acceptable to a server in S^c, assign it to that server; otherwise, if $\dot{p}_{r_i} = 0$ and r_i is acceptable to a server in S^n, assign it to that server; otherwise, reject it.

(1) R'_j $(1 \le j \le k)$ is the list of requests accepted by server s_j before r_i is released.
(2) r_i is acceptable to a server s_j if and only if r_i is not in conflict with the requests in R'_j, i.e., $\forall r'_q \in R'_j$, $|t_{r_i} - t_{r'_q}| \ge 2t$.

▶ **Theorem 7.** *Bi-PGA is* $\frac{k}{\lfloor k/2 \rfloor}$*-competitive for kSmL-U if* $a \ge t$*. In particular, Bi-PGA is* 2*-competitive for kSmL-U if* $a \ge t$ *and* k *is even.*

Proof. For simplification of the analysis, we suppose that OPT can use k servers to serve requests that start at location 0 and another k servers to serve requests that drop off at location 0. This simplification does not decrease the profit gained by OPT. With this we can analyse the requests in $R'(0, X)$ and $R'(X, 0)$ independently. In the following analysis, we focus on the requests that start at location 0. Let $R'(0, X) = \{r'_1, \ldots, r'_{|R'(0,X)|}\}$ and $R^*(0, X) = \{r^*_1, \ldots, r^*_{|R^*(0,X)|}\}$, indexed in the order in which the requests are released.

Similar to the proof of Theorem 6, the analysis of the algorithm is divided into two parts. First, we reassign the requests in $R'(0, X)$ and $R^*(0, X)$ by repeated application of the following reassignment rule so that servers are assigned to the accepted requests in round-robin fashion. Then we show that the profit gained by the algorithm is within a certain factor of the profit accrued by OPT.

Reassignment Rule Assume that request r^*_o is assigned to server $s^*_{o \bmod k}$ for $o < i$ and r^*_i is not assigned to the server $s^*_{i \bmod k}$. Suppose r^*_i is assigned to s^*_j, $j \ne i \bmod k$. We reassign r^*_i to server $s_{i \bmod k}$, reassign all requests in $R^* \setminus \{r^*_1, r^*_2, \ldots, r^*_i\}$ that are assigned to s^*_j (denote the set of these requests by $\Re_j$) to server $s^*_{i \bmod k}$, and reassign all requests in $R^* \setminus \{r^*_1, r^*_2, \ldots, r^*_i\}$ that are assigned to $s^*_{i \bmod k}$ (denote them by $\Re_{i \bmod k}$) to server s^*_j.

Since request r^*_o is assigned to server $s^*_{o \bmod k}$ for $o < i$, the latest request r_l with pick-up time earlier than $t_{r^*_i}$ that is accepted by $s^*_{i \bmod k}$ ends no later than the latest request with pick-up time earlier than $t_{r^*_i}$ that is accepted by s^*_j if $j \ne i \bmod k$. Reassigning all requests of $\Re_j$ to server $s^*_{i \bmod k}$ is valid. Because the first request in $\Re_{i \bmod k}$ accepted by $s^*_{i \bmod k}$ starts no earlier than $t_{r^*_i}$ and any two consecutive requests in $\Re_{i \bmod k}$ are not in conflict, reassigning all requests of $\Re_{i \bmod k}$ to server s^*_j is valid as well. From this it follows that $R^*(0, X)$ is still a valid solution with the same profit after the reassignment. Similarly, we reassign the requests in $R'(0, X)$ based on the above process until request r'_i is assigned to $s'_{i \bmod \lfloor k/2 \rfloor}$ for $i \le |R'(0, X)|$. Note that this reassignment does not affect the validity of $R'(0, X)$, and $P_{R'(0,X)}$ does not change.

The remainder of the proof proceeds similarly as the proof of Theorem 6, but here we have that Bi-PGA has $\lfloor k/2 \rfloor$ servers while OPT has k servers and all requests accepted by OPT and Bi-PGA start at location 0.

Let $\bar{R}^*(0, X)$ be the subset of $R^*(0, X)$ that contains the requests from the $(ik + 1)^{th}$ to the $(ik + \lfloor k/2 \rfloor)^{th}$, for all i. In other words, $\bar{R}^*(0, X)$ contains the requests that are accepted by the first $\lfloor k/2 \rfloor$ servers of OPT. Suppose the first $\lfloor k/2 \rfloor$ servers of OPT accept k^*_0 requests that start at location 0, i.e., $\bar{R}^*(0, X) = \{r^*_1, r^*_2, \ldots, r^*_{k^*_0}\}$. Suppose Bi-PGA accepts k_0 requests that start at location 0, i.e., $R'(0, X) = \{r'_1, r'_2, \ldots, r'_{k_0}\}$. We claim that $\bar{R}^*(0, X)$ can be transformed into $R'(0, X)$ without reducing its profit, thus showing that $P_{\bar{R}^*(0,X)} \le P_{R'(0,X)}$, and hence $P_{R^*(0,X)} \le \frac{k}{\lfloor k/2 \rfloor} P_{R'(0,X)}$.

As Bi-PGA accepts the request r_γ which is the first acceptable request that starts at location 0, it is clear that $t_{r'_1} \le t_{r^*_1}$. If $r'_1 \neq r^*_1$, we can replace r^*_1 by r'_1 in $\bar{R}^*(0,X)$. Similarly, if $i \le \lfloor k/2 \rfloor$, request r'_i starts earlier than request r^*_i. Otherwise, server s'_i accepts request r^*_i instead of r'_i. If $r'_i \neq r^*_i$, we can replace r^*_i by r'_i in $\bar{R}^*(0,X)$.

Now assume, that the first i ($i \ge \lfloor k/2 \rfloor$) requests in $\bar{R}^*(0,X)$ are identical to the first i requests in $R'(0,X)$, where $1 \le i \le k^*_0$. Note that server $s'_{(i+1) \bmod \lfloor k/2 \rfloor}$ and $s^*_{i+1 \bmod \lfloor k/2 \rfloor}$ are at location $\dot{p}_{r'_{(i+1)-\lfloor k/2 \rfloor}}$ ($\dot{p}_{r'_{(i+1)-\lfloor k/2 \rfloor}} \neq 0$) at time $\dot{t}_{r'_{(i+1)-\lfloor k/2 \rfloor}}$. If there is a request r^*_{i+1} accepted by server $s^*_{(i+1) \bmod \lfloor k/2 \rfloor}$, there must also be a request r'_{i+1} accepted by $s'_{(i+1) \bmod \lfloor k/2 \rfloor}$ and request r'_{i+1} starts no later than request r^*_{i+1}. If $r'_{i+1} \neq r^*_{i+1}$, we can replace r^*_{i+1} by r'_{i+1} in $\bar{R}^*(0,X)$. If there is no such request r^*_{i+1} accepted by server $s^*_{(i+1) \bmod \lfloor k/2 \rfloor}$, then $i+1 > k^*_0$, and hence it follows that $\bar{R}^*(0,X)$ is identical to $R'(0,X)$ (or $R'(0,X)$ even contains additional requests). ◀

4 kSmL-A: Arbitrary travel times

▶ **Theorem 8.** *For $a \ge 2Lt$ and an arbitrary number k of servers, no deterministic on-line algorithm for kSmL-A can achieve competitive ratio smaller than $\frac{1}{2} \ln L$.*

Proof. Consider a star with $m+1$ nodes and $d(0,v) = v$ for $1 \le v \le m$. Note that $L = m$ and hence $\ln L = \ln m$. The adversary presents requests in γ phases, where phase i, for $1 \le i \le \gamma$, consists of k identical requests. The requests are released based on the release rule for kSmL-A shown in Algorithm 4. All requests appear at the same time.

Algorithm 4 Release Rule for kSmL-A.

Initialization: The adversary presents the requests in phase 1: k copies of the request $(\nu \cdot t - a, \nu \cdot t, 1, 0)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \ge 0$.
$i = 1$;
While $i < m$ do
    Let k_i be the number of servers used in phase i.
    if $\sum_{j=1}^{i} (j \cdot k_j) \le \frac{2ki}{\ln m}$, then break;
    else $i = i + 1$ and the adversary releases the requests in phase i;
$\gamma = i$;
(1) Phase i ($1 < i \le m$) consists of k copies of the request $(\tilde{t}_{r_1}, t_{r_1}, i, 0)$.

We make four observations.

(a) The requests in any two different phases are in conflict.
(b) For all $i < \gamma$, $\sum_{j=1}^{i} (j \cdot k_j) > \frac{2ki}{\ln m}$;
(c) If $\gamma > 1$, $k_1 > \frac{2k}{\ln m}$;
(d) For all $i < \gamma$, $\sum_{j=1}^{i} k_j > \frac{2k}{\ln m} \cdot \sum_{j=1}^{i} \frac{1}{j}$. (This follows from (b) and (c).)

If $\gamma < m$, the adversary has stopped releasing requests because $\sum_{j=1}^{\gamma} (j \cdot k_j) \le \frac{2k\gamma}{\ln m}$. In this case, OPT accepts all requests in phase γ, and we have $P_{R^*} = \gamma k r$. Since $P_{R'} = r \cdot \sum_{j=1}^{\gamma} (j \cdot k_j) \le \frac{2kr\gamma}{\ln m}$, $P_{R^*}/P_{R'} \ge \frac{1}{2} \cdot \ln m$.

Now assume $\gamma = m$. Using (d) for $i = m-1$, we have $\sum_{j=1}^{m-1} k_j > \frac{2k}{\ln m} \cdot \sum_{j=1}^{m-1} \frac{1}{j} > \frac{2k}{\ln m} \cdot \ln m = 2k$, a contradiction because the algorithm has only k servers and by (a) no server can serve requests from different phases. Therefore, the case $\gamma = m$ cannot occur. ◀

The Classified Greedy Algorithm

We use a deterministic version of the "Classify and Randomly Select" paradigm, which has been widely used in on-line interval scheduling and many other problems [2, 8], to design a classified greedy algorithm (CGA). We partition the requests into classes based on their travel time, and we assign a number of servers to each class of the requests. Given k servers and a star of $m+1$ locations whose edge lengths satisfy $1 \leq d(0,i) \leq L$ for all $1 \leq i \leq m$, we use $\lambda = \lceil \log L \rceil$ groups of servers. We require that $k \geq \lambda$, and for ease of presentation we assume that k is an integer multiple of λ. Group j, $1 \leq j \leq \lceil \log L \rceil$, contains k/λ servers that only serve requests whose travel time is between $2^{j-1}t$ and $2^j t$ (we say that those requests are in class j).

The classified greedy algorithm (CGA) can now be stated in a simple way: When a request r_i arrives, let $j = \lceil \log d(p_{r_i}, \dot{p}_{r_i}) \rceil$ be the class of r_i (if $d(p_{r_i}, \dot{p}_{r_i}) = 1$, set $j = 1$). If r_i is acceptable to any server from group j, accept r_i with that server. Otherwise reject it.

To simplify the analysis, we suppose that for each j, $1 \leq j \leq \lceil \log L \rceil$, OPT can use k separate servers to serve the requests whose travel times are between $2^{j-1}t$ and $2^j t$. This simplification does not decrease the profit gained by OPT. In this way we can analyse the requests in different classes independently. In the following analysis, we focus on an arbitrary class j. For class j, let OPT_j be the requests of class j that are accepted by OPT, and let CGA_j be the requests of class j accepted by CGA. It is clear that $P_{R^*}/P_{R'} = O(\log L)$ follows if we can show that $\frac{|OPT_j|}{|CGA_j|} = O(\log L)$ for each j.

▶ **Lemma 9.** *For each j, $\frac{|OPT_j|}{|CGA_j|} = O(\log L)$.*

Proof. For the purpose of the analysis, partition the set of k servers of OPT_j into k/λ sets of size λ arbitrarily, where $\lambda = \lceil \log L \rceil$ as above. Each of these sets is assigned to a distinct server s'_i among the k/λ servers of CGA_j. For $1 \leq i \leq k/\lambda$, let A_i be the set of λ servers of OPT_j that is assigned to s'_i, and let $R'(i)$ denote the set of requests accepted by s'_i.

For each OPT_j server $s^*_e \in A_i$, let $R^*(e)$ be the set of requests accepted by s^*_e and $\bar{R}^*(e)$ be the set of requests accepted by s^*_e that are not accepted by CGA_j. Let $\bar{R}^*(A_i) = \bigcup_{s^*_e \in A_i} \bar{R}^*(e)$. We claim that $|\bar{R}^*(e)| \leq \alpha |R'(j)|$ for some constant α. If this claim holds, we get that $|OPT_j| \leq |CGA_j| + \sum_i |\bar{R}^*(A_i)| \leq |CGA_j| + \sum_i \lambda\alpha |R'(i)| = (1+\lambda\alpha)|CGA_j| = O(\lambda) \cdot |CGA_j|$, proving the lemma.

It remains to prove the claim. Consider any request r_h in $\bar{R}^*(e)$. As s'_i did not accept r_h, s'_i must have accepted another request r_c with start time in $(t_{r_h} - 3 \cdot 2^j t, t_{r_h}]$; otherwise, the $3 \cdot 2^j t$ time units would have been sufficient for s'_i to serve the previous request and make an empty move to the pick-up location of r_h. We charge r_h to r_c. In this way, every request in $\bar{R}^*(e)$ is charged to a request in $R'(i)$.

We bound the number of requests that can be charged to a single request r_c in $R'(i)$. By the above charging scheme, every request that was accepted by s^*_e and charges r_c has a start time in $[t_{r_c}, t_{r_c} + 3 \cdot 2^j t)$. As all requests in class j have travel time at least $2^{j-1}t$, the start times of consecutive requests accepted by s^*_e differ by at least $2^{j-1}t$. A half-open interval of length $3 \cdot 2^j t$ can therefore contain at most $\frac{3 \cdot 2^j t}{2^{j-1}t} = 6$ request start times, and hence r_c is charged by at most 6 requests from $\bar{R}^*(e)$. This establishes the claim, with $\alpha = 6$. ◀

▶ **Theorem 10.** *CGA is $O(\log L)$-competitive for kSmL-A if $a \geq 2Lt$ and the number of servers is at least $\lceil \log L \rceil$.*

5 Conclusion

We have studied an on-line problem with k servers and $m+1$ locations in a star network that is motivated by applications such as car sharing between an airport and hotels. In particular,

we have analyzed the effects that different constraints on the booking time of requests have on the competitive ratio that can be achieved. We have given matching lower and upper bounds on the competitive ratio. It would be interesting to extend our results to the case of other networks.

References

1 Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS '00)*, volume 1770 of *LNCS*, pages 639–650. Springer, 2000. doi:10.1007/3-540-46541-3_53.

2 Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive Non-Preemptive Call Control. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 312–320. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314510.

3 Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight Bounds for Online TSP on the Line. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 994–1005. SIAM, 2017. doi:10.1137/1.9781611974782.63.

4 Katerina Böhmová, Yann Disser, Matús Mihalák, and Rastislav Srámek. Scheduling Transfers of Resources over Time: Towards Car-Sharing with Flexible Drop-Offs. In *Proceedings of the 12th Latin American Symposium on Theoretical Informatics (LATIN'16)*, volume 9644 of *LNCS*, pages 220–234. Springer, 2016. doi:10.1007/978-3-662-49529-2_17.

5 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

6 Ananya Christman, William Forcier, and Aayam Poudel. From theory to practice: Maximizing revenues for on-line dial-a-ride. *J. Comb. Optim.*, 35(2):512–529, 2018. doi:10.1007/s10878-017-0188-z.

7 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On Minimizing the Maximum Flow Time in the Online Dial-a-Ride Problem. In *Proceedings of the 3rd International Workshop on Approximation and Online Algorithms (WAOA 2005)*, volume 3879 of *LNCS*, pages 258–269. Springer, 2006. doi:10.1007/11671411_20.

8 Richard J. Lipton and Andrew Tomkins. Online Interval Scheduling. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 302–311. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314506.

9 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing Between Two Locations: Online Scheduling with Flexible Advance Bookings. In *Proceedings of the 24th International Conference on Computing and Combinatorics (COCOON 2018)*, volume 10976 of *LNCS*, pages 242–254. Springer, 2018. doi:10.1007/978-3-319-94776-1_21.

10 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing between Two Locations: Online Scheduling with Two Servers. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *LIPIcs*, pages 50:1–50:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.50.

11 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Online Scheduling of Car-Sharing Requests between Two Locations with Many Cars and Flexible Advance Bookings. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *LIPIcs*, pages 64:1–64:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.64.

12 Fanglei Yi and Lei Tian. On the Online Dial-A-Ride Problem with Time-Windows. In *Proceedings of the 1st International Conference on Algorithmic Applications in Management (AAIM '05)*, volume 3521 of *LNCS*, pages 85–94. Springer, 2005. doi:10.1007/11496199_11.

Beyond Boolean Surjective VCSPs

Gregor Matl
Department of Informatics, Technical University of Munich, Germany
matlg@in.tum.de

Stanislav Živný
Department of Computer Science, University of Oxford, UK
standa.zivny@cs.ox.ac.uk

Abstract

Fulla, Uppman, and Živný [ACM ToCT'18] established a dichotomy theorem for Boolean surjective general-valued constraint satisfaction problems (VCSPs), i.e., VCSPs on two-element domains in which both labels have to be used in a solution. This result, in addition to identifying the complexity frontier, features the discovery of a new non-trivial tractable case (called EDS) that does not appear in the non-surjective setting.

In this work, we go beyond Boolean domains. As our main result, we introduce a generalisation of EDS to arbitrary finite domains called SEDS (similar to EDS) and establish a conditional complexity classification of SEDS VCSPs based on a reduction to smaller domains. This gives a complete classification of SEDS VCSPs on three-element domains. The basis of our tractability result is a natural generalisation of the Min-Cut problem, in which only solutions of certain size (given by a lower and upper bound) are permitted. We show that all near-optimal solutions to this problem can be enumerated in polynomial time, which might be of independent interest.

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

Keywords and phrases constraint satisfaction problems, valued constraint satisfaction, surjective constraint satisfaction, graph cuts

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.52

Related Version A full version of the paper is available at [19], `https://arxiv.org/abs/1901.07107`.

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 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.
Gregor Matl: Work done while at the University of Oxford.
Stanislav Živný: Supported by a Royal Society University Research Fellowship.

1 Introduction

Constraint satisfaction problems (CSPs) are fundamental computer science problems studied in artificial intelligence, logic (as model checking of the positive primitive fragment of first-order logic), graph theory (as homomorphisms between relational structures), and databases (as conjunctive queries) [13]. A vast generalisation of CSPs is that of general-valued CSPs (VCSPs) [21], see also [6]. Recent years have seen some remarkable progress on our understanding of the computational complexity of CSPs and VCSPs, as will be discussed later in related work. We start with a few definitions to state existing as well as our new results.

We consider regular, surjective and lower-bounded VCSPs on the extended rationals $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$. An *instance* $I = (V, D, \phi_I)$ of either of these problems is given by a finite set of variables $V = \{x_1, \ldots, x_n\}$, a finite set of labels D called the *domain*, and an *objective function* $\phi_I : D^n \to \overline{\mathbb{Q}}$. The objective function is of the form

$$\phi_I(x_1, \ldots, x_n) = \sum_{i=1}^{t} w_i \cdot \gamma_i(\boldsymbol{x}_i),$$

where $t \in \mathbb{N}$ and, for each $1 \leq i \leq t$, $\gamma_i : D^{\mathrm{ar}(\gamma_i)} \to \overline{\mathbb{Q}}$ is a *weighted relation* of arity $\mathrm{ar}(\gamma_i) \in \mathbb{N}$, $w_i \in \mathbb{Q}_{\geq 0}$ is a *weight* and $\boldsymbol{x}_i \in V^{\mathrm{ar}(\gamma_i)}$ is a tuple of variables from V called the *scope* of γ_i.

Regular, surjective and lower-bounded VCSPs differ only in their solution space, although this makes a big difference in complexity. If I is an instance of a regular VCSP, an *assignment* is a map $s : V \to D$ assigning a label from D to each variable. In the surjective setting, only a surjective map $s : V \to D$ is an assignment. For lower-bounded VCSPs, a lower bound $l : D \to \mathbb{N}_0$ is provided and an assignment is a map $s : V \to D$ such that $\left|s^{-1}(d)\right| \geq l(d)$ for every label $d \in D$. In other words, a lower bound $l(d)$ on the number of occurrences of each label $d \in D$ is imposed. The *value* of an assignment s is given by $\phi_I(s(x_1), \ldots, s(x_n))$. An assignment is called *feasible* if its value is finite, and is called *optimal* if it is of minimal value among all assignments for the instance. The objective is to obtain an optimal assignment.

While finding an optimal assignment is NP-hard in general, valued constraint languages impose a natural restriction on the types of instances that are allowed. A *valued constraint language*, or simply a *language*, is a possibly infinite set of weighted relations. In this paper, we only consider languages of bounded arity, that is languages admitting a fixed upper bound on the arity of all weighted relations contained in them. Weighted relations in any VCSP instance will be stored explicitly.

We denote the class of regular VCSP instances with objective functions using only weighted relations from a language Γ by $\mathrm{VCSP}(\Gamma)$. Similarly, $\mathrm{VCSP}_s(\Gamma)$ is the class of surjective VCSP instances with weighted relations from Γ and, for some lower bound $l : D \to \mathbb{N}_0$, $\mathrm{VCSP}_l(\Gamma)$ is the class of lower-bounded VCSP instances over Γ with bound l.

A language Γ is *globally tractable* if there is a polynomial-time algorithm for solving each instance of $\mathrm{VCSP}(\Gamma)$, or *globally intractable* if $\mathrm{VCSP}(\Gamma)$ is NP-hard. Analogously, Γ is *globally s-tractable* if there is a polynomial-time algorithm for $\mathrm{VCSP}_s(\Gamma)$, or *globally s-intractable* if $\mathrm{VCSP}_s(\Gamma)$ is NP-hard. And Γ is *globally ℓ-tractable* if $\mathrm{VCSP}_l(\Gamma)$ is solvable in polynomial time for every fixed lower bound $l : D \to \mathbb{N}_0$, or *globally ℓ-intractable* if $\mathrm{VCSP}_l(\Gamma)$ is NP-hard for at least one fixed lower bound $l : D \to \mathbb{N}_0$. Thus, global ℓ-tractability implies global s-tractability, and global s-intractability implies global ℓ-intractability.

The following examples show how well-studied variants of the Min-Cut problem can be modelled in the VCSP frameworks we have defined.

▶ **Example 1** (r-Terminal Min-Cut)**.** Given a graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{Q}_{\geq 0}$ and designated terminal vertices $s_1, \ldots, s_r \in V$, the r-Terminal Min-Cut problem asks to partition V into subsets $X_1, \ldots, X_r$ such that $s_d \in X_d$ for all $d \in [r] := \{1, \ldots, r\}$, while the accumulated weight of all edges going between distinct sets X_i and X_j is minimised. For $r = 2$, this problem is also known as the (s,t)-Min-Cut problem.

We show how this problem can be represented as a regular VCSP. Let $\gamma_{r\text{-cut}}$ denote the binary weighted relation defined for $x, y \in [r]$ by $\gamma_{r\text{-cut}}(x, y) = 0$ if $x = y$ and $\gamma_{r\text{-cut}}(x, y) = 1$ otherwise. Furthermore, for each label $d \in [r]$, let ρ_d denote the constant relation given by $\rho_d(d) = 0$ and $\rho_d(x) = \infty$ for $d \neq x \in [r]$. Let $\Gamma_{r\text{-cut}} = \{\gamma_{r\text{-cut}}, \rho_1, \ldots, \rho_r\}$.

Finding an optimal r-terminal cut is equivalent to solving the VCSP $(\Gamma_{r\text{-cut}})$ instance $I = (V, [r], \phi)$ with objective function

$$\phi(x_1, \ldots, x_n) = \rho_1(s_1) + \cdots + \rho_r(s_r) + \sum_{(u,v)\in E} w(u,v) \cdot \gamma_{r\text{-cut}}(u,v).$$

To see this, observe that there is a correspondence between feasible assignments $s : V \to [r]$ and r-terminal cuts $X_1, \ldots, X_r$ by setting $X_d = \{v \in V : s(v) = d\}$, with the objective value remaining equal. Hence, an optimal assignment induces an optimal cut.

The r-Terminal Min-Cut problem can be solved in polynomial time if $r = 2$, but it is NP-hard for any $r \geq 3$ [8]. Since every VCSP $(\Gamma_{r\text{-cut}})$ instance can also be reduced to an instance of the r-Terminal Min-Cut problem, the language $\Gamma_{r\text{-cut}}$ is globally tractable if $r = 2$ and globally intractable for $r \geq 3$. ♢

▶ **Example 2** (r-Way Min-Cut)**.** Without setting out any terminals, the r-Way Min-Cut problem asks to partition V into non-empty subsets $X_1, \ldots, X_r$ such that weight of the induced cut is minimised. Finding an optimal r-way min-cut is equivalent to solving the VCSP$_s$ $(\{\gamma_{r\text{-cut}}\})$ instance $I = (V, [r], \phi)$ with objective function

$$\phi(x_1, \ldots, x_r) = \sum_{(u,v)\in E} w(u,v) \cdot \gamma_{r\text{-cut}}(u,v).$$

The r-Way Min-Cut problem can be solved in polynomial time for every fixed integer r [11]. Since every VCSP$_s$ $(\{\gamma_{r\text{-cut}}\})$ instance can be reduced to a r-Way Min-Cut problem as well, the language $\{\gamma_{r\text{-cut}}\}$ is globally s-tractable. ♢

For a fixed $l : D \to V$, VCSP$_l$ $(\{\gamma_{r\text{-cut}}\})$ allows to model a generalisation of the r-Way Min-Cut problem where a partition $X_1, \ldots, X_r$ of V minimising the induced cut is sought under the condition that $|X_d| \geq l(d)$ for every $d \in D$. As far as we know, the complexity of both VCSP$_l$ $(\{\gamma_{r\text{-cut}}\})$ and the lower-bounded r-Way Min-Cut problem is unknown.

Related Work

Early results on CSPs include the fundamental results of Schaefer on Boolean CSPs [20] and of Hell and Nešetřil on graph CSPs [12]. The computational complexity of CSPs has drawn a lot of attention following the seminal paper of Feder and Vardi [9]. Using the algebraic approach [15, 3], the complexity of CSPs on finite domains was resolved in two independent papers by Bulatov [4] and Zhuk [24]. The computational complexity of the problem of minimising the number of unsatisfied constraints (and more generally rational-valued weighted relations) was obtained by Thapper and Živný in [23]. Finally, the computational complexity of general-valued CSPs on finite domains was obtained by the work of Kozik and Ochremiak [18] and Kolmogorov, Krokhin, and Rolínek [16].

In addition to constraints that apply locally to the variables specified as arguments, forms of VCSPs have been considered where global conditions are imposed. Among those are CSPs with global cardinality constraints, or CCSPs, where it is specified how often exactly each label has to occur in an assignment. A dichotomy theorem for CCSPs on finite domains was established by Bulatov and Marx [5].

Surjective VCSPs, which can be seen as imposing a global condition as well, have been studied by Fulla, Uppman, and Živný [10], following earlier results on CSPs by Creignou and Hébrard [7] and Bodirsky, Kára, and Martin [1]. Unfortunately, the algebraic approach that has proved pivotal in the understanding of the computational complexity of regular CSPs and VCSPs is not applicable in the surjective setting.

The following two facts are easy to show (see, e.g, [10]): (i) intractable languages are also s-intractable; (ii) a tractable language Γ is also s-tractable *if* Γ includes all constant relations. Consequently, new s-tractable languages can only occur (if at all) as subsets of tractable languages that do not contain all constant relations. [10] identified the first example of such languages. In particular, [10] identified languages on the Boolean domain that are essentially a downset, or EDS, as a new class of efficiently solvable problems and, in doing so, provided a classification of surjective VCSPs on the Boolean domain.

The tractability result of EDS languages is based on the Generalised Min-Cut (GMC) problem for graphs, also introduced in [10]. In a GMC instance, the goal is to find a non-trivial subset of the vertices such that the weight of the induced cut and a superadditive set function are minimised simultaneously. [10] showed how the objective function of surjective VCSPs that are EDS can be approximated by an instance of the GMC problem. In addition, they provided a polynomial-time algorithm to enumerate all solutions to the GMC problem that are optimal up to a constant factor, which in combination results in an efficient algorithm for surjective VCSPs that are EDS.

Contributions

This paper extends the class EDS to arbitrary finite domains. We introduce a class SIM of languages that exhibit properties similar to Boolean languages. Based on this class, we define the class SEDS as a natural extension of EDS and classify languages from this extension based on two criteria. Firstly, we give a subclass SDS of SEDS that guarantees global ℓ-tractability without additional requirements. Secondly, we prove that the complexity of lower-bounded VCSPs over any remaining SEDS languages is equivalent to the complexity over a particular language on a smaller domain, which can be constructed by including all possible ways to assign a certain label (formally defined in Section 3). This is illustrated in Figure 1 (left).

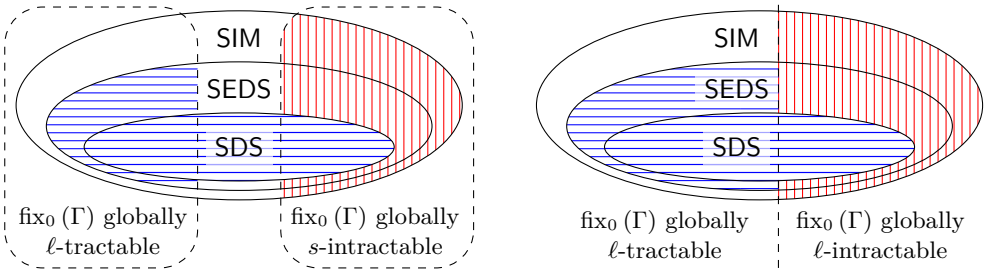


Figure 1 Classification of SEDS languages on arbitrary finite domains (left) and on three-element domains (right). A language Γ is globally ℓ-tractable when marked by horizontal (blue) lines and globally s-intractable when marked by vertical (red) lines, depending on the language fix$_0$ (Γ) on a smaller domain. (Recall that global s-intractability implies global ℓ-intractability.) In case of three-element domains, the Boolean language fix$_0$ (Γ) is either globally ℓ-tractable or globally ℓ-intractable, while this is not known for larger domains.

One implication of our results is a dichotomy theorem for lower-bounded VCSPs on the Boolean domain; every Boolean language is either globally ℓ-tractable or globally ℓ-intractable. Although lower-bounded VCSPs are more general than surjective VCSPs, this classification coincides with the dichotomy theorem for surjective VCSPs given by [10]. (Details are given in the full version of this paper [19].)

In addition, combining our reduction of SEDS languages to a smaller domain and the dichotomy theorem for the Boolean domain leads to a classification of all SEDS languages on three-element domains with respect to ℓ-tractability, which is featured on the right-hand side of Figure 1.

The foundation of our results is an extension of the Generalised Min-Cut problem that might be of independent interest. Given integers $p, q \in \mathbb{N}_0$, a graph with non-negative edge weights and a superadditive set function defined on its vertices, the goal in the Bounded Generalised Min-Cut problem is, just like in the GMC problem, to find a subset of the vertices such that the sum of the induced cut and the superadditive set function evaluated on it are minimal among all possible solutions. The solution space, however, is restricted to subsets containing at least q and at most all but p vertices.

If an optimal solution has value 0, there can be exponentially many optimal solution, e.g. when there are no edges and the superadditive function always evaluates to 0. Our main algorithmic result is that, for all other instances and any constant bounds $p, q \in \mathbb{N}_0$, all solutions that are optimal up to a constant factor can be enumerated in polynomial time (and thus, in particular, there are only polynomially many of them).

2 The Bounded Generalised Min-Cut Problem

We begin by presenting our algorithm for the Bounded Generalised Min-Cut problem. The problem is based on the notion of superadditive set functions, which we define first.

▶ **Definition 3.** *A set function on a finite set V is a function $f : 2^V \to \overline{\mathbb{Q}}$ defined on subsets of V; it is normalised if it satisfies $f(\emptyset) = 0$ and $f(X) \geq 0$ for all $X \subseteq V$.*

A set function f on V is increasing if it is normalised and $f(X) \leq f(Y)$ for all $X \subseteq Y \subseteq V$. It is superadditive if it is normalised and, for all disjoint $X, Y \subseteq V$, it holds that

$$f(X) + f(Y) \leq f(X \cup Y). \tag{SUP}$$

Since equation (SUP) implies that $f(X) \leq f(X) + f(Y \backslash X) \leq f(Y)$ for all $X \subseteq Y \subseteq V$, every superadditive set function must also be increasing.

▶ **Definition 4.** *For $q, p \in \mathbb{N}_0$, the Bounded Generalised Min-Cut problem with lower bound q an upper bound p is denoted by GMC_q^p.*

A GMC_q^p instance h is given by an undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ and an oracle defining a superadditive set function f on V. For $X \subseteq V$, let $w(X) = \sum_{|\{u,v\} \cap X| = 1} w(\{u, v\})$ denote the weight of the cut induced by X.

A solution for instance h is any set $X \subseteq V$ such that $|X| \geq q$ and $|X| \leq |V| - p$. The objective is to minimise the value $h(X) = f(X) + w(X)$. A solution X is optimal if the value $h(X)$ is minimal among all solutions for this instance. We denote the value of an optimal solution by λ. For any $\alpha \geq 1$, a solution X is α-optimal if $h(X) \leq \alpha\lambda$.

The Generalised Min-Cut problem, simply denoted by GMC, is the Bounded Generalised Min-Cut problem with lower and upper bound 1. All α-optimal solutions of a GMC instance can be enumerated in polynomial time according to [10, Theorem 5.11], which we restate here.

▶ **Theorem 5.** *For any instance h of the GMC problem on n vertices with optimal value $0 < \lambda < \infty$ and any constant $\alpha \in \mathbb{N}$, the number of α-optimal solutions is at most $n^{20\alpha - 15}$. There is an algorithm that finds all of them in polynomial time.*

We will assume that all edges are positive-valued, as they can be ignored otherwise. To simplify the problem further, observe that it can be determined in polynomial time whether the optimal value of a GMC_q^p instance is $\lambda = 0$ or $\lambda = \infty$. If $\lambda = 0$, an optimal solution can

be found by checking all connected components, because a solution of value 0 cannot cut any edges and because the superadditive set function f is increasing. Moreover, in order to determine whether $\lambda = \infty$ it is sufficient to check all solutions of size q. When these solutions all have infinite value, each one must either contain an edge of infinite weight or the superadditive set function must evaluate to infinity. In either case, all supersets will have infinite value as well, implying $\lambda = \infty$.

Consequently, our goal is to provide a polynomial-time algorithm for enumerating near-optimal solutions in the case that the optimal value is both positive and finite. Before doing so, we give two auxiliary lemmas. The first one is based on [10, Lemma 5.6].

▶ **Lemma 6.** *For any $p, q \in \mathbb{N}_0$, any GMC_q^p instance h on a graph $G = (V, E)$ and any subset $V' \subseteq V$, there is a GMC_q^p instance h' on the induced subgraph $G[V']$ that preserves the objective value of all solutions $X \subseteq V'$. In particular, any α-optimal solution X of h such that $X \subseteq V'$ is α-optimal for h' as well.*

Proof. Edges with exactly one endpoint in V' need to be taken into account separately because they do not appear in the induced subgraph. We accomplish that by defining the new set function f' by

$$f'(X) = f(X) + \sum_{u \in X} \sum_{v \in V \setminus V'} w(u, v)$$

for all $X \subseteq V'$. By the construction, f' is superadditive, and the objective value $h'(X)$ for any solution $X \subseteq V'$ equals $h(X)$.

Note that the minimum objective value for h' is greater than or equal to the minimum objective value for h. Therefore, any solution $X \subseteq V'$ that is α-optimal for h is also α-optimal for h'. ◀

The next lemma, which is based on [10, Lemma 5.10], can be deduced from the superadditivity of f and the posimodularity of the cut function w.

▶ **Lemma 7.** *Let h be a GMC_q^p instance over vertices V with optimal value λ and let $X, Y \subseteq V$ such that $h(X) \leq \alpha\lambda$ and $w(Y) \leq \beta\lambda$ for some $\alpha \geq 1$ and $\beta \geq 0$. Then it holds*

$$h(X \setminus Y) + h(X \cap Y) < (\alpha + 2\beta)\lambda.$$

We now proceed with our main algorithmic result. We only sketch the proof here but full details are given in the full version [19].

▶ **Theorem 8.** *For some constant $q \geq 2$, let h be a GMC_q^1 instance on a graph $G = (V, E)$ of size $n = |V|$ with optimal value $0 < \lambda < \infty$. Let $Y \cup Z = V$ be a partition of V and let $Y_1 \cup \cdots \cup Y_k = Y$ for some $k \in \mathbb{N}_0$ be a partition of Y satisfying $0 < |Y_i| < q$ and $h(Y_i) \leq \frac{\lambda}{3q}$ for all $1 \leq i \leq k$.*

Then for every constant $\alpha \geq 1$, at most $\frac{|Z|}{n} \cdot n^{\tau(q,\alpha)}$ α-optimal solutions $X \subseteq V$ of h satisfy $|X \cap Y| < q$, where $\tau(q, \alpha) = 60q\alpha + 41q + 7$. These solutions can all be enumerated in polynomial time.

Note that with $Y = \emptyset$ and $Z = V$, this theorem states for any GMC_q^1 instance that the number of α-optimal solutions is bounded by $n^{\tau(q,\alpha)}$.

Proof sketch. We give a proof by induction over $n + \frac{|Z|}{n+1}$. For $n \leq q$ or $Z = \emptyset$, there are no solutions of the described form and hence, the statement holds.

Now, fix some $n > q$, some GMC_q^1 instance h on a graph $G = (V, E)$ of size n with optimum value $0 < \lambda < \infty$ and partitions $Y \cup Z = V$ and $Y_1 \cup \cdots \cup Y_k = Y$ as described. By the induction hypothesis, we can assume that the theorem holds for every graph of size $n' < n$ as well as for every partition $\tilde{Y} \cup \tilde{Z} = V$ of graph G satisfying $\left|\tilde{Z}\right| < |Z|$.

According to Lemma 6, there exists a GMC_q^1 instance h_Z on the induced subgraph $G[Z]$ that preserves the objective value of every solution $X' \subsetneq Z$ with respect to h. In the following, we treat h_Z as a GMC instance (i.e. with lower bound 1). Let λ_Z denote the optimal value of h_Z and let $Y_{k+1} \subsetneq Z$ be an optimal solution of h_Z, i.e. $h_Z(Y_{k+1}) = \lambda_Z$.

Consider any α-optimal solution $X \subseteq V$ of h satisfying $|X \cap Y| < q$. For some integer t, let $i_1, \ldots, i_t$ denote indices such that $X \cap Y = X \cap (Y_{i_1} \cup \cdots \cup Y_{i_t})$, i.e. such that X has vertices only in $Y_{i_1}, \ldots, Y_{i_t}$ and Z. Since $|X \cap Y| < q$, we require that $t < q$. Let $U = Y_{i_1} \cup \cdots \cup Y_{i_t}$. We distinguish two cases, which are illustrated in Figure 2.

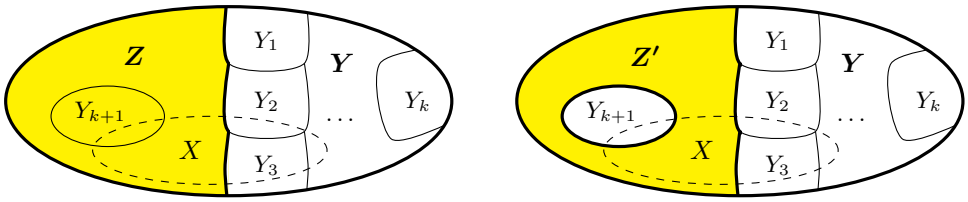


Figure 2 Given a partition $V = Y \cup Z$ with $Y = Y_1 \cup \cdots \cup Y_k$ of the vertices of a GMC_q^1 instance h, we want to find every solution X such that $h(X) \leq \alpha\lambda$ and $|X \cap Y| < q$. Consider the GMC instance h_Z on $G[Z]$ with optimal solution Y_{k+1}. If $h(Y_{k+1}) \geq \frac{\lambda}{3q}$, $X \cap Z$ must be a near-optimal solution of h (left, first case). Otherwise, we apply the induction hypothesis either on the partition $V = Z' \cup (Y_1 \cup \cdots \cup Y_{k+1})$ or on the subgraph $G[Z']$, where $Z' = Z \backslash Y_{k+1}$ (right, second case).

In the first case, we assume that $\lambda_Z \geq \frac{\lambda}{3q}$. From our assumption that $h(Y_i) \leq \frac{\lambda}{3q}$ for all $1 \leq i \leq k$, it can be deduced that $w(U) < \frac{\lambda}{3}$. By Lemma 7 with $\beta = \frac{1}{3}$, it must hold that

$$h(X \backslash U) + h(X \cap U) \leq \left(\alpha + \frac{2}{3}\right)\lambda.$$

Since $X \cap Z = X \backslash U$, our assumption $\lambda_Z \geq \frac{\lambda}{3q}$ then implies that $h(X \cap Z) \leq (3q\alpha + 2q)\lambda_Z$. Hence, if $X \cap Z \subsetneq Z$, then $X \cap Z$ is a $(3q\alpha + 2q)$-optimal solution of h_Z when treated as a GMC instance. The number of choices for $X \cap Z$ can then be bounded by Theorem 5.

At the same time, there are less than n^q ways to pick at most $q - 1$ vertices from Y and therefore less than n^q choices for $X \cap Y$. By pairing up all choices for $X \cap Z$ with those for $X \cap Y$, we can conclude that there are at most $\frac{1}{n} \cdot n^{\tau(q,\alpha)}$ choices for X in this case.

In the second case, we assume $\lambda_Z \leq \frac{\lambda}{3q}$. Note that $h_Z(Y_{k+1}) = \lambda_Z < \lambda$ implies $|Y_{k+1}| < q$. Let $Y' = Y \cup Y_{k+1}$, $Z' = Z \backslash Y_{k+1}$ and $U' = U \cup Y_{k+1}$.

If $|X \cap Y'| < q$, we can apply the induction hypothesis for instance h with the partitions $Y' \cup Z' = V$ and $Y' = Y_1 \cup \cdots \cup Y_k \cup Y_{k+1}$. Consequently, the number of such solutions is at most $\frac{|Z'|}{n} \cdot n^{\tau(\alpha)} \leq \frac{|Z|-1}{n} \cdot n^{\tau(\alpha)}$.

Therefore, we now assume that $|X \cap Y'| \geq q$ and need to show that there are at most $\frac{1}{n} \cdot n^{\tau(\alpha)}$ choices for X in this case. Given that $\lambda_Z \leq \frac{\lambda}{3q}$, we can deduce that $w(U') \leq \frac{\lambda}{3}$. Thus, Lemma 7 implies that

$$h(X \backslash U') + h(X \cap U') \leq \left(\alpha + \frac{2}{3}\right)\lambda.$$

Being a solution of h, the set $X \cap U' = X \cap Y'$ must have value $h(X \cap Q') \geq \lambda$. It therefore holds that

$$h(X \cap Z') = h(X \backslash U') \leq \left(\alpha + \frac{2}{3}\right)\lambda - h(X \cap Q') \leq \left(\alpha - \frac{1}{3}\right)\lambda.$$

Let $h_{Z'}$ denote the GMC_q^1 instance on the induced subgraph $G[Z']$ that preserves the value of h as detailed in Lemma 6. There are roughly n^q choices for X such that $|X \cap Z'| < q$ or $X = Z'$. Otherwise, the set $X \cap Z'$ must be an $\left(\alpha - \frac{1}{3}\right)$-optimal solution of $h_{Z'}$. By applying the induction hypothesis on $h_{Z'}$ with the trivial partition $\emptyset \cup Z' = Z'$, the number of $\left(\alpha - \frac{1}{3}\right)$-optimal solutions can be bounded by $n^{\tau\left(q,\alpha-\frac{1}{3}\right)}$.

Next, we limit the number of choices for $X \cap Y'$. Since X contains at most $q-1$ vertices from Y (less than n^q choices) and since Y_{k+1} contains at most $q-1$ vertices (at most 2^{q-1} choices), the number of possible choices for $X \cap Y'$ is limited by $n^q \cdot 2^{q-1} \leq n^{2q}$.

Pairing up each possible choice for $X \cap Z'$ with each choice for $X \cap Y'$ gives a total of at most $\frac{1}{n} \cdot n^{\tau(q,\alpha)}$ solutions, as required.

A polynomial-time algorithm to enumerate all such solutions follows immediately from these calculations. To see this, note that only the second case is defined recursively. Therefore, checking both the first and the second case does not increase the overall complexity of $n^{O(q+\alpha)}$. In particular, it is not necessary to know the value λ beforehand. ◀

▶ **Corollary 9.** *For any $p, q \in \mathbb{N}_0$ and $\alpha \geq 1$, where q and α are constants, and for any GMC_q^p instance h with optimal value $0 < \lambda < \infty$, all α-optimal solutions can be enumerated in polynomial time.*

Proof. Let $h = f + w$ be a GMC_q^p instance with $0 < \lambda < \infty$. First, we assume that $p \geq 1$ and $q \geq 2$. The superadditive set function

$$f'(X) = \begin{cases} \infty & \text{if } |X| > |V| - p \\ f(X) & \text{otherwise} \end{cases}$$

defines a GMC_q^1 instance $h' = f' + w$ where every solution $X \subseteq V$ of size $|X| > |V| - p$ is infeasible so that the set of feasible solutions and their values are identical for h and h'. Therefore, it is sufficient to enumerate all α-optimal solutions of h', which can be accomplished in polynomial time according to Theorem 8

If $p = 0$ or $q < 2$, there are up to $|V| + 2$ additional solutions that can all be checked in polynomial time. ◀

3 Extending EDS to Larger Domains

In this section, we formally introduce the classes SIM, SEDS and SDS. In order to simplify our notation, we will subsequently always consider the $(k+1)$-element domain $D = \{0, 1, \ldots, k\}$ for some integer k. Any other domain of size $k+1$ can simply be relabelled without affecting its properties. One label from the domain will play a special role; without loss of generality (due to relabellings), it will be 0.

3.1 k-Set Functions

It will be convenient to go back and forth between weighted relations and k-set functions, which is, subject to a minor technical assumption, always possible.

▶ **Definition 10.** *Let $k \in \mathbb{N}$ and let V be a finite set. A k-set function on V is a function $f : (k+1)^V \to \overline{\mathbb{Q}}$ defined on k-tuples of pairwise disjoint subsets of V. A k-set function f over V is normalised if it satisfies $f(\emptyset, \dots, \emptyset) = 0$ and $f(X_1, \dots, X_k) \geq 0$ for all disjoint $X_1, \dots, X_k \subseteq V$.*

Note that a 1-set function is simply a *set function* as defined in Section 2.

▶ **Definition 11.** *Let γ be an n-ary weighted relation on the $(k+1)$-element domain $D = \{0, 1, \dots, k\}$, and let f be the k-set function on $V = [n]$ that is defined for disjoint sets $X_1, \dots, X_k \subseteq V$ by $f(X_1, \dots, X_k) = \gamma(\boldsymbol{x})$, where the i-th coordinate of $\boldsymbol{x}$ is given by $x_i = d$ if $i \in X_d$ for some $0 \neq d \in D$ and $x_i = 0$ otherwise. Then γ corresponds to f.*

Furthermore, we say that γ corresponds under normalisation to a k-set function if $\gamma(\mathbf{0}^n) < \infty$ and $\gamma(\mathbf{0}^n) \leq \gamma(\boldsymbol{x})$ for all $\boldsymbol{x} \in D^n$. In this case, the k-set function corresponding under normalisation to γ is the normalised k-set function corresponding to $\gamma - \gamma(\mathbf{0}^n)$, i.e. the weighted relation with offset such that the assignment $\mathbf{0}^n$ evaluates to 0.

According to this definition, there is a unique k-set function corresponding to every weighted relation on the $(k+1)$-element domain, and vice versa. Furthermore, assuming that $\gamma(\mathbf{0}^n) < \infty$, a weighted relation γ corresponds under normalisation to a k-set function precisely if it admits multimorphism $\langle c_0 \rangle$ (the definition of which can be found in [6]).

▶ **Definition 12.** *Let f be a k-set function and g a set function on V. We say that g α-approximates f if, for all disjoint $X_1, \dots, X_k \subseteq V$, it holds that*

$$g(X_1 \cup \dots \cup X_k) \leq f(X_1, \dots, X_k) \leq \alpha \cdot g(X_1 \cup \dots \cup X_k).$$

3.2 Fixing a Label: Reduced Languages

Reducing a language to a smaller domain by fixing all possible occurrences of a certain label, as defined subsequently, will be a central tool in our classification.

▶ **Definition 13.** *Let f be a k-set function on V, let $0 \leq d \leq k$ be a label from the domain and let $U \subseteq V$. Then $\mathrm{fix}_{d=U}[f]$ is the $(k-1)$-set function defined for disjoint sets $X_1, \dots, X_{d-1}, X_{d+1}, \dots, X_k \subseteq V \backslash U$ by*

$$\mathrm{fix}_{d=U}[f](X_1, \dots, X_{d-1}, X_{d+1}, \dots, X_k) = f(X_1, \dots, X_{d-1}, U, X_{d+1}, \dots, X_k).$$

Let γ be the weighted relation on domain D corresponding to f. Then $\mathrm{fix}_{d=U}[\gamma]$ denotes the weighted relation of arity $|V \backslash U|$ on domain $D \backslash \{d\}$ corresponding to $\mathrm{fix}_{d=U}[f]$.

In other words, $\mathrm{fix}_{d=U}[\gamma]$ takes an assignment from the domain $D \backslash \{d\}$ to all variables except for those with index in U, and evaluates it through γ by assigning label d to the remaining variables. In Definition 14, we generalise this concept in order to express the language that is generated by fixing every possible assignment of a certain label.

▶ **Definition 14.** *Let Γ be a language on domain D and let $d \in D$. For any $\gamma \in \Gamma$, let $\mathrm{fix}_d(\gamma) = \{\mathrm{fix}_{d=U}[\gamma] : U \subseteq V\}$. We define the language $\mathrm{fix}_d(\Gamma)$ on domain $D \backslash \{d\}$ by $\mathrm{fix}_d(\Gamma) = \bigcup_{\gamma \in \Gamma} \mathrm{fix}_d(\gamma)$.*

3.3 Extending EDS to Larger Domains

The class EDS, or *essentially a downset*, has been introduced in [10] for the Boolean domain.

▶ **Definition 15.** *For any $\alpha \geq 1$, a normalised set function f on V is α-EDS if, for all $X, Y \subseteq V$, it holds that*

$$f(X \backslash Y) \leq \alpha \cdot (f(X) + f(Y)). \tag{EDS}$$

A weighted relation is α-EDS if it corresponds under normalisation to a set function that is α-EDS. Moreover, a language Γ is EDS if there is some $\alpha \geq 1$ such that every weighted relation $\gamma \in \Gamma$ is α-EDS.

Fulla et al. showed [10] that EDS languages are globally s-tractable. We improve upon this result by proving that such languages are in fact globally ℓ-tractable, and we extend the idea of being essentially a downset to larger domains through the classes SIM, SEDS and SDS.

Intuitively, a language is SIM, or *similar to a Boolean language*, if, for every weighted relation, the value of any two assignments that assign label 0 to precisely the same set of variables is equal up to a constant factor.

▶ **Definition 16.** *Let f be a normalised k-set function on set V. For any $\alpha \geq 1$, f is called α-SIM if, for all disjoint $X_1, \ldots, X_k \subseteq V$ and all disjoint $Y_1, \ldots, Y_k \subseteq V$ such that $X_1 \cup \cdots \cup X_k = Y_1 \cup \cdots \cup Y_k$, it holds that*

$$f(X_1, \ldots, X_k) \leq \alpha \cdot f(Y_1, \ldots, Y_k). \tag{SIM}$$

A weighted relation is α-SIM if it corresponds under normalisation to a k-set function that is α-SIM. Moreover, a language Γ is SIM if there is some $\alpha \geq 1$ such that every weighted relation $\gamma \in \Gamma$ is α-SIM.

Note that every normalised set function is 1-SIM. Hence, EDS is a subclass of SIM. Going beyond the Boolean domain, the class SEDS of languages *similar to EDS* arises as a natural generalisation of EDS.

▶ **Definition 17.** *For any $\alpha \geq 1$, a normalised k-set function f on V is α-SEDS if it is α-SIM and, for all disjoint $X_1, \ldots, X_k \subseteq V$ and all disjoint $Y_1, \ldots, Y_k \subseteq V$, it holds that*

$$f(X_1 \backslash Y_1, \ldots, X_k \backslash Y_k) \leq \alpha \cdot (f(X_1, \ldots, X_k) + f(Y_1, \ldots, Y_k)). \tag{SEDS}$$

A weighted relation is α-SEDS if it corresponds under normalisation to a k-set function that is α-SEDS. Moreover, a language Γ is SEDS if there is some $\alpha \geq 1$ such that every weighted relation $\gamma \in \Gamma$ is α-SEDS.

The class SDS, or *similar to a downset*, imposes a stricter requirement than SEDS. When any arguments of a weighted relation are changed to label 0, the value should decrease, stay equal or increase by at most a constant factor.

▶ **Definition 18.** *For any $\alpha \geq 1$, a normalised k-set function f on V is α-SDS if it is α-SIM and in addition, for all disjoint $X_1, \ldots, X_k, Y_1, \ldots, Y_k \subseteq V$, it holds that*

$$f(X_1, \ldots, X_k) \leq \alpha \cdot f(X_1 \cup Y_1, \ldots, X_k \cup Y_k). \tag{SDS}$$

A weighted relation is α-SDS if it corresponds under normalisation to a k-set function that is α-SDS, and a language Γ is SDS if there is some $\alpha \geq 1$ such that every weighted relation $\gamma \in \Gamma$ is α-SDS.

Note that SDS is a subclass of SEDS. To see this, consider any α-SDS k-set function f on V. Then it holds for all disjoint $X_1, \ldots, X_k \subseteq V$ and all disjoint $Y_1, \ldots, Y_k \subseteq V$ that

$$f(X_1 \backslash Y_1, \ldots, X_k \backslash Y_k) \leq \alpha \cdot f(X_1, \ldots, X_k) \leq \alpha \cdot (f(X_1, \ldots, X_k) + f(Y_1, \ldots, Y_k)),$$

proving that f is α-SEDS.

4 Classifying SEDS and SDS Languages

In this section, we first show that a SEDS language Γ is globally ℓ-tractable if it is SDS or if the reduced language $\mathrm{fix}_0(\Gamma)$ is globally ℓ-tractable. Afterwards, we prove global s-intractability of the remaining SEDS languages conditioned on global s-intractability of $\mathrm{fix}_0(\Gamma)$.

We begin by restating [10, Theorem 5.17] concerning EDS languages and then devise similar approximations for SEDS and SDS languages.

▶ **Theorem 19.** *For any α-EDS set function f on V, there exists a* GMC *instance h that $\alpha^{n+2}\left(n^3+2n\right)$-approximates f, where $n=|V|$.*

▶ **Lemma 20.** *For any α-SEDS k-set function f on V, there exists an α-EDS set function g that α^2-approximates f.*

Proof. We define the set function g on V by $g(X)=\frac{1}{\alpha}f(X,\emptyset,\dots,\emptyset)$. Observe that, since f is normalised, it holds $g(\emptyset)=f(\emptyset,\dots,\emptyset)=0$ and $g(X)=\frac{1}{\alpha}f(X,\emptyset,\dots,\emptyset)\geq 0$ for every $X\subseteq V$. Thus, g is normalised as well. In addition, for all $X,Y\subseteq V$, it holds that

$$\alpha\cdot(g(X)+g(Y))=f(X,\emptyset,\dots,\emptyset)+f(Y,\emptyset,\dots,\emptyset)\geq\frac{1}{\alpha}\cdot f(X\backslash Y,\emptyset,\dots,\emptyset)=g(X\backslash Y),$$

where the second step uses equation (SEDS). Hence, g is α-EDS.

It remains to show that g α^2-approximates f. For this purpose, consider any disjoint $X_1,\dots,X_k\subseteq V$ and let $X=\bigcup_{i=1}^{k}X_i$ denote their union. Since f is α-SIM, it holds that

$$g(X)=\frac{1}{\alpha}f(X,\emptyset,\dots,\emptyset)\leq f(X_1,\dots,X_k)\leq\alpha\cdot f(X,\emptyset,\dots,\emptyset)=\alpha^2\cdot g(X).$$ ◀

By combining Lemma 20 and Theorem 19, we can deduce the following result.

▶ **Theorem 21.** *For any α-SEDS k-set function f on V, there exists a* GMC *instance h that $\alpha^{n+4}\left(n^3+2n\right)$-approximates f, where $n=|V|$.*

For SDS languages, we can provide an even tighter result.

▶ **Theorem 22.** *For any α-SDS k-set function f on V, there exists a superadditive set function g that $n\alpha^{n+1}$-approximates f, where $n=|V|$.*

Based on these approximations, we now show our main tractability theorem, which in places closely follows the proof of [10, Theorem 5.18].

▶ **Theorem 23.** *Let Γ be a SEDS language. Then Γ is globally ℓ-tractable if it is SDS or if the reduced language $\mathrm{fix}_0(\Gamma)$ is globally ℓ-tractable.*

Proof. Let Γ be an SEDS language on domain D. Then every weighted relation $\gamma\in\Gamma$ corresponds under normalisation to a k-set function f_γ. Furthermore, weighted relations in Γ are of bounded arity. If Γ is SDS, Theorem 22 implies that for some $\alpha\in\mathbb{N}$, every such k-set function f_γ can be α-approximated by a superadditive set function h_γ. In the following, we treat h_γ as a GMC instance without any edge weights. If Γ is not SDS, we can still α-approximate every k-set function f_γ by a GMC instance h_γ according to Theorem 21, but there is no restriction on the edge weights.

Let $l:D\to\mathbb{N}_0$ be a fixed lower bound and consider any $\mathrm{VCSP}_l(\Gamma)$ instance I with objective function

$$\phi_I(x_1,\dots,x_n)=\sum_{i=1}^{t}w_i\cdot\gamma_i\left(\boldsymbol{x}^i\right).$$

Let f_I be the k-set function corresponding under normalisation to the objective function ϕ_I. We construct a GMC instance h that α-approximates f_I.

For $i \in [t]$, we relabel the vertices of h_{γ_i} to match the variables in the scope $\boldsymbol{x}^i$ of the i-th constraint (i.e., vertex j is relabelled to x^i_j) and identify vertices in case of repeated variables. As the constraint is weighted by a non-negative factor w_i, we also scale the weights of the edges of h_{γ_i} and the superadditive set function by w_i (note that non-negative scaling preserves superadditivity). Instance h is then obtained by adding up GMC instances h_{γ_i} for all $i \in [t]$. In the following, we treat h as a $\mathrm{GMC}^{l(0)}_{l^*}$ instance, where $l^* = \sum_{i=1}^{k} l(i)$. Note that if Γ is SDS, h has zero edge weights.

Let $X_0, \dots, X_k$ be a partition of $[n]$ such that $f_I(X_1, \dots, X_k)$ is minimal among all partitions satisfying $|X_d| \geq l(d)$ for all $d \in D$. In other words, $X_0, \dots, X_k$ corresponds to an optimal assignment for instance I. Let $X = \bigcup_{d=1}^{k} X_d$ denote all indices with non-zero labels. In addition, let $Y \subseteq [n]$ denote an optimal solution of the $\mathrm{GMC}^{l(0)}_{l^*}$ instance h and let $\lambda = h(Y)$.

Since $|Y| \geq l^*$, there must exist some partition $Y_1, \dots, Y_k$ of Y such that $|Y_d| \geq l(d)$ for all $1 \leq d \leq k$. Because h α-approximates f_I, it holds that

$$\lambda \leq h(X) \leq f_I(X_1, \dots, X_k) \leq f_I(Y_1, \dots, Y_k) \leq \alpha \cdot h(Y) = \alpha \cdot \lambda.$$

Hence, X is an α-optimal solution of h.

As discussed in Section 2, it can be determined in polynomial time whether $\lambda = 0$, $\lambda = \infty$ or $0 < \lambda < \infty$. Furthermore, if $\lambda = 0$, a solution Z such that $h(Z) = 0$ can be found. Because Z must have size $|Z| \geq l^*$ as a solution of $\mathrm{GMC}^{l(0)}_{l^*}$ instance h, we can select some partition $Z_1, \dots, Z_k$ of Z such that $|Z_d| \geq l(d)$ for all $1 \leq d \leq k$. Since h α-approximates f_I, it must hold $f_I(Z_1, \dots, Z_k) \leq \alpha \cdot h(Z) = 0$, meaning that $Z_1, \dots, Z_k$ represents an optimal assignment for instance I.

If $\lambda = \infty$, then obviously there are no feasible solutions.

Otherwise, it holds $0 < \lambda < \infty$. In this case, we distinguish whether Γ is SDS or $\mathrm{fix}_0(\Gamma)$ is globally ℓ-tractable.

First, we assume that Γ is SDS and hence, that h has zero edge weights. We claim that the size of X is bounded by a constant. For the sake of contradiction, assume that $|X| \geq (\alpha+1)\, l^*$. Then there are disjoint subsets $Z_1, Z_2, \dots, Z_{\alpha+1} \subseteq X$ such that $|Z_i| \geq l^*$ for all $1 \leq i \leq \alpha + 1$. Being a solution of h, every Z_i must have value at least $h(Z_i) \geq \lambda$. Based on the superadditivity of h, we arrive at the contradiction

$$(\alpha+1) \cdot \lambda \leq h(Z_1) + \dots + h(Z_{\alpha+1}) \leq h(X) \leq \alpha \cdot \lambda.$$

Thus, it must hold $|X| < (\alpha+1)\, l^*$. This leaves less than $O\left(n^{(\alpha+1)l^*}\right)$ possible choices for X, each of which admits at most $O\left(k^{(\alpha+1)l^*}\right)$ partitions of the form $X_1 \cup \dots \cup X_k = X$. By checking all of these, we can retrieve the sets $X_1, \dots, X_k$ in polynomial time.

Now, we assume that $\mathrm{fix}_0(\Gamma)$ is globally ℓ-tractable. According to Corollary 9, there are only polynomially many α-optimal solutions of h, all of which can be computed in polynomial time. X must be among those solutions. By repeating the following process for all of them, we can assume that X is known, and so is $X_0 = [n] \setminus X$.

Let $D^* = D \setminus \{0\}$ and let $l_{\restriction_{D^*}} : D^* \to \mathbb{N}$ denote the restriction of l to D^*. We can efficiently find a minimal assignment for the $\mathrm{VCSP}_{l_{\restriction_{D^*}}}(\mathrm{fix}_0(\Gamma))$ instance $I_X = (X, D^*, \phi_X)$ with objective function $\phi_X = \mathrm{fix}_{0=X_0}[\phi_I]$. The sets $X_1, \dots, X_k$ represent an assignment for I_X and, by assigning label 0 to the variables in X_0, every assignment for I_X induces an assignment for I with the same objective value. Thus, an optimal assignment for I_X induces an optimal assignment for I. ◀

▶ Remark 24. If Γ is SDS, the algorithm presented in Theorem 23 can in fact, for every fixed lower bound $l : D \to \mathbb{N}_0$ and every $\mathrm{VCSP}_l(\Gamma)$ instance I with optimal value $0 < \lambda < \infty$, enumerate all optimal solutions of I in polynomial time.

If Γ is SEDS and $\mathrm{fix}_0(\Gamma)$ is globally ℓ-tractable, this property holds true under the condition that for every $\mathrm{VCSP}_l(\mathrm{fix}_0(\Gamma))$ instance with optimal value $0 < \lambda < \infty$, all optimal solutions can be enumerated in polynomial time.

To complete our analysis of SEDS languages, we will now focus on the case that a language is not SDS and that $\mathrm{fix}_0(\Gamma)$ is globally s-intractable. Going even beyond SEDS, our main hardness result is that SIM languages are globally s-intractable under those circumstances. We only give a brief sketch of the proof here and provide the full proof in the full version of the paper [19].

▶ **Theorem 25.** *Let Γ be a valued constraint language over domain D that is SIM, but not SDS, and let $\mathrm{fix}_0(\Gamma)$ be globally s-intractable. Then Γ is globally s-intractable.*

Proof sketch. Since Γ is not SDS, there must exist a weighted relation $\gamma \in \Gamma$ of some arity r and disjoint $X_1, \ldots, X_k, Y_1, \ldots, Y_k \subseteq [r]$ such that, in violation of equation (SDS), the k-set function f corresponding under normalisation to γ satisfies

$$f(X_1, \ldots, X_k) \gg f(X_1 \cup Y_1, \ldots, X_k \cup Y_k).$$

Given any $\mathrm{VCSP}_s(\mathrm{fix}_0(\Gamma))$ instance $I^* = (V, D^*, \phi_I^*)$ on domain $D^* = D \setminus \{0\}$, we construct a $\mathrm{VCSP}_s(\Gamma)$ instance $I = (V \cup \{z\}, D, \phi_I)$ instance as follows. Let z denote an additional variable. The objective function ϕ_I consists of two components. The first component utilises the relation γ to ensure that any optimal assignment s for I must satisfy $s(z) = 0$ and $s(x) \neq 0$ for all $x \in V$. The second component models ϕ_I^* by replacing the constraints by corresponding weighted relations from Γ, where pinning values to label 0 is simulated by plugging in z.

This way, a solution for I^* can be obtained by solving I, thereby reducing $\mathrm{VCSP}_s(\mathrm{fix}_0(\Gamma))$ to $\mathrm{VCSP}_s(\Gamma)$. ◀

On the Boolean domain, we obtain a complete classification of lower-bounded VCSPs, which coincides with the classification of Boolean surjective VCSPs provided by [10].

▶ **Theorem 26.** *Let Γ be a Boolean language. Then Γ is globally ℓ-tractable if and only it is globally s-tractable. Otherwise, Γ is globally ℓ-intractable.*

Moreover, we can now classify lower-bounded VCSPs over SEDS languages on three-element domains.

▶ **Theorem 27.** *Let Γ be a SEDS language on domain $D = \{0, 1, 2\}$. Then Γ is globally ℓ-tractable if it is SDS or if $\mathrm{fix}_0(\Gamma)$ is globally ℓ-tractable, and globally ℓ-intractable otherwise.*

Proof. If Γ is SDS or $\mathrm{fix}_0(\Gamma)$ globally ℓ-tractable, the statement follows from Theorem 23. Otherwise, $\mathrm{fix}_0(\Gamma)$ must be globally s-intractable by Theorem 26 and the dichotomy from [10, Theorem 3.2]. Hence, Γ is globally s-intractable by Theorem 25, which gives the result. ◀

5 Conclusions

Based on the newly introduced Bounded Generalised Min-Cut problem and its tractability, which might be of independent interest, we have provided a conditional complexity classification of surjective and lower-bounded SEDS VCSPs on non-Boolean domains. Consequently, we obtained a dichotomy theorem with respect to ℓ-tractability for Boolean domains as well as, more interestingly, for SEDS languages on three-element domains.

While our results only pertain to languages that admit multimorphism $\langle c_d \rangle$ for some label d we expect our results and techniques to be useful in identifying new s-tractable and ℓ-tractable languages going beyond those admitting $\langle c_d \rangle$.

As mentioned in Section 1, globally tractable languages that include constant relations are also s-tractable. It is easy to show the same for global ℓ-tractability. For example, this shows that well-studied sources of tractability such as submodularity [22] and its generalisation k-submodularity [14], which are known to be globally tractable [17], are also globally ℓ-tractable.

What other non-Boolean languages are s-tractable and ℓ-tractable? Our results are a first step in this direction. In all cases we encountered global s-(in)tractability coincides with global ℓ-(in)tractability. We do not know whether this is true in general.

The natural next step is to consider languages on three-element domains. As is often the case in the (V)CSP literature, languages on three-element domains are significantly more complex than Boolean languages; for instance, compare two-element CSPs [20] and three-element CSPs [2]. As a concrete open problem (of a surjective VCSP on a three-element domain), the *3-No-Rainbow-Colouring* problem [1] asks to colour the vertices of a three-regular hypergraph such that every colour is used at least once, while no hyperedge attains all three colours. The complexity of this problem is open both the in the decision setting (is there a colouring) and also in the optimisation setting (what is the maximum number of properly colourable hyperedges/minimum number of improperly colourable hyperedges).

References

1 Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems – a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012. doi:10.1016/j.dam.2012.03.029.

2 Andrei Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006. doi:10.1145/1120582.1120584.

3 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

4 Andrei A. Bulatov. A Dichotomy Theorem for Nonuniform CSPs. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

5 Andrei A. Bulatov and Dániel Marx. The complexity of global cardinality constraints. *Logical Methods in Computer Science*, Volume 6, Issue 4, 2010. doi:10.2168/LMCS-6(4:4)2010.

6 David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Andrei A. Krokhin. The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006. doi:10.1016/j.artint.2006.04.002.

7 Nadia Creignou and Jean-Jacques Hébrard. On Generating All Solutions of Generalized Satisfiability Problems. *Informatique Théorique et Applications*, 31:499–511, November 1997. URL: http://www.numdam.org/article/ITA_1997__31_6_499_0.pdf.

8 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994. doi:10.1137/S0097539792225297.

9 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.

10 Peter Fulla, Hannes Uppman, and Stanislav Živný. The complexity of Boolean surjective general-valued CSPs. *ACM Transactions on Computation Theory*, 11(1), 2018. Article No. 4. doi:10.1145/3282429.

11 Olivier Goldschmidt and Dorit S. Hochbaum. A Polynomial Algorithm for the k-cut Problem for Fixed k. *Mathematics of Operations Research*, 19(1):24–37, 1994. doi:10.1287/moor.19.1.24.

12 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

13 Pavol Hell and Jaroslav Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008. doi:10.1016/j.cosrev.2008.10.003.

14 Anna Huber and Vladimir Kolmogorov. Towards Minimizing k-Submodular Functions. In *Proceedings of the 2nd International Symposium on Combinatorial Optimization (ISCO'12)*, volume 7422 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2012. doi:10.1007/978-3-642-32147-4_40.

15 Peter Jeavons, David Cohen, and Marc Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548, July 1997. doi:10.1145/263867.263489.

16 Vladimir Kolmogorov, Andrei Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. *SIAM Journal on Computing*, 46(3):1087–1110, 2017. doi:10.1137/16M1091836.

17 Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued CSPs. *SIAM Journal on Computing*, 44(1):1–36, 2015. doi:10.1137/130945648.

18 Marcin Kozik and Joanna Ochremiak. Algebraic Properties of Valued Constraint Satisfaction Problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 846–858. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-47672-7_69.

19 Gregor Matl and Stanislav Živný. Beyond Boolean Surjective VCSPs. Technical report, arXiv, January 2019. arXiv:1901.07107.

20 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

21 Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 631–637, 1995. URL: http://ijcai.org/Proceedings/95-1/Papers/083.pdf.

22 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

23 Johan Thapper and Stanislav Živný. The Complexity of Finite-Valued CSPs. *Journal of the ACM*, 63(4):37:1–37:33, September 2016. doi:10.1145/2974019.

24 D. Zhuk. A Proof of CSP Dichotomy Conjecture. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 331–342, 2017. doi:10.1109/FOCS.2017.38.

The Containment Problem for Unambiguous Register Automata

Antoine Mottet
Department of Algebra, Faculty of Mathematics and Physics, Charles University, Czech Republic

Karin Quaas
University of Oldenburg, Germany

Abstract

We investigate the complexity of the containment problem "Does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold?", where $\mathcal{B}$ is an unambiguous register automaton and $\mathcal{A}$ is an arbitrary register automaton. We prove that the problem is decidable and give upper bounds on the computational complexity in the general case, and when $\mathcal{B}$ is restricted to have a fixed number of registers.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Data words, Register automata, Unambiguous Automata, Containment Problem, Language Inclusion Problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.53

Funding *Antoine Mottet*: This author received funding from DFG Graduiertenkolleg 1763 (QuantLA) and from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 771005, "CoCoSym").
Karin Quaas: Supported by DFG, QU 316/1-2.

1 Introduction

Register automata [10] are a widely studied model of computation that extend finite automata with finitely many *registers* that are able to hold values from an infinite domain and perform equality comparisons with data from the input word. This allows register automata to accept *data languages*, i.e., sets of *data words* over $\Sigma \times \mathbb{D}$, where Σ is a finite alphabet and $\mathbb{D}$ is an infinite set called the data domain. The study of register automata is motivated by problems in formal verification and database theory, where the objects under study are accompanied by annotations (identification numbers, labels, parameters, ...), see the survey by Ségoufin [18]. One of the central problems in these areas is to check whether a given input document or program complies with a given input specification. In our context, this problem can be formalized as a *containment problem*: given two register automata $\mathcal{A}$ and $\mathcal{B}$, does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold, i.e., is the data language accepted by $\mathcal{A}$ included in the data language accepted by $\mathcal{B}$? Here, $\mathcal{B}$ is understood as a specification, and one wants to check whether $\mathcal{A}$ satisfies the specification. For arbitrary register automata, the containment problem is undecidable [14, 4]. It is known that one can recover decidability in two different ways. First, the containment problem is known to be PSPACE-complete when $\mathcal{B}$ is a deterministic register automaton [4]. This is a severe restriction on the expressive power of $\mathcal{B}$, and it is of practical interest to find natural classes of register automata that can be tackled algorithmically and that can express more properties than deterministic register automata. Secondly, one can recover decidability of the containment problem when $\mathcal{B}$ is a non-deterministic register automaton with a single register [10, 4]. However, in this setting, the problem is Ackermann-complete [6]; it can therefore hardly be considered tractable.

This motivates the study of *unambiguous* register automata, which are non-deterministic register automata for which every data word has at most one accepting run. Such automata are strictly more expressive than deterministic register automata [10, 11].

In the present paper, we investigate the complexity of the containment problem when $\mathcal{B}$ is restricted to be an unambiguous register automaton. We prove that the problem is decidable with a 2-EXPSPACE complexity, and is even decidable in EXPSPACE if the number of registers of $\mathcal{B}$ is a fixed constant. This is a striking difference to the non-deterministic case, where even for a fixed number of registers greater than 1 the problem is undecidable. Classically, one way to approach the containment problem (for general models of computation) is to reduce it to a reachability problem on an infinite state transition system, called the *synchronized state space of $\mathcal{A}$ and $\mathcal{B}$*, cf. [15]. Proving decidability or complexity upper bounds for the containment problem then amounts to finding criteria of termination or bounds on the complexity of a reachability algorithm on this space. In this paper, our techniques also rely on the analysis of the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$, where our main contribution is to provide a bound on the size of synchronized states that one needs to explore before being able to certify that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds. This bound is found by identifying elements of the synchronized state space whose behaviour is similar, and by showing that every element of the synchronized state space is equivalent to a small one. In the general case, where $\mathcal{B}$ is unambiguous and $\mathcal{A}$ is an arbitrary non-deterministic register automaton, we bound the size of the graph that one needs to inspect by a triple exponential in the size of $\mathcal{A}$ and $\mathcal{B}$. In the restricted case that $\mathcal{B}$ has a fixed number of registers, we proceed to give a better bound that is only doubly exponential in the size of $\mathcal{A}$ and $\mathcal{B}$.

Related Literature. A thorough study of the current literature on register automata reveals that there exists a variety of different definitions of register automata, partially with significantly different semantics. In this paper, we study register automata as originally introduced by Kaminski and Francez [10]. Such register automata process data words over an infinite data domain. The registers can take data values that appear in the input data word processed so far. The current input datum can be compared for (in)equality with the data that is stored in the registers. Kaminski and Francez study register automata mainly from a language-theoretic point of view; more results on the connection to logic, as well as the decidability status and computational complexity of classical decision problems like emptiness and containment are presented, e.g., in [17, 14, 4]. In [7], register automata over *ordered* data domains are studied.

Kaminski and Zeitlin [11] define a generalisation of the model in [10], in the following called *register automata with guessing*. The registers in such automata can non-deterministically reassign, or "guess", the datum of a register. In particular, such register automata can store data values that have not appeared in the input data word before, in contrast to the register automata in [10]. Register automata with guessing are strictly more expressive than register automata; for instance, there exists a register automaton with guessing that accepts the complement of the data language accepted by the register automaton in Figure 1 (Example 4 in [11]). Figueira [5] studies an alternating version of this model, also over ordered data domains. Colcombet [2, 1] considers *unambiguous* register automata with guessing. In Theorem 12 in [2], it is claimed that this automata class is effectively closed under complement, so that universality, containment and equivalence are decidable; however, to the best of our knowledge, this claim remains unproved.

Finally, unambiguity has become an important topic in automata theory, as witnessed by the growing body of literature in the recent years [8, 13, 3, 16]. In addition to the motivations mentioned above, unambiguous automata form an important model of computation due to

their *succinctness* compared to their deterministic counterparts. For example, it is known that unambiguous finite automata can be exponentially smaller than deterministic automata [12] while the fundamental problems (such as emptiness, universality, containment, equivalence) remain tractable.

2 Main Definitions

We study register automata as introduced in the seminal paper by Kaminski and Francez [10]. Throughout the paper, Σ denotes a finite alphabet, and $\mathbb{D}$ denotes an infinite set of data values. In our examples, we assume $\mathbb{D} = \mathbb{N}$, the set of non-negative integers. A *data word* is a finite sequence $(\sigma_1, d_1)\dots(\sigma_k, d_k) \in (\Sigma \times \mathbb{D})^*$. A *data language* is a set of data words. We use ε to denote the *empty data word*. The *length* k of a data word w is denoted by $|w|$. Given a data word w as above and $0 \le i \le k$, we define the infix $w(i,j] := (\sigma_{i+1}, d_{i+1})\dots(\sigma_j, d_j)$. Note that $w(i,i] = \varepsilon$. We use $\mathrm{data}(w)$ to denote the set $\{d_1, \dots, d_k\}$ of all data occurring in w. We use $\mathrm{proj}(w)$ to denote the projection of w onto Σ^*, i.e., the word $\sigma_1 \dots \sigma_k$.

Let $\mathbb{D}_\perp$ denote the set $\mathbb{D} \cup \{\perp\}$, where $\perp \notin \mathbb{D}$ is a fresh symbol not occurring in $\mathbb{D}$. A *partial isomorphism* of $\mathbb{D}_\perp$ is an injection $f\colon S \to \mathbb{D}_\perp$ with finite domain $S \subset \mathbb{D}_\perp$ such that if $\perp \in S$, then $f(\perp) = \perp$. We use boldface lower-case letters like $\boldsymbol{a}, \boldsymbol{b}, \dots$ to denote tuples in $\mathbb{D}_\perp^n$, where $n \in \mathbb{N}$. Given a tuple $\boldsymbol{a} \in \mathbb{D}_\perp^n$, we write a_i for its i-th component, and $\mathrm{data}(\boldsymbol{a})$ denotes the set $\{a_1, \dots, a_n\} \subseteq \mathbb{D}_\perp$ of all data occurring in $\boldsymbol{a}$.

Let $R = \{r_1, \dots, r_n\}$ be a finite set of *registers*. A *register valuation* is a mapping $\boldsymbol{a} : R \to \mathbb{D}_\perp$; we may write a_i as shorthand for $\boldsymbol{a}(r_i)$. Let $\mathbb{D}_\perp^R$ denote the set of all register valuations. Given $\lambda \subseteq R$ and $d \in \mathbb{D}$, define the register valuation $\boldsymbol{a}[\lambda \leftarrow d]$ by $(\boldsymbol{a}[\lambda \leftarrow d])(r_i) := d$ if $r_i \in \lambda$, and $(\boldsymbol{a}[\lambda \leftarrow d])(r_i) := a_i$ otherwise.

A *register constraint* over R is defined by the grammar

$$\phi ::= \texttt{true} \mid = r \mid \neg\phi \mid \phi \wedge \phi\,,$$

where $r \in R$. We use $\Phi(R)$ to denote the set of all register constraints over R. We may use $\neq r$ or $\phi_1 \vee \phi_2$ as shorthand for $\neg(= r)$ and $\neg(\neg\phi_1 \wedge \neg\phi_2)$, respectively. The satisfaction relation $\models$ for $\Phi(R)$ on $\mathbb{D}_\perp^R \times \mathbb{D}$ is defined by structural induction in the obvious way; e.g., $\boldsymbol{a}, d \models (= r_1 \wedge \neq r_2)$ if $a_1 = d$ and $a_2 \neq d$.

A *register automaton over* Σ is a tuple $\mathcal{A} = (R, \mathcal{L}, \ell_{\mathrm{in}}, \mathcal{L}_{\mathrm{acc}}, E)$, where

- R is a finite set of registers,
- $\mathcal{L}$ is a finite set of *locations*,
- $\ell_{\mathrm{in}} \in \mathcal{L}$ is the *initial location*,
- $\mathcal{L}_{\mathrm{acc}} \subseteq \mathcal{L}$ is the set of *accepting locations*, and
- $E \subseteq \mathcal{L} \times \Sigma \times \Phi(R) \times 2^R \times \mathcal{L}$ is a finite set of *edges*. We may write $\ell \xrightarrow{\sigma,\phi,\lambda} \ell'$ to denote an edge $(\ell, \sigma, \phi, \lambda, \ell') \in E$. Here, σ is the label of the edge, ϕ is the register constraint of the edge, and λ is the set of updated registers of the edge. A register constraint `true` is vacuously true and may be omitted; likewise we may omit λ if $\lambda = \emptyset$.

A *state* of $\mathcal{A}$ is a pair $(\ell, \boldsymbol{a}) \in \mathcal{L} \times \mathbb{D}_\perp^R$, where ℓ is the current location and $\boldsymbol{a}$ is the current register valuation. Given two states $(\ell, \boldsymbol{a})$ and $(\ell', \boldsymbol{a'})$ and some input letter $(\sigma, d) \in (\Sigma \times \mathbb{D})$, we postulate a transition $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell', \boldsymbol{a'})$ if there exists some edge $\ell \xrightarrow{\sigma,\phi,\lambda} \ell'$ such that $\boldsymbol{a}, d \models \phi$ and $\boldsymbol{a'} = \boldsymbol{a}[\lambda \leftarrow d]$. If the context is clear, we may omit the index $\mathcal{A}$ and write $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,d} (\ell', \boldsymbol{a'})$ instead of $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell', \boldsymbol{a'})$. We use $\longrightarrow^*$ to denote the reflexive transitive closure of $\longrightarrow$. A *run* of $\mathcal{A}$ on the data word $(\sigma_1, d_1)\dots(\sigma_k, d_k)$ is a sequence $(\ell_0, \boldsymbol{a^0}) \xrightarrow{\sigma_1,d_1} (\ell_1, \boldsymbol{a^1}) \xrightarrow{\sigma_2,d_2} \dots \xrightarrow{\sigma_k,d_k} (\ell_k, \boldsymbol{a^k})$ of transitions. We say that a run *starts in*

$(\ell, \boldsymbol{a})$ if $(\ell_0, \boldsymbol{a^0}) = (\ell, \boldsymbol{a})$. A run is *initialized* if it starts in $(\ell_{\text{in}}, \{\bot\}^R)$, and a run is *accepting* if $\ell_k \in \mathcal{L}_{\text{acc}}$. The data language *accepted* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of data words $w \in (\Sigma \times \mathbb{D})^*$ such that there exists an initialized accepting run of $\mathcal{A}$ on w.

We classify register automata into *deterministic register automata* (DRA), *unambiguous register automata* (URA), and *non-deterministic register automata* (NRA). A register automaton is a DRA if for every data word w there is at most one initialized run. A register automaton is a URA if for every data word w there is at most one initialized accepting run. A register automaton without any restriction is an NRA. We say that a data language $L \subseteq (\Sigma \times \mathbb{D})^*$ is DRA-recognizable (URA-recognizable and NRA-recognizable, respectively), if there exists a DRA (URA and NRA, respectively) $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$. We write **DRA**, **URA**, and **NRA** for the class of DRA-recognizable, URA-recognizable, and NRA-recognizable, respectively, data languages. Note that **DRA** $\subseteq$ **URA** $\subseteq$ **NRA**. Also note that, albeit a semantical property, the unambiguity of a register automaton can be decided using a simple extension of a product construction, cf. [2].

The *containment problem* is the following decision problem: given two register automata $\mathcal{A}$ and $\mathcal{B}$, does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold? We consider two more decision problems that stand in a close relation to the containment problem (namely, they both reduce to the containment problem): the *universality problem* is the question whether $L(\mathcal{B}) = (\Sigma \times \mathbb{D})^*$ for a given register automaton $\mathcal{B}$. The *equivalence problem* is to decide, given two register automata $\mathcal{A}$ and $\mathcal{B}$, whether $L(\mathcal{A}) = L(\mathcal{B})$.

3 Some Facts about Register Automata

For many computational models, a straightforward approach to solve the containment problem is by a reduction to the emptiness problem using the equivalence: $L(\mathcal{A}) \subseteq L(\mathcal{B})$ if, and only if, $L(\mathcal{A}) \cap \overline{L(\mathcal{B})} = \emptyset$. This approach proves useful for **DRA**, which is closed under complementation. Using the decidability of the emptiness problem for NRA, as well as the closure of **NRA** under intersection [10], we obtain the decidability of the containment problem for the case where $\mathcal{A}$ is an NRA and $\mathcal{B}$ is a DRA. More precisely, and using results in [4], the containment problem for this particular case is PSPACE-complete.

In contrast to **DRA**, the class **NRA** is not closed under complementation [10] so that the above approach must fail if $\mathcal{B}$ is an NRA. Indeed, it is well known that the containment problem for the case where $\mathcal{B}$ is an NRA is undecidable [4]. The proof is a reduction from the halting problem for Minsky machines: an NRA is capable to accept the complement of a set of data words encoding halting computations of a Minsky machine.

In this paper, we are interested in the containment problem for the case where $\mathcal{A}$ is an NRA and $\mathcal{B}$ is a URA. When attempting to solve this problem, an obvious idea is to ask whether the class **URA** is closed under complementation. Kaminski and Francez [10] proved that **URA** is *not* closed under complementation, and this even holds for the class of data languages that are accepted by URA that only use a single register. In Figure 1, we show a standard example of a URA for which the complement of the accepted data language cannot even be accepted by an NRA [11]. Intuitively, this automaton is unambiguous because it is not possible for two different runs of the automaton on some data word to reach the location ℓ_1 with the same register valuation at the same time. Therefore, at any time only one run can proceed to the accepting location ℓ_2. Note that this also implies **DRA** $\subsetneq$ **URA**.

An alternative approach for solving the containment problem is to explore the (possibly infinite) *synchronized state space of $\mathcal{A}$ and $\mathcal{B}$*, cf. [15]. Intuitively, the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$ stores for every state $(\ell, \boldsymbol{a})$ that $\mathcal{A}$ is in after processing a data word w

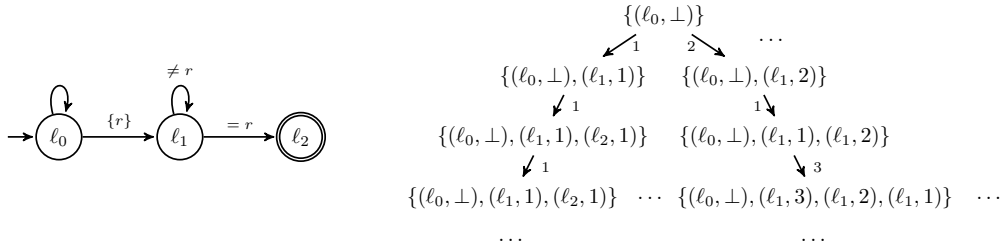


Figure 1 On the left we depict a URA with a single register r and over a singleton alphabet (we omit the labels at the edges). The complement of the data language accepted by this URA cannot be accepted by any NRA. On the right we show a finite part of the infinite state space of the URA.

the *set of states* that $\mathcal{B}$ is in after processing the same data word w. For an example, see the computation tree on the right side of Figure 1, where the leftmost branch shows the set of states that the URA on the left side of Figure 1 reaches after processing the data word $(\sigma, 1)(\sigma, 1)(\sigma, 1)$, and the rightmost branch shows the set of states that the URA reaches after processing the data word $(\sigma, 2)(\sigma, 1)(\sigma, 3)$. The key property of the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$ is that it contains sufficient information to decide whether for every data word for which there is an initialized accepting run in $\mathcal{A}$ there is also an initialized accepting run in $\mathcal{B}$. We formalize this intuition in the following paragraphs.

We start by defining the *state space* of a given NRA. Fix an NRA $\mathcal{A} = (R, \mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$ over Σ. A *configuration* of $\mathcal{A}$ is a finite set $C \subseteq (\mathcal{L} \times \mathbb{D}_\bot^R)$ of states of $\mathcal{A}$; if $C = \{(\ell, \boldsymbol{a})\}$ is a singleton set, in slight abuse of notation and if the context is clear, we may omit the parentheses and write $(\ell, \boldsymbol{a})$. Given a configuration C and an input letter $(\sigma, d) \in (\Sigma \times \mathbb{D})$, we use $\text{Succ}_{\mathcal{A}}(C, (\sigma, d))$ to denote the *successor configuration of C on the input* (σ, d), formally defined by

$$\text{Succ}_{\mathcal{A}}(C, (\sigma, d)) := \{(\ell, \boldsymbol{a}) \in (\mathcal{L} \times \mathbb{D}_\bot^R) \mid \exists (\ell', \boldsymbol{a'}) \in C. (\ell', \boldsymbol{a'}) \xrightarrow{\sigma, d}_{\mathcal{A}} (\ell, \boldsymbol{a})\}.$$

In order to extend this definition to data words, we define inductively $\text{Succ}_{\mathcal{A}}(C, \varepsilon) := C$ and $\text{Succ}_{\mathcal{A}}(C, w \cdot (\sigma, d)) := \text{Succ}_{\mathcal{A}}(\text{Succ}_{\mathcal{A}}(C, w), (\sigma, d))$. We say that a configuration C is *reachable in* $\mathcal{A}$ if there exists some data word w such that $C = \text{Succ}_{\mathcal{A}}((\ell_{\text{in}}, \{\bot\}^R), w)$. We say that a configuration C is *coverable in* $\mathcal{A}$ if there exists some configuration $C' \supseteq C$ such that C' is reachable in $\mathcal{A}$. We say that a configuration C is *accepting* if there exists $(\ell, \boldsymbol{a}) \in C$ such that $\ell \in \mathcal{L}_{\text{acc}}$; otherwise we say that C is *non-accepting*. We define $\text{data}(C) := \bigcup_{(\ell, \boldsymbol{a}) \in C} \text{data}(\boldsymbol{a})$ as the set of data occurring in configuration C.

The following proposition follows immediately from the definition of URA.

▶ **Proposition 1.** *If $\mathcal{A}$ is a* URA *and C, C' are two configurations of $\mathcal{A}$ such that $C \cap C' = \emptyset$ and $C \cup C'$ is coverable, then for every data word w the following holds: if* $\text{Succ}_{\mathcal{A}}(C, w)$ *is accepting, then* $\text{Succ}_{\mathcal{A}}(C', w)$ *is non-accepting.*

Let C, C' be two configurations of $\mathcal{A}$. Consider two data words $w = (\sigma_1, d_1) \dots (\sigma_k, d_k)$ and $w' = (\sigma_1, d'_1) \dots (\sigma_k, d'_k)$ such that $\text{proj}(w) = \text{proj}(w')$. Recall that a partial function $f \colon \mathbb{D}_\bot \to \mathbb{D}_\bot$ with finite domain is a *partial isomorphism* if it is an injection such that if $\bot \in \text{dom}(f)$ then $f(\bot) = \bot$. Let f be a partial isomorphism of $\mathbb{D}_\bot$ and let C be a configuration with $\text{data}(C) \subseteq \text{dom}(f)$. We define $f(C) := \{(\ell, f(d_1), \dots, f(d_{|R|})) \mid (\ell, d_1, \dots, d_{|R|}) \in C\}$; likewise, if $\{d_1, \dots, d_k\} \subseteq \text{dom}(f)$, we define $f((\sigma_1, d_1) \dots (\sigma_k, d_k)) := (\sigma_1, f(d_1)) \dots (\sigma_k, f(d_k))$. We say that C, w and C', w' are *equivalent with respect to* f, written $C, w \sim_f C', w'$, if

$$f(C) = C' \text{ and } f(w) = w'. \qquad (\star)$$

If $w = w' = \varepsilon$, then we may simply write $C \sim_f C'$. We write $C \sim C'$ if $C \sim_f C'$ for some partial isomorphism f of $\mathbb{D}_\perp$.

▶ **Proposition 2.** *If* $C, w \sim C', w'$, *then* $\mathrm{Succ}_{\mathcal{A}}(C, w(0,i]), w(i,k] \sim \mathrm{Succ}_{\mathcal{A}}(C', w'(0,i])$, $w'(i,k]$ *for all* $0 \leq i \leq k$, *where* $k = |w|$.

Proof. The proof is by induction on i. For the induction base, let $i = 0$. But then $\mathrm{Succ}_{\mathcal{A}}(C, w(0,0])) = \mathrm{Succ}_{\mathcal{A}}(C, \varepsilon) = C$ and $w(0,k] = w$, and similarly for C' and w', so that the statement holds by assumption. For the induction step, let $i > 0$. Define $C_{i-1} := \mathrm{Succ}_{\mathcal{A}}(C, w(0, i-1])$ and similarly C'_{i-1}. By induction hypothesis, there exists some bijective mapping

$$f_{i-1} : \mathrm{data}(C_{i-1}) \cup \mathrm{data}(w(i-1,k]) \to \mathrm{data}(C'_{i-1}) \cup \mathrm{data}(w'(i-1,k])$$

satisfying $(\star)$ $f_{i-1}(C_{i-1}) = C'_{i-1}$ and $f_{i-1}(w(i-1,k]) = w'(i-1,k]$. Define $C_i := \mathrm{Succ}_{\mathcal{A}}(C_{i-1}, (\sigma_i, d_i))$ and $C'_i := \mathrm{Succ}_{\mathcal{A}}(C'_{i-1}, (\sigma_i, d'_i))$. Note that $\mathrm{data}(C_i) \subseteq \mathrm{data}(C_{i-1}) \cup \{d_i\}$, and similarly for $\mathrm{data}(C'_i)$. Let f_i be the restriction of f_{i-1} to $\mathrm{data}(C_i) \cup \mathrm{data}(w(i,k])$. We are going to prove that $C_i, w(i,k] \sim_{f_i} C'_i, w'(i,k]$. Note that $f_i(w(i,k]) = w'(i,k]$ holds by definition of f_i and (2). We prove $f_i(C_i) \subseteq C'_i$. Suppose $(\ell, \boldsymbol{a}) \in C_i$. Hence there exists $(\ell_{i-1}, \boldsymbol{b}) \in C_{i-1}$ such that $(\ell_{i-1}, \boldsymbol{b}) \xrightarrow{\sigma_i, d_i} (\ell, \boldsymbol{a})$. Thus there exists an edge $\ell_{i-1} \xrightarrow{\sigma_i, \phi, \lambda} \ell$ such that $\boldsymbol{b}, d_i \models \phi$ and $\boldsymbol{a} = \boldsymbol{b}[\lambda \leftarrow d_i]$. By induction hypothesis, there exists $(\ell_{i-1}, \boldsymbol{b'}) \in C'_{i-1}$ such that $f_{i-1}(\boldsymbol{b}) = \boldsymbol{b'}$. By induction on the structure of ϕ, one can easily prove that $\boldsymbol{b}, d_i \models \phi$ if, and only if, $\boldsymbol{b'}, d'_i \models \phi$. Define $\boldsymbol{a'} := \boldsymbol{b'}[\lambda \leftarrow d'_i]$. We prove $f_i(\boldsymbol{a}) = \boldsymbol{a'}$: there are two cases: (i) If $r \in \lambda$, then $f_i(\boldsymbol{a}(r)) = f_i(d_i) = d'_i = \boldsymbol{a'}(r)$. (ii) If $r \notin \lambda$, then $f_i(\boldsymbol{a}(r)) = f_i(\boldsymbol{b}(r)) = f_{i-1}(\boldsymbol{b}(r)) = \boldsymbol{a'}(r)$. Hence, $f_i(\boldsymbol{a}) = \boldsymbol{a'}$. Altogether $(\ell, f_i(\boldsymbol{a})) \in C'_i$, and thus $f_i(C_i) \subseteq C'_i$. The proof for $C'_i \subseteq f_i(C_i)$ is analogous. Altogether, $C_i, w(i,k] \sim_{f_i} C'_i, w'(i,k]$. ◀

As an immediate consequence of Proposition 2, we obtain that $\sim$ preserves the configuration properties of being *accepting* respectively *non-accepting*.

▶ **Corollary 3.** *Let* C *and* C' *be two configurations of* $\mathcal{A}$. *If* $C, w \sim C', w'$ *and* $\mathrm{Succ}_{\mathcal{A}}(C, w)$ *is non-accepting (accepting, respectively), then* $\mathrm{Succ}_{\mathcal{A}}(C', w')$ *is non-accepting (accepting, respectively).*

Combining the last corollary with Proposition 1, we obtain

▶ **Corollary 4.** *If* $\mathcal{A}$ *is a* URA *and* C, C' *are two configurations such that* $C \cap C' = \emptyset$ *and* $C \cup C'$ *is coverable in* $\mathcal{A}$, *then for every data word* w *such that* $C, w \sim C', w$, *the configurations* $\mathrm{Succ}_{\mathcal{A}}(C, w)$ *and* $\mathrm{Succ}_{\mathcal{A}}(C', w)$ *are non-accepting.*

For the rest of this paper, let $\mathcal{A} = (R^{\mathcal{A}}, \mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{\mathrm{in}}, \mathcal{L}^{\mathcal{A}}_{\mathrm{acc}}, E^{\mathcal{A}})$ be an NRA over Σ, and let $\mathcal{B} = (R^{\mathcal{B}}, \mathcal{L}^{\mathcal{B}}, \ell^{\mathcal{B}}_{\mathrm{in}}, \mathcal{L}^{\mathcal{B}}_{\mathrm{acc}}, E^{\mathcal{B}})$ be a URA over Σ. Without loss of generality, we assume $R^{\mathcal{A}} \cap R^{\mathcal{B}} = \emptyset$ and $\mathcal{L}^{\mathcal{A}} \cap \mathcal{L}^{\mathcal{B}} = \emptyset$. We let m be the number of registers of $\mathcal{A}$, and we let n be the number of registers of $\mathcal{B}$.

A *synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$ is a pair $((\ell, \boldsymbol{d}), C)$, where $(\ell, \boldsymbol{d}) \in (\mathcal{L}^{\mathcal{A}} \times \mathbb{D}_\perp^{R^{\mathcal{A}}})$ is a single state of $\mathcal{A}$, and $C \subseteq (\mathcal{L}^{\mathcal{B}} \times \mathbb{D}_\perp^{R^{\mathcal{B}}})$ is a configuration of $\mathcal{B}$. Given a synchronized configuration S, we use $\mathrm{data}(S)$ to denote the set $\mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(C)$ of all data occurring in S. We define $S_{\mathrm{in}} := ((\ell^{\mathcal{A}}_{\mathrm{in}}, \{\perp\}^m), \{(\ell^{\mathcal{B}}_{\mathrm{in}}, \{\perp\}^n)\})$ to be the *initial synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$. We define the *synchronized state space of* $\mathcal{A}$ *and* $\mathcal{B}$ to be the (infinite) state transition system $(\mathbb{S}, \Rightarrow)$, where $\mathbb{S}$ is the set of all synchronized configurations of $\mathcal{A}$ and $\mathcal{B}$, and $\Rightarrow$ is defined as follows. If $S = ((\ell, \boldsymbol{d}), C)$ and $S' = ((\ell', \boldsymbol{d'}), C')$, then $S \Rightarrow S'$ if there

exists a letter $(\sigma, d) \in (\Sigma \times \mathbb{D})$ such that $(\ell, \boldsymbol{d}) \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell', \boldsymbol{d'})$, and $\mathrm{Succ}_{\mathcal{B}}(C, (\sigma, d)) = C'$. We say that a synchronized configuration S *reaches a synchronized configuration* S' *in* $(\mathbb{S}, \Rightarrow)$ if there exists a path in $(\mathbb{S}, \Rightarrow)$ from S to S'. We say that a synchronized configuration S is *reachable in* $(\mathbb{S}, \Rightarrow)$ if S_{in} reaches S. We say that a synchronized configuration $S = ((\ell, \boldsymbol{d}), C)$ is *coverable in* $(\mathbb{S}, \Rightarrow)$ if there exists some synchronized configuration $S' = ((\ell, \boldsymbol{d}), C')$ such that $C' \supseteq C$ and S' is reachable in $(\mathbb{S}, \Rightarrow)$.

We aim to reduce the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ to a reachability problem in $(\mathbb{S}, \Rightarrow)$. For this, call a synchronized configuration $((\ell, \boldsymbol{d}), C)$ *bad* if $\ell \in \mathcal{L}^{\mathcal{A}}_{\mathrm{acc}}$ is an accepting location and C is non-accepting, i.e., $\ell' \notin \mathcal{L}^{\mathcal{B}}_{\mathrm{acc}}$ for all $(\ell', \boldsymbol{a}) \in C$. The following proposition is easy to prove, cf. [15].

▶ **Proposition 5.** *$L(\mathcal{A}) \subseteq L(\mathcal{B})$ does not hold if, and only if, some bad synchronized configuration is reachable in $(\mathbb{S}, \Rightarrow)$.*

We extend the equivalence relation $\sim$ defined above to synchronized configurations in a natural manner, i.e, given a partial isomorphism f of $\mathbb{D}_\perp$ such that $\mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(C) \subseteq \mathrm{dom}(f)$, we define $((\ell, \boldsymbol{d}), C) \sim_f ((\ell, \boldsymbol{d'}), C')$ if $f(C) = C'$ and $f(\boldsymbol{d}) = \boldsymbol{d'}$. We shortly write $S \sim S'$ if there exists a partial isomorphism f of $\mathbb{D}_\perp$ such that $S \sim_f S'$. Clearly, an analogon of Proposition 2 holds for this extended relation. In particular, we have the following:

▶ **Proposition 6.** *Let S, S' be two synchronized configurations of $(\mathbb{S}, \Rightarrow)$ such that $S \sim S'$. If S reaches a bad synchronized configuration, so does S'.*

Note that the state transition system $(\mathbb{S}, \Rightarrow)$ is infinite. First of all, $(\mathbb{S}, \Rightarrow)$ is not finitely branching: for every synchronized configuration $S = ((\ell, \boldsymbol{d}), C)$ in $\mathbb{S}$, every datum $d \in \mathbb{D}$ may give rise to its own individual synchronized configuration S_d such that $S \Rightarrow S_d$. However, it can be easily seen that for every two different data values $d, d' \in \mathbb{D} \backslash \mathrm{data}(S)$, if inputting (σ, d) gives rise to a transition $S \Rightarrow S_d$ and inputting (σ, d') gives rise to a transition $S \Rightarrow S_{d'}$ (for some $\sigma \in \Sigma$), then $S_d \sim S_{d'}$. Hence there exist synchronized configurations $S_1, \ldots, S_k$ for some $k \in \mathbb{N}$ such that $S \Rightarrow S_i$ for all $i \in \{1, \ldots, k\}$, and such that for all $S' \in \mathbb{S}$ with $S \Rightarrow S'$ there exists $i \in \{1, \ldots, k\}$ such that $S_i \sim S'$. This is why we define in Section 4.3 the notion of *abstract configuration*, representing synchronized configurations up to the relation $\sim$. Second, and potentially more harmful for the termination of an algorithm to decide the reachability problem from Proposition 5, the configuration C of $\mathcal{B}$ in a synchronized configuration may grow unboundedly. As an example, consider the URA on the left side of Figure 1. For every $k \geq 1$, the configuration $\{(\ell_0, \perp), (\ell_1, d_1), (\ell_1, d_2) \ldots, (\ell_1, d_k)\}$ with pairwise distinct data values $d_1, \ldots, d_k$ is reachable in this URA by inputting the data word $(\sigma, d_1)(\sigma, d_2) \ldots (\sigma, d_k)$. In the next section, we prove that we can solve the reachability problem from Proposition 5 by focussing on a subset of configurations of $\mathcal{B}$ that are bounded in size, thus reducing to a reachability problem on a finite graph.

4 The Containment Problem for Register Automata

4.1 Types

Given $k \in \mathbb{N}$, a k-type[1] of $\mathbb{D}_\perp$ is a quantifier-free formula $\varphi(y_1, \ldots, y_k)$ formed by a conjunction of (positive or negative) literals of the form $y_i = y_j$ and $y_i = \perp$ that is satisfiable in $\mathbb{D}_\perp$. A k-type is *complete* if for any other quantifier-free formula $\psi(y_1, \ldots, y_k)$, either

[1] Types are a standard notion of model theory (see, e.g., [9] for a definition). The definition that we give here coincides with the standard notion of types when applied to $\mathbb{D}_\perp$.

$\forall y_1, \ldots, y_k.(\varphi(y_1, \ldots, y_k) \Rightarrow \psi(y_1, \ldots, y_k))$ holds or $\varphi \wedge \psi$ is unsatisfiable. It is easy to see that given $\boldsymbol{a} \in \mathbb{D}^k$, there is a unique complete k-type φ such that $\varphi(\boldsymbol{a})$ holds in $\mathbb{D}_\perp$. We call φ the *type of* $\boldsymbol{a}$ and denote it by $\mathrm{tp}(\boldsymbol{a})$. It may be observed that $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{D}_\perp^k$ have the same type if, and only if, there exists a partial isomorphism f of $\mathbb{D}_\perp$ such that $f(\boldsymbol{a}) = \boldsymbol{b}$.

Recall that m and n denote the number of registers of $\mathcal{A}$ and $\mathcal{B}$. For every $\boldsymbol{a} \in \mathbb{D}_\perp^n$ and for every complete $(2n+m)$-type $\varphi(\boldsymbol{y})$, where $\boldsymbol{y} = (y_1, \ldots, y_{2n+m})$, we define the set

$$\mathcal{L}_\varphi(\boldsymbol{a}) = \{\ell' \in \mathcal{L}^{\mathcal{B}} \mid \exists \boldsymbol{b} \in \mathbb{D}_\perp^n \text{ such that } (\ell', \boldsymbol{b}) \in C \text{ and } \varphi(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d}) \text{ holds in } \mathbb{D}_\perp\}.$$

Let $S = ((\ell, \boldsymbol{d}), C)$ be a synchronized configuration and let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{D}_\perp^n$ be two register valuations occurring in C, i.e., there exist $\ell_{\boldsymbol{a}}, \ell_{\boldsymbol{b}} \in \mathcal{L}^{\mathcal{B}}$ such that $(\ell_{\boldsymbol{a}}, \boldsymbol{a}), (\ell_{\boldsymbol{b}}, \boldsymbol{b}) \in C$. We say that $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *are indistinguishable in* S, written $\boldsymbol{a} \equiv_S \boldsymbol{b}$, if $\mathcal{L}_\varphi(\boldsymbol{a}) = \mathcal{L}_\varphi(\boldsymbol{b})$ for every complete $(2n+m)$-type $\varphi(\boldsymbol{y})$.

▶ **Example 7.** Let $(\ell^{\mathcal{A}}, 3)$ be a state in some NRA with a single register, and let $C' = \{(\ell, 1, 3), (\ell, 2, 3), (\ell', 1, 2)\}$ be a configuration of a URA with two registers. Let $S' = ((\ell^{\mathcal{A}}, 3), C')$ be the corresponding synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Consider $\boldsymbol{a} = (1, 3)$ and $\boldsymbol{b} = (2, 3)$. For the 5-type

$$\varphi_1 = (y_1 \neq y_2) \wedge (y_1 \neq y_3) \wedge (y_2 = y_4) \wedge (y_4 = y_5) \wedge (y_3 \neq y_2)$$

we have $\mathcal{L}_{\varphi_1}(\boldsymbol{a}) = \{\ell\}$ as $\varphi_1(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d})$ holds in $(\mathbb{N}, =)$, and similarly, $\mathcal{L}_{\varphi_1}(\boldsymbol{b}) = \{\ell\}$ as $\varphi_1(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{d})$ holds in $(\mathbb{N}, =)$. However, we have $\mathcal{L}_{\varphi_2}(\boldsymbol{a}) = \{\ell'\}$ and $\mathcal{L}_{\varphi_2}(\boldsymbol{b}) = \emptyset$ for the 5-type

$$\varphi_2 = (y_1 \neq y_2) \wedge (y_1 = y_3) \wedge (y_2 \neq y_4) \wedge (y_2 = y_5) \wedge (y_4 \neq y_1).$$

Hence $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ does *not* hold. However, $\boldsymbol{a} \equiv_S \boldsymbol{b}$ for $S = ((\ell^{\mathcal{A}}, 3), C)$ with $C := C' \cup \{(\ell', 2, 1)\}$.

▶ **Proposition 8.** *Let $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Let $\boldsymbol{a}, \boldsymbol{b}$ be such that $\boldsymbol{a} \equiv_S \boldsymbol{b}$. Then the map $f\colon \mathrm{data}(\boldsymbol{a}) \to \mathrm{data}(\boldsymbol{b})$ defined by $f(a_i) := b_i$ is a partial isomorphism of $\mathbb{D}_\perp$. Moreover, if we let $C_{\boldsymbol{a}} := \{(\ell, \boldsymbol{a}) \in C \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$ and $C_{\boldsymbol{b}} := \{(\ell, \boldsymbol{b}) \in C \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$, then $C_{\boldsymbol{a}} \sim_f C_{\boldsymbol{b}}$.*

Proof. Let φ be the complete $(2n+m)$-type of $(\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{d})$. Note that for two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{D}_\perp^n$, $\varphi(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{d})$ holds in $\mathbb{D}_\perp$ iff $\boldsymbol{u} = \boldsymbol{v}$ and $\mathrm{tp}(\boldsymbol{a}, \boldsymbol{d}) = \mathrm{tp}(\boldsymbol{u}, \boldsymbol{d}) = \mathrm{tp}(\boldsymbol{v}, \boldsymbol{d})$.

Let now $(\ell, \boldsymbol{a})$ be in $C_{\boldsymbol{a}}$. By definition, this means that $\ell \in \mathcal{L}_\varphi(\boldsymbol{a})$. By indistinguishability, $\ell \in \mathcal{L}_\varphi(\boldsymbol{b})$ so that

$$\varphi(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) \text{ holds in } \mathbb{D}_\perp \tag{$\dagger$}$$

for some $(\ell, \boldsymbol{c}) \in C$. Now, ($\dagger$) implies $\boldsymbol{b} = \boldsymbol{c}$ and $\mathrm{tp}(\boldsymbol{b}) = \mathrm{tp}(\boldsymbol{a})$. The former implies that $(\ell, \boldsymbol{b}) \in C_{\boldsymbol{b}}$, while the latter implies that f is a partial isomorphism. Conversely, we obtain that $(\ell, \boldsymbol{b}) \in C_{\boldsymbol{b}}$ implies $(\ell, \boldsymbol{a}) \in C_{\boldsymbol{a}}$. Hence $f(C_{\boldsymbol{a}}) = C_{\boldsymbol{b}}$ and thus $C_{\boldsymbol{a}} \sim_f C_{\boldsymbol{b}}$. ◀

4.2 Collapsing Configurations

As we pointed out in the introduction, the crucial ingredient of our algorithm for deciding whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds is to prevent configurations C in a synchronized configuration $((\ell, \boldsymbol{d}), C)$ to grow unboundedly. We do this by *collapsing two* subconfigurations $C_{\boldsymbol{a}}, C_{\boldsymbol{b}} \subseteq C$ that behave equivalently with respect to reaching a bad synchronized configuration in $(\mathbb{S}, \Rightarrow)$ into a *single* subconfiguration. The key notions for deciding when two subconfigurations can be collapsed into a single one are *k-types* and *indistinguishability* from the previous subsection.

▶ **Proposition 9.** *Let $S' = ((\ell, \boldsymbol{d}), C')$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two distinct register valuations in C' such that $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$. Let $C_{\boldsymbol{b}} := \{(\ell, \boldsymbol{b}) \in C' \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$. Then $S := ((\ell, \boldsymbol{d}), C' \setminus C_{\boldsymbol{b}})$ reaches a bad synchronized configuration if, and only if, S' reaches a bad synchronized configuration.*

Proof. The "if" direction follows from the simple observation that for every data word w, if $\mathrm{Succ}_{\mathcal{B}}(C', w)$ is non-accepting, then so is $\mathrm{Succ}_B(D, w)$ for every subset $D \subseteq C'$. For the "only if" direction, let $C_{\boldsymbol{a}} := \{(\ell, \boldsymbol{a}) \in C' \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$ and $C := C' \setminus (C_{\boldsymbol{a}} \cup C_{\boldsymbol{b}})$. Let m be the number of registers of $\mathcal{A}$ and n be the number of registers of $\mathcal{B}$. Suppose that there exists a data word w such that there exists an accepting run of $\mathcal{A}$ on w that starts in $(\ell, \boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{a}} \cup C, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{b}}, w)$ is accepting; otherwise we are done. Without loss of generality, we assume that $\mathrm{data}(w) \cap \mathrm{data}(S') \subseteq \mathrm{data}(\boldsymbol{b}) \cup \mathrm{data}(\boldsymbol{d})$. Otherwise, pick for every $d \in \mathrm{data}(w) \cap (\mathrm{data}(\boldsymbol{a}) \cup \mathrm{data}(C))$ such that $d \notin \mathrm{data}(\boldsymbol{b}) \cup \mathrm{data}(\boldsymbol{d})$, a fresh datum $d' \in \mathbb{D}$ not occurring in $\mathrm{data}(w) \cup \mathrm{data}(S')$, and simultaneously replace every occurrence of d in w by d'. Let w' be the resulting data word. Then $(\ell, \boldsymbol{d}), w \sim (\ell, \boldsymbol{d}), w'$ and $C_{\boldsymbol{b}}, w \sim C_{\boldsymbol{b}}, w'$. By Corollary 3, $\mathrm{Succ}_{\mathcal{A}}((\ell, \boldsymbol{d}), w')$ is accepting, and $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{b}}, w')$ is accepting, too. Then there must exist some accepting run of $\mathcal{A}$ on w' starting in $(\ell, \boldsymbol{d})$, and, by Proposition 1, $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{a}} \cup C, w')$ must be non-accepting. Hence, we could continue the proof with w' instead of w. Let us assume henceforth that $\mathrm{data}(w) \cap \mathrm{data}(S') \subseteq \mathrm{data}(\boldsymbol{b}) \cup \mathrm{data}(\boldsymbol{d})$ holds.

Let now w'' be the data word obtained from w as follows: for every $b_i \in \mathrm{data}(w)$ with $b_i \neq a_i$, pick some fresh datum $e_i \in \mathbb{D}$ not occurring in $\mathrm{data}(w) \cup \mathrm{data}(S')$. Then replace every occurrence of the letter b_i in w by e_i.

Note that $(\ell, \boldsymbol{d}), w \sim (\ell, \boldsymbol{d}), w''$: the key argument for this is that by $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ we have $b_i \notin \mathrm{data}(\boldsymbol{d})$ whenever $b_i \neq a_i$. By Corollary 3, $\mathrm{Succ}_{\mathcal{A}}((\ell, \boldsymbol{d}), w'')$ is accepting. Hence there must exist some accepting run of $\mathcal{A}$ on w'' starting in $(\ell, \boldsymbol{d})$.

Further note that $C_{\boldsymbol{a}}, w'' \sim C_{\boldsymbol{b}}, w''$: by Proposition 8, $C_{\boldsymbol{a}} \sim_f C_{\boldsymbol{b}}$, where $f : \mathrm{data}(\boldsymbol{a}) \to \mathrm{data}(\boldsymbol{b})$ is the bijective mapping defined by $f(a_i) = b_i$ for all $1 \leq i \leq n$. Now let $g : \mathrm{data}(\boldsymbol{a}) \cup \mathrm{data}(w'') \to \mathrm{data}(\boldsymbol{b}) \cup \mathrm{data}(w'')$ be the bijective mapping that agrees with f on all data in $\mathrm{data}(\boldsymbol{a})$, and that maps each datum $d \in \mathrm{data}(w'') \setminus \mathrm{data}(\boldsymbol{a})$ to d. One can easily see that g is a bijection such that $g(C_{\boldsymbol{a}}) = C_{\boldsymbol{b}}$ and $g(w'') = w''$ so that indeed $C_{\boldsymbol{a}}, w'' \sim_g C_{\boldsymbol{b}}, w''$. By Corollary 4, $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{a}}, w'')$ and $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{b}}, w'')$ are non-accepting.

Finally, we prove that $\mathrm{Succ}_{\mathcal{B}}(C, w'')$ is non-accepting, too. For this, let $(\ell', \boldsymbol{c}) \in C$; we prove that $\mathrm{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'')$ is non-accepting. We distinguish the following two cases:

- For all $1 \leq i \leq n$ with $a_i \neq b_i$ we have $b_i \notin \mathrm{data}(\boldsymbol{c})$. Then $(\ell', \boldsymbol{c}), w \sim (\ell', \boldsymbol{c}), w''$, as witnessed by the bijection f such that $f(b_i) = e_i$ for all $b_i \in \mathrm{data}(w)$ such that $b_i \neq a_i$, and that is the identity otherwise. Recall that by assumption $\mathrm{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w)$ is non-accepting. By Corollary 3, $\mathrm{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'')$ is non-accepting.
- There exists $1 \leq i \leq n$ such that $a_i \neq b_i$ and $b_i \in \mathrm{data}(\boldsymbol{c})$.
  Let $\varphi(\boldsymbol{y})$ be the $(2n + m)$-type of $(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$, and note that $\ell' \in \mathcal{L}_\varphi(\boldsymbol{b})$. By assumption $\ell' \in \mathcal{L}_\varphi(\boldsymbol{a})$ and there exists a state $(\ell', \boldsymbol{c'}) \in C$ such that $\varphi(\boldsymbol{a}, \boldsymbol{c'}, \boldsymbol{d})$ holds. Note that for all $1 \leq j \leq n$ such that $b_i = c_j$ we have $a_i = c'_j$. By assumption, $b_i = c_j$ for some $1 \leq j \leq n$. Since $a_i \neq b_i$, we can infer $c_j \neq c'_j$, and hence $(\ell', \boldsymbol{c}) \neq (\ell', \boldsymbol{c'})$. Next we prove $(\ell', \boldsymbol{c}), w'' \sim (\ell', \boldsymbol{c'}), w''$. We define $f : \mathrm{data}(\boldsymbol{c}) \cup \mathrm{data}(w'') \to \mathrm{data}(\boldsymbol{c'}) \cup \mathrm{data}(w'')$ as follows:

$$f\colon \begin{cases} c_p \mapsto c'_p & 1 \leq p \leq n \\ e \mapsto e & e \in \mathrm{data}(w'') \end{cases}$$

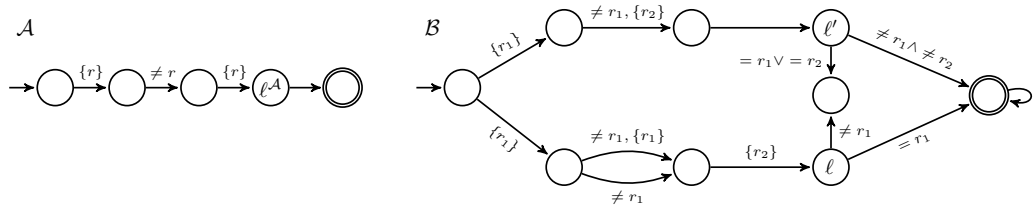


Figure 2 An NRA $\mathcal{A}$ and a URA $\mathcal{B}$ over a singleton alphabet for which $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

We prove below that

(i) for all $1 \leq p, q \leq n$, $c_p = c_q$ iff $c'_p = c'_q$;

(ii) for all $1 \leq p \leq n$, for all $e \in \text{data}(w'')$, $e = c_p$ iff $e = c'_p$;

note that this implies that f is well-defined and f is a bijective mapping, and hence $(\ell', \boldsymbol{c}), w'' \sim_f (\ell', \boldsymbol{c'}), w''$. By Proposition 2, $\text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'') \sim \text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c'}), w'')$. By Corollary 4, $\text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'')$ and $\text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c'}), w'')$ are non-accepting. We now prove the two items from above: (i) Follows directly from the fact that $\varphi(\boldsymbol{a}, \boldsymbol{c'}, \boldsymbol{d})$ and $\varphi(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$ hold, which implies that $\boldsymbol{c'}$ and $\boldsymbol{c}$ have the same type. For (ii), recall that $\text{data}(w) \cap \text{data}(S') \subseteq \text{data}(\boldsymbol{b}) \cup \text{data}(\boldsymbol{d})$. This, the definition of w'', and $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ yield the claim.

Altogether, we proved that $\text{Succ}_{\mathcal{B}}(C', w'')$ is non-accepting, while there exists some accepting run $(\ell, \boldsymbol{d}) \longrightarrow^* (\ell'', \boldsymbol{d''})$ of $\mathcal{A}$ on w''. This finishes the proof for the "only if" direction. ◀

When S is obtained from S' by applying Proposition 9 to some pair of register valuations, we say that S' *collapses to* S. We say that S is *maximally collapsed* if for all pairs $\boldsymbol{a}$ and $\boldsymbol{b}$ of distinct register valuations appearing in C we have that $\boldsymbol{a} \equiv_S \boldsymbol{b}$ does *not* hold. Note that in Proposition 9, the synchronized configuration S is again coverable. By iterating Proposition 9, one obtains that a coverable synchronized configurations reaches a bad synchronized configuration if, and only if, it collapses in finitely many steps to a maximally collapsed synchronized configuration that also reaches a bad synchronized configuration.

Before we present our algorithm for deciding the containment problem, we would like to point out that the intuitive notion of *types* alone is not sufficient for deciding whether synchronized configurations can be collapsed. More precisely, given a coverable synchronized configuration $S' = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C')$ and two register valuations $\boldsymbol{a}$ and $\boldsymbol{b}$ that occur in C' and for which $\text{tp}(\boldsymbol{a}, \boldsymbol{d}) = \text{tp}(\boldsymbol{b}, \boldsymbol{d})$, it is in general *not* the case that S' reaches a bad synchronized configuration if $S := ((\ell, \boldsymbol{d}), C' \backslash C_{\boldsymbol{b}})$, where $C_{\boldsymbol{b}} := \{(\ell, \boldsymbol{b}) \in C' \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$, reaches a bad synchronized configuration. To see that, consider Figure 2, where two register automata over a singleton alphabet (we omit the labels at the edges) are depicted: an NRA $\mathcal{A}$ with a single register r on the left side, and a URA $\mathcal{B}$ with two registers r_1 and r_2 on the right side. Note that $L(\mathcal{A}) \subseteq L(\mathcal{B})$. After processing the input data word $w = (\sigma, 1)(\sigma, 2)(\sigma, 3)$, the synchronized configuration $S' = ((\ell^{\mathcal{A}}, 3), C')$, where $C' := \{(\ell, 1, 3), (\ell, 2, 3), (\ell', 1, 2)\})$, is reached in the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$. For $\boldsymbol{a} = (1, 3)$ and $\boldsymbol{b} = (2, 3)$, we have $\text{tp}(\boldsymbol{a}, \boldsymbol{d}) = \text{tp}(\boldsymbol{b}, \boldsymbol{d})$, but $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ does not hold (cf. Example 7). Indeed, $\text{Succ}_{\mathcal{B}}(C' \backslash C_{\boldsymbol{b}}, (\sigma, 2))$ is non-accepting, while C' cannot reach any non-accepting configuration.

4.3 Abstract Configurations

In this section, we study synchronized configurations up to the equivalence relation $\sim$. Recall that m is the number of registers of $\mathcal{A}$ and n is the number of registers of $\mathcal{B}$. An *abstract synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$ is a tuple (ℓ, Γ, φ) where φ is a complete $(sn+m)$-type for some $s \in \mathbb{N}$, Γ is an s-tuple of subsets of $\mathcal{L}^{\mathcal{B}}$, and $\ell \in \mathcal{L}^{\mathcal{A}}$.

The *size* of an abstract synchronized configuration is defined to be $(sn+m)\log(sn+m) + s|\mathcal{L}^{\mathcal{B}}| + \log(|\mathcal{L}^{\mathcal{A}}|)$, which corresponds to the size needed on the tape of a Turing machine to encode an abstract synchronized configuration (where one encodes, for example, an $(sn+m)$-type by giving for each of the $sn+m$ variables, a number in $\{1,\dots,sn+m\}$ in a way that $y_i = y_j$ is a conjunct in φ iff y_i and y_j are assigned the same number).

Every synchronized configuration $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ gives rise to an abstract synchronized configuration in the following way: let $\boldsymbol{a}^1,\dots,\boldsymbol{a}^s$ be the distinct register valuations in C, listed in some arbitrary order. Let φ be the complete $(sn+m)$-type of $(\boldsymbol{a}^1,\dots,\boldsymbol{a}^s,\boldsymbol{d})$. Let $C_{\boldsymbol{a}^i} := \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell,\boldsymbol{a}^i) \in C\}$. We obtain an abstract synchronized configuration $(\ell^{\mathcal{A}}, (C_{\boldsymbol{a}^1},\dots,C_{\boldsymbol{a}^s}), \varphi)$. Different enumerations of the register valuations of C can yield different abstract configurations. We let $\mathrm{abs}(S)$ be the set of all abstract synchronized configurations that can be obtained from S. Every two abstract synchronized configurations in $\mathrm{abs}(S)$ can be obtained from one another by permuting the variables from the type and the entries from the tuple accordingly. It is easy to prove that $S \sim S'$ if, and only if, $\mathrm{abs}(S) = \mathrm{abs}(S')$.

An abstract configuration (ℓ,Γ,φ) is said to be *maximally collapsed* if there exists a synchronized configuration S such that $(\ell,\Gamma,\varphi) \in \mathrm{abs}(S)$ and such that S is maximally collapsed (equivalently, one could ask that *every* S such that $(\ell,\Gamma,\varphi) \in \mathrm{abs}(S)$ is maximally collapsed). The main result of this section is that the number of different register valuations in a maximally collapsed synchronized configuration is bounded. Let $B_r \leq r^r$ be the number of complete r-types, which is also called the *Bell number* of order r.

▶ **Proposition 10.** *Let $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ be a maximally collapsed synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. The number of different register valuations appearing in C is bounded by $(B_{2n+m} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(2n+m)^n}$.*

Proof. We first prove a slightly worse upper bound, to give an idea of the proof. Let $K := B_{2n+m}$. We prove that the number of different register valuations is bounded by $2^{|\mathcal{L}^{\mathcal{B}}|K}$. Associate with every register valuation $\boldsymbol{a}$ appearing in C the K-tuple $(\mathcal{L}_{\varphi_1}(\boldsymbol{a}),\dots,\mathcal{L}_{\varphi_K}(\boldsymbol{a}))$ of subsets of $\mathcal{L}^{\mathcal{B}}$, where $\varphi_1,\dots,\varphi_K$ is an enumeration of all the complete $(2n+m)$-types. Note that there are at most $2^{|\mathcal{L}^{\mathcal{B}}|K}$ such tuples. Suppose by contradiction that S contains more than $2^{|\mathcal{L}^{\mathcal{B}}|K}$ different register valuations. By the pigeonhole principle there are two different register valuations $\boldsymbol{a}$ and $\boldsymbol{b}$ that have the same associated K-tuple. Note that if $\boldsymbol{a}$ and $\boldsymbol{b}$ share the same K-tuple, then $\boldsymbol{a} \equiv_S \boldsymbol{b}$. By Proposition 9, S could be collapsed further, contradiction. Hence, we proved an upper bound of $2^{|\mathcal{L}^{\mathcal{B}}|K}$ on the number of different register valuations appearing in a given maximally collapsed synchronized configuration.

We now proceed to prove the actual bound. The important fact is that when $\boldsymbol{a}$ and $\boldsymbol{d}$ are fixed in S, then few (i.e., $\leq (2n+m)^n$) entries in the tuple $(\mathcal{L}_{\varphi_1}(\boldsymbol{a}),\dots,\mathcal{L}_{\varphi_K}(\boldsymbol{a}))$ are non-empty. Indeed, in a given $(2n+m)$-type, each of the variables $y_{n+1},\dots,y_{2n}$ can be constrained to be equal to one of $y_1,\dots,y_n,y_{2n+1},\dots,y_{2n+m}$, or constrained to be different than all of them.

Therefore, it remains to bound the number of K-tuples with entries in $2^{\mathcal{L}^{\mathcal{B}}}$ and with at most $(2n+m)^n$ non-empty entries. Each such tuple is characterised by the subset $T \subseteq \{1,\dots,K\}$ of entries that are non-empty, together with a $|T|$-tuple of non-empty subsets of $\mathcal{L}^{\mathcal{B}}$. Since $|T|$ can be bounded by $(2n+m)^n$, we obtain that there are at most $K^{(2n+m)^n} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|(2n+m)^n}$ possible tuples, and thus at most $(B_{2n+m} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(2n+m)^n}$ different register valuations. ◀

Note that the bound in Proposition 10 is doubly exponential in n and exponential in $|\mathcal{L}^{\mathcal{B}}|$ and m. As a direct corollary, we obtain a bound on the number of maximally collapsed abstract synchronized configurations.

▶ **Proposition 11.** *The number of maximally collapsed abstract configurations is bounded by a triple exponential in $|\mathcal{A}|$ and $|\mathcal{B}|$. If the number of registers of $\mathcal{B}$ is fixed, then the number of maximally collapsed abstract configurations is bounded by a double exponential in $|\mathcal{A}|$ and $|\mathcal{B}|$.*

Proof. Recall that m is the number of registers of $\mathcal{A}$ and n is the number of registers of $\mathcal{B}$. By Proposition 10, a maximally collapsed synchronized configuration $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ is such that C contains at most $K := (B_{2n+m} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(2n+m)^n}$ different register valuations. Therefore, any abstract synchronized configuration in $\mathrm{abs}(S)$ is described by an $(sn+m)$-type with $s \leq K$. For a given s, there are at most $B_{sn+m} \cdot |\mathcal{L}^{\mathcal{B}}|^s \cdot |\mathcal{L}^{\mathcal{A}}|$ different abstract synchronized configurations. Summing up from $s = 0$ to K, we obtain that there are at most

$$\begin{aligned}\sum_{s=0}^{K} B_{sn+m} \cdot |\mathcal{L}^{\mathcal{B}}|^s \cdot |\mathcal{L}^{\mathcal{A}}| &\leq |\mathcal{L}^{\mathcal{A}}| \cdot \left(B_m + B_{n+m}|\mathcal{L}^{\mathcal{B}}| + \cdots + B_{nK+m} \cdot |\mathcal{L}^{\mathcal{B}}|^K\right) \\ &\leq |\mathcal{L}^{\mathcal{A}}| \cdot (1+K) \cdot B_{nK+m} \cdot |\mathcal{L}^{\mathcal{B}}|^K \\ &\leq |\mathcal{L}^{\mathcal{A}}| \cdot (1+K) \cdot (nK+m)^{(nK+m)} \cdot |\mathcal{L}^{\mathcal{B}}|^K\end{aligned}$$

maximally collapsed abstract synchronized configurations. Since K is doubly exponential in $|\mathcal{A}|$ and $|\mathcal{B}|$, this gives the first result. The second result follows from the fact that for fixed n, K only depends exponentially on m and $|\mathcal{L}^{\mathcal{B}}|$. ◀

Given abstract synchronized configurations $(\ell^{\mathcal{A}}, \Gamma, \varphi)$ and $(\ell'^{\mathcal{A}}, \Gamma', \varphi')$, define $(\ell^{\mathcal{A}}, \Gamma, \varphi) \rightsquigarrow (\ell'^{\mathcal{A}}, \Gamma', \varphi')$ if there exist synchronized configurations S and S' such that:

- $S \Rightarrow S'$,
- $(\ell^{\mathcal{A}}, \Gamma, \varphi)$ is in $\mathrm{abs}(S)$,
- S' can be maximally collapsed to some S'' such that $(\ell'^{\mathcal{A}}, \Gamma', \varphi')$ is in $\mathrm{abs}(S'')$.

▶ **Lemma 12.** *Given two abstract synchronized configurations $(\ell^{\mathcal{A}}, \Gamma, \varphi)$ and $(\ell'^{\mathcal{A}}, \Gamma', \varphi')$, deciding whether $(\ell^{\mathcal{A}}, \Gamma, \varphi) \rightsquigarrow (\ell'^{\mathcal{A}}, \Gamma', \varphi')$ holds can be done in polynomial space.*

Proof. In this proof, we assume without loss of generality that $\mathbb{D} = \mathbb{N}$. Let s be such that φ is an $(sn+m)$-type. Note that there is a synchronized configuration S of the form $((\ell^{\mathcal{A}}, \boldsymbol{d}), D)$ such that $\mathrm{data}(D) \cup \mathrm{data}(\boldsymbol{d}) \subseteq \{1, \ldots, sn+m\}$ and such that $(\ell^{\mathcal{A}}, \Gamma, \varphi) \in \mathrm{abs}(S)$. This S is moreover computable in polynomial space.

To decide whether $(\ell^{\mathcal{A}}, \Gamma, \varphi) \rightsquigarrow (\ell'^{\mathcal{A}}, \Gamma', \varphi')$ holds, one simply:

- guesses a letter $\sigma \in \Sigma$ and a datum d in $\{1, \ldots, sn+m+1\}$,
- computes a synchronized configuration S' obtained by firing the transition corresponding to (σ, d) from S,
- guesses a sequence $(\boldsymbol{a}^1, \boldsymbol{b}^1), \ldots, (\boldsymbol{a}^r, \boldsymbol{b}^r)$ of register valuations such that Proposition 9 can be applied r times to obtain a maximally collapsed configuration S'',
- checks that $(\ell'^{\mathcal{A}}, \Gamma', \varphi')$ is in $\mathrm{abs}(S'')$.

At the second step, the size of S' is polynomially bounded by the size of $\mathcal{A}$, $\mathcal{B}$, and of S. Moreover, the maximal length of a collapsing sequence in the third step is also polynomially bounded, as the number of distinct register valuations decreases after each application of Proposition 9. Therefore, this algorithm uses a polynomial amount of space. ◀

As for synchronized configuration, an abstract synchronized configuration $(\ell^{\mathcal{A}}, \Gamma, \varphi)$ is called *bad* if $\ell^{\mathcal{A}}$ is an accepting location and none of the states in Γ contains an accepting location.

▶ **Proposition 13.** *A bad synchronized configuration is reachable in $(\mathbb{S}, \Rightarrow)$ if, and only if, a bad abstract synchronized configuration is reachable from $\mathrm{abs}(S_{\mathrm{in}})$.*

Proof. We prove that for every coverable synchronized configuration S and every $n \geq 0$, a bad synchronized configuration is reachable in n steps from S if, and only if, a bad abstract synchronized configuration is reachable in n steps from $\mathrm{abs}(S)$. The statement then follows by taking $S := S_{\mathrm{in}}$. The proof goes by induction on n, where the case $n = 0$ is trivial in both directions.

Suppose now that S reaches a bad synchronized configuration in n steps. Let S' be such that $S \Rightarrow S'$ and such that S' reaches a bad synchronized configuration in $n-1$ steps. Let S'' be such that S' can be maximally collapsed to S''. By iterating Proposition 9, we have that S'' reaches a bad synchronized configuration in $n-1$ steps (the fact that the length of the path is unchanged can be seen from the proof of Proposition 9). It follows from the induction hypothesis that some $(\ell', \Gamma', \varphi') \in \mathrm{abs}(S'')$ reaches a bad abstract synchronized configuration in $n-1$ steps. Let (ℓ, Γ, φ) be an arbitrary abstraction in $\mathrm{abs}(S)$. We have by definition $(\ell, \Gamma, \varphi) \rightsquigarrow (\ell', \Gamma', \varphi')$, so that (ℓ, Γ, φ) reaches a bad abstract synchronized configuration in n steps. The converse direction is proved similarly. ◀

Finally, we are able to present the main theorem.

▶ **Theorem 14.** *The containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ is a non-deterministic register automaton and $\mathcal{B}$ is an unambiguous register automaton, is in 2-EXPSPACE. If the number of registers of $\mathcal{B}$ is fixed, the problem is in EXPSPACE.*

Proof. The algorithm checks whether a bad abstract synchronized configuration is reachable from $\mathrm{abs}(S_{\mathrm{in}})$, using the classical non-deterministic logspace algorithm for reachability. Every node of the graph can be stored using double-exponential space (see the second paragraph at the beginning of Section 4.3), and the size of the graph is triply exponential in the size of $\mathcal{A}$ and $\mathcal{B}$ by Proposition 11. Moreover, the relation $\rightsquigarrow$ is decidable in polynomial space by Lemma 12. Therefore, we obtain that the algorithm uses at most a double-exponential amount of space. In case the number of registers of $\mathcal{B}$ is fixed, Proposition 11 implies that the size of the graph is doubly exponential in the size of $\mathcal{A}$ and $\mathcal{B}$. We obtain that the algorithm uses at most an exponential amount of space. ◀

As an immediate corollary of Theorem 14, we obtain that the universality problem is in 2-EXPSPACE and in PSPACE for fixed number of registers. Similarly, the equivalence problem for unambiguous register automata is in 2-EXPSPACE.

5 Open Problems

The most obvious problem is to figure out the *exact* computational complexity of the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, when $\mathcal{B}$ is an URA. Finding lower bounds for unambiguous automata is a hard problem. Techniques for proving lower complexity bounds of the containment problem (respectively the universality problem) for the case where $\mathcal{B}$ is a non-deterministic automaton rely heavily on non-determinism (cf. Theorem 5.2 in [4]); as was already pointed out in [2], we are lacking techniques for finding lower computational complexity bounds for the case where $\mathcal{B}$ is unambiguous, even for the class of finite automata. Concerning the upper bound, computer experiments revealed that maximally collapsed synchronized configurations seem to remain small. Based on these experiments, we believe that the bound in Proposition 10 is not optimal and can be improved to $O(2^{poly(n,m,|\mathcal{L}^{\mathcal{B}}|)})$. If this is correct, we would obtain an EXPSPACE upper-bound for the general containment problem.

We also would like to study to what extent our techniques can be used to solve the containment problem for other computation models. In particular, we are interested in the following:

- One can extend the definition of register automata to work over an ordered domain, where the register constraints are of the form $< r$ and $> r$. Proposition 9 turns out to be false in this setting, but it seems plausible that there exists a collapsibility notion that would work for this model.
- An automaton $\mathcal{B}$ is said to be k-ambiguous if it has at most k accepting runs for every input data word, and polynomially ambiguous if the number of accepting runs for some input data word w is bounded by $p(|w|)$ for some polynomial p. Again, it is likely that simple modifications of Proposition 9 would give an algorithm for the containment problem for k-ambiguous register automata.
- Last but not least, we would like to point out that our techniques cannot directly be applied to the class of unambiguous register automata *with guessing* which we mentioned in the introduction. Thus, the respective containment problem remains open for future research.

References

1 Thomas Colcombet. Forms of Determinism for Automata (Invited Talk). In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.1.

2 Thomas Colcombet. Unambiguity in Automata Theory. In Jeffrey Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

3 Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When is Containment Decidable for Probabilistic Automata? 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 121:1–121:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.121.

4 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009. doi:10.1145/1507244.1507246.

5 Diego Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:22)2012.

6 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

7 Diego Figueira, Piotr Hofman, and Slawomir Lasota. Relating timed and register automata. In Sibylle B. Fröschle and Frank D. Valencia, editors, *Proceedings 17th International Workshop on Expressiveness in Concurrency, EXPRESS'10, Paris, France, August 30th, 2010.*, volume 41 of *EPTCS*, pages 61–75, 2010. doi:10.4204/EPTCS.41.5.

8 Nathanaël Fijalkow, Cristian Riveros, and James Worrell. Probabilistic Automata of Bounded Ambiguity. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.19.

9 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.

10 Michael Kaminski and Nissim Francez. Finite-Memory Automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

11 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *International Journal of Foundations of Computer Science*, Volume 21, Issue 05, 2010.

12 Hing Leung. Descriptional complexity of nfa of different ambiguity. *Int. J. Found. Comput. Sci.*, 16(5):975–984, 2005. doi:10.1142/S0129054105003418.

13 Michał Skrzypczak. Unambiguous Languages Exhaust the Index Hierarchy. 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 140:1–140:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.140.

14 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.

15 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.

16 Mikhail Raskin. A Superpolynomial Lower Bound for the Size of Non-Deterministic Complement of an Unambiguous Automaton. 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 138:1–138:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.138.

17 Hiroshi Sakamoto and Daisuke Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000. doi:10.1016/S0304-3975(99)00105-X.

18 Luc Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006. doi:10.1007/11874683_3.

Stabilization Time in Weighted Minority Processes

Pál András Papp
ETH Zürich, Switzerland
apapp@ethz.ch

Roger Wattenhofer
ETH Zürich, Switzerland
wattenhofer@ethz.ch

Abstract

A minority process in a weighted graph is a dynamically changing coloring. Each node repeatedly changes its color in order to minimize the sum of weighted conflicts with its neighbors. We study the number of steps until such a process stabilizes. Our main contribution is an exponential lower bound on stabilization time. We first present a construction showing this bound in the adversarial sequential model, and then we show how to extend the construction to establish the same bound in the benevolent sequential model, as well as in any reasonable concurrent model. Furthermore, we show that the stabilization time of our construction remains exponential even for very strict switching conditions, namely, if a node only changes color when almost all (i.e., any specific fraction) of its neighbors have the same color. Our lower bound works in a wide range of settings, both for node-weighted and edge-weighted graphs, or if we restrict minority processes to the class of sparse graphs.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Mathematics of computing → Graph algorithms; Theory of computation → Distributed computing models; Theory of computation → Self-organization

Keywords and phrases Minority process, Benevolent model

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.54

Related Version The full version of the paper is available online as arXiv preprint `https://arxiv.org/abs/1902.01228`.

1 Introduction

Given a simple graph and an initial coloring of its nodes, a minority process is a sequence of states (colorings) such that each state is obtained from the previous state by some of the nodes deciding to change their color. Each node, when it has the opportunity to act, switches to the least frequent color in its neighborhood. This may then prompt other neighbors of the node to switch their color, too, leading to a sequence of steps and a dynamically changing coloring. A state is stable when no node in the graph wants to change its color anymore, and the number of steps until a stable state is reached is known as the stabilization time of the process.

Minority processes have numerous applications in different areas where agents in a system are motivated to anti-coordinate with their neighbors. Assume, for instance, a set of wireless devices, each using a given frequency from a predefined set of frequencies for communication. In order to minimize interference with their neighbors, each device may repeatedly decide to switch to the frequency which is the least used in its neighborhood. In another setting, assume that some companies need to decide which product or commodity to produce, and they repeatedly adjust their strategy to avoid competition with specific other companies (that are e.g. geographically close, or share the same costumer base) [16]. Minority processes also appear in a wide range of other areas, including cellular biology [10], physics [6, 7] and social sciences [9].

It is often quite natural to model such settings not only as graphs, but as weighted graphs, since in many applications, either the nodes or edges of the graphs naturally exhibit some kind of weights that define their importance in the minority setting. For example, when selecting products, some competitors may be larger or more resourceful than others, and thus it is more crucial for their neighbors to differentiate from these specific nodes. In the frequency allocation setting, some nodes may handle much more traffic than others, and thus it is more important to avoid interference with such neighbors. Frequency allocation also provides a natural example for edge weights, since the severity of interference can also depend on the distance between neighboring devices, and thus it might be more imperative for nodes to avoid interference with closer neighbors.

The paper considers minority processes in these weighted cases, when the cost function of a node to minimize is not simply the number of its conflicts, but the sum of these conflicts multiplied by the weight of the neighboring node or by the weight of the connecting edge. In such a weighted setting, the only straightforward upper bound on the number of steps is exponential. In this paper, we prove an asymptotically matching lower bound of $2^{\Theta(n)}$, showing that there are weighted graphs where stabilization can indeed last for an exponential number of steps.

For a realistic analysis of stabilization time in applications, some further aspects of the processes are also worth studying. To avoid unreasonably many switches, nodes may decide not to switch color if this benefit is too small. Thus it is often more reasonable to assume a proportional switching rule in the weighted setting, i.e. that a node only decides to change its color if this reduces its cost at least by a given fraction of its weighted degree (or, equivalently, if a large fraction of its neighborhood has the same color). Note that this is a significantly stricter switching rule, and thus proving a lower bound on the number of steps under this rule is a stronger result. Furthermore, in most application areas, the underlying graphs are sparse, i.e. contain only $O(n)$ edges, so it is also interesting to study if the behavior is different when restricting ourselves to sparse graph instances.

There are multiple different models to study minority processes, sequential and concurrent alike. Even in the sequential setting, when only one node switches in each step, we can observe different behaviors depending on the order in which the nodes are selected. For example, this order may be chosen by a benevolent player who aims to minimize stabilization time, or an adversarial player aiming to maximize it. While stabilization time in these models have been studied thoroughly in the related area of majority processes, stabilization time in minority processes has remained open.

In the paper, we present weighted graph constructions that prove an exponential lower bound on stabilization time. Our lower bound holds both for node-weighted and edge-weighted graphs, for any number of colors, and also if we restrict the process to the class of sparse graphs.

The main contributions of the paper are as follows. We first present a construction that shows an exponential lower bound in the adversarial model. Then with further improvements to the construction, we prove that the same bound also holds in the benevolent model. This shows that there are graphs where not only one, but every possible run of the process takes exponential time. Moreover, we also show that the lower bound holds not only for the sequential process, but also in any reasonable concurrent setting. Our lower bounds are shown for a very strict switching rule, when a node is only allowed to switch if a given fraction of its neighbors have the same color. Most surprisingly, our results show that even with this rule, the exponential lower bound holds for any non-trivial fraction of the neighborhood.

2 Related Work

The question of stabilization time has only been studied in detail for majority processes. In [15], the authors devise a weighted graph construction, which exhibits a majority process with $2^{\Theta(n)}$ stabilization time both in the synchronous and the adversarial sequential models (benevolent models are not discussed in this paper). For the unweighted case, the stabilization time of majority processes has been characterized by [11] in the synchronous, sequential adversarial and sequential benevolent models. The study of [15] also shows further results on some slightly different variants of majority processes in unweigthed graphs. On the other hand, apart from a straightforward $O(n^2)$ upper bound in the unweighted case [14, 15], to our knowledge, the stabilization time of minority processes in these models has remained open so far.

However, for unweighted graphs, there are numerous theoretical studies that focus on different properties of stable states, both in case of minority [16, 3, 21, 1, 8] and majority [3, 12, 13, 20, 4, 2, 5] processes.

Minority processes have also been thoroughly studied in special classes of graphs, such as grids, trees or cycles, by the cellular automata community [17, 18, 19]. However, these results work with unweighted graphs, and a different variant of the minority process which considers the closed neighborhood of nodes. Besides the theoretical results, some of these studies also include an experimental analysis of the process on grids.

Papers working with minority processes almost always consider the basic switching rule, i.e. when nodes switch color for any small amount of improvement (although they sometimes assume different rules for tie-breaking). Some slightly different switching rules, based on distance-2 neighborhood of nodes, are examined in [14]; however, the aim of these modified rules is not to achieve earlier stabilization, but to reduce the number of conflicts in the final (stable) state. To our knowledge, however, minority processes have not yet been studied under the proportional switching rule.

3 Models and Notation

Preliminaries and notation

In the paper, we consider simple, undirected graphs, denoted by $G = (V, E)$, with V being the set of nodes and E the set of edges. The number of nodes is denoted by n, the edge between vertices u and v is denoted by $e(u, v)$. In case of *node-weighted graphs*, we assume a positive weight function $w : V \to \mathbb{R}^+$ on the nodes of the graph, while for *edge-weighted graphs*, we assume $w : E \to \mathbb{R}^+$ on the edges.

For a specific node $v \in V$, we denote by $N(v)$ the neighborhood of v. In case of node-weighted graphs, for a set $S \subseteq V$, we denote by W_S the sum of weights $\sum_{u \in S} w(u)$. Specifically, we use $W_{N(v)}$ to denote the sum of weights in v's neighborhood.

Given a set of colors Γ, a *coloring* is a function $C : V \to \Gamma$. If for some edge $e(u, v)$ we have $C(u) = C(v)$, then we have a *conflict*, and the edge in question is a *conflicting edge*. Generally, the goal of graph coloring is to minimize the number of conflicts in the graph.

We also use the notation $N_S(v) := \{u \mid u \in N(v) \text{ and } C(u) = C(v)\}$ and $N_O(v) := N(v) \setminus N_S(v)$ for a node v under a coloring C (the *same-color* and *other-color neighborhood* of v, respectively). Note that since we will use these notions in regard to a state of the process (a current coloring of G), we assume that the coloring function C is clear from the context, and thus it is not included in the above notation for simplicity.

In both weighted settings, we have a natural cost function f for each node v of the graph. In node-weighted graphs, we define $f(v) = \sum_{u \in N_S(v)} w(u)$, while in the edge-weighted setting, we define the cost function as $f(v) = \sum_{u \in N_S(v)} w(e(u, v))$. The aim of nodes in the minority process is to minimize this cost function. For a color $c \in \Gamma$, let $f_c(v)$ denote the cost that node v would have if it was recolored to color c, with the colors of all nodes in $N(v)$ remaining unchanged. Let us denote the preferred color of v by $c^* = \arg\min_c f_c(v)$; in case of multiple minimal values, we select an arbitrary one of them as c^*. When v *switches*, it changes its color to c^*. If $f(v) - f_{c^*}(v)$ is above a given threshold, or more generally, if the relation of $f(v)$ and $f_{c^*}(v)$ satisfies a specific condition known as the *switching rule*, then v is *switchable*.

A *minority process* on G is a sequence of colorings S_0, S_1, ..., known as *states*, where, except for S_0, each state S_i can be obtained from S_{i-1} by switching a set of nodes that are switchable in S_{i-1}. The state S_0 is referred to as the *initial state*. Given a graph and an initial state, the set of nodes to be switched in each step (and thus the entire sequence of states) is determined by the *model*, as discussed below.

We say that a state S_i is *stable* if there are no switchable nodes in S_i. A process *stabilizes* if it reaches a stable state; the number of steps until the process stabilizes is the *stabilization time* of the process.

While presenting our construction, we assume node-weighted graphs and $|\Gamma| = 2$ available colors. Section 4 discusses how to generalize our lower bound to edge-weighted graphs or more than 2 colors.

Models

We consider minority processes in the following models:

- **Sequential Adversarial (SA):** In each step, only one node switches. This node is chosen by an adversarial player, who aims to maximize the stabilization time.
- **Sequential Benevolent (SB):** In each step, only one node switches. This node is chosen by a benevolent player, who aims to minimize the stabilization time.
- **Concurrent Benevolent (CB):** In each step, the benevolent player can switch any set of switchable nodes concurrently, in order to minimize the stabilization time.

There are many further popular models of minority processes, for example, with synchronous or randomized behavior. However, these models always exhibit a larger stabilization time than model CB, since in model CB, the benevolent player is free to choose any sequence of (possibly concurrent) steps to minimize stabilization time, and thus he can also simulate the behavior of any of these additional models. Therefore, a lower bound for model CB also implies the same bound in these various other models.

Note that in concurrent models, it is possible that neighboring nodes repeatedly force each other to switch at the same step, cycling through the same colors infinitely. Because of this, related studies in the synchronous model often use an alternative definition of stabilization, also considering a periodically repeating process to be stable. However, the design of our benevolent construction ensures that connected nodes can never be switchable at the same time, and thus in our graphs, even in concurrent models, the process always terminates in a fixed state. Nonetheless, our lower bound also holds with this alternative, more permissive definition of stabilization.

Our lower bound construction for model SA is shown in Section 5. Then Section 6 describes how to extend this construction to the case of model SB. Once we present our construction for model SB, it will follow that this same construction also proves the lower bound in model CB. As the construction heavily restricts the set of selectable sequences,

always allowing only a few switchable nodes in the graph, even in model CB, the benevolent player has no other option than to execute exactly the same steps as in the sequential case, possibly some of them at the same time. On the other hand, the construction will have specific nodes that alone switch $2^{\Theta(n)}$ times, and thus even with some of the steps executed simultaneously, stabilization takes $2^{\Theta(n)}$ steps.

Switching rules

Most of the related work studies the following switching rule:

Rule I (*Basic Switching*): v is switchable if $W_{N_S(v)} - W_{N_O(v)} > 0$.

Here we introduce a stricter switching rule, based on a real parameter λ (where $0 < \lambda < 1$):

Rule II (*Proportional Switching*): v is switchable if $W_{N_S(v)} - W_{N_O(v)} \geq \lambda \cdot W_{N(v)}$.

This alternative switching condition is reasonable in many settings where switching comes with a certain cost for the node, and therefore, it is only beneficial when this allows the node to reduce its cost considerably, i.e. by a given factor of $W_{N(v)}$. Since we have $W_{N_S(v)} + W_{N_O(v)} = W_{N(v)}$ in the case of two colors, this condition is equivalent to $W_{N_S(v)} \geq \frac{1+\lambda}{2} \cdot W_{N(v)}$, i.e. that a node is only allowed to switch if $\frac{1+\lambda}{2}$ fraction of its (weighted) neighborhood has the same color. Therefore, if λ is close to 1, then Rule II intuitively means that in order to switch v twice, we also have to switch *almost every* neighbor of v in the meantime to make v switchable again for the second time.

While the above definition of Rule II is more intuitive, for the analysis, it is often convenient to express Rule II in another alternative form: v is switchable if $W_{N_S(v)} \geq \Lambda \cdot W_{N_O(v)}$, for some other constant Λ. One can show that this is equivalent to the definition with a choice of $\Lambda := \frac{1+\lambda}{1-\lambda}$. We will mostly use this alternative Λ parameter throughout our analysis.

Our technique proves the lower bound for Rule II with any $\lambda < 1$. However, for ease of presentation, we are first going to describe our construction for a specific parameter value of $\lambda \approx \frac{2}{3}$. Note that $\lambda = \frac{2}{3}$ corresponds to 5 in the Λ-notation; let us introduce the new notation $\Lambda_B := 5$ for this base value. We need this extra notation because the construction we present is actually not for $\Lambda = 5$, but in fact only for $\Lambda = 5 - \epsilon$ with any $\epsilon > 0$, hence proving the lower bound for Rule II with any $\Lambda < 5$ (or, using the λ-notation, for any $\lambda < \frac{2}{3}$). Note that we have specifically chosen $\lambda > \frac{1}{2}$ for demonstration because some challenges in the construction are easier if $\lambda \leq \frac{1}{2}$.

Given the proof of the lower bound for $\Lambda = 5 - \epsilon$ with any $\epsilon > 0$, we then discuss how to generalize the same construction technique for any other odd integer Λ_B as a base value. This proves the lower bound for $\Lambda = 7 - \epsilon$, $\Lambda = 9 - \epsilon$, and so on, with any $\epsilon > 0$.

Note that $\lim_{\Lambda_B \to \infty} \lambda = 1$, that is, as Λ_B goes to infinity, the λ value corresponding to $\Lambda_B - \epsilon$ gets arbitrarily close to 1 (this follows from the fact that λ can be expressed as $\frac{\Lambda-1}{\Lambda+1}$, by the definition of Λ). Therefore, we can obtain any $\lambda < 1$ value with an appropriate odd integer Λ_B and appropriate $\epsilon > 0$, and since our construction can be generalized for $\Lambda_B - \epsilon$ with any such Λ_B and ϵ, this already establishes the lower bound for every $\lambda \in (0, 1)$.

While it is not required for our lower bound proof, the full version of the paper also presents a general method to prove the monotonicity of the lower bound: that is, for any λ_0 and $\lambda < \lambda_0$ values, given a construction for λ_0, there is a straightforward way to convert it into a construction for λ. Note that this monotonicity is trivial in the adversarial case: since any node that is switchable for Rule II with λ_0 is also switchable for the rule with λ, the construction for λ_0 is, without any change, also a valid construction for λ, exhibiting the same stabilization time. The case is, however, not this simple for benevolent models, where a

lower λ value may allow a wider set of moves for the benevolent player, which might reduce the stabilization time significantly. Monotonicity in this model can be shown using so-called fixed nodes; see the full version for a discussion.

Helpful tools and definitions

We say that a node v is *dominated* by a subset $S \subseteq N(v)$ if $W_S \geq \frac{\Lambda}{\Lambda+1} \cdot W_{N(v)}$, that is, if S having the same color as v is enough to make v switchable. If v is dominated by a single-node subset $\{u\}$, then we say that v is a *follower node*, and u is the *dominant node* of v; this implies that the preferred color of v is always simply the opposite of u's color.

One tool we will frequently use in our constructions is the addition of so-called *fixed node neighbors*. A fixed node is a node that is added to the graph construction in a way that ensures it can never become switchable throughout the process, and thus always keeps its initial color. This can easily be achieved by adding a black and a white *stabilizer node* to the graph, and connecting each fixed node to the stabilizer of the opposite color. If we then assign significantly larger weights to the stabilizer nodes than to all other nodes in the graph (i.e., sufficiently large weights such that each fixed node is a followers of its (opposite-colored) stabilizer node neighbor), then the fixed nodes can indeed never switch throughout the process.

In our construction, each fixed node we add is only connected to one specific node v, and its only purpose is to influence the behavior of v in the process (i.e., make it easier or harder to switch v to a specific color). We may add a separate black and a white fixed node neighbor (with any desired weight) to every node v of the construction. However, note that it makes no sense to add more than two fixed neighbors to a node v: if we were to add two same-colored fixed neighbors to v, we could simply combine the two into one fixed neighbor with the sum of the two weights. Therefore, the use of fixed node neighbors adds at most $2n + 2$ extra nodes to the graph, only changing the magnitude of n by a constant factor, and thus it does not affect the exponential nature of stabilization time.

4 Basic Observations

Node or edge weights

We consider minority processes on both node-weighted and edge-weighted graphs. Note that edge weights have at least as much (in fact, more) expressive power than node weights: assume that we have a graph G with some node weights $w(v)$, and consider the edge-weighted graph that consist of the same nodes and edges, and edge weights are defined as $w(e(u, v)) = w(u) \cdot w(v)$. A minority process in this derived graph behaves the exact same way as in the original, node-weighted graph: for any node v, each neighbor $u \in N(v)$ stands for a $\frac{w(u)}{W_{N(v)}}$ portion of $W_{N(v)}$ in the node-weighted case, and u contributes exactly the same $\frac{w(u) \cdot w(v)}{W_{N(v)} \cdot w(v)}$ portion in the derived edge-weighted graph.

This implies that for any node-weighted graph, we can create a corresponding edge-weighted graph with the same stabilization time, regardless of the model. Therefore, when showing the lower bounds of the paper, we only consider node-weighted graph constructions. Our observations imply that the same lower bound will then also hold for edge-weighted graphs.

Number of colors

The constructions in the paper assume there are only two available colors: *black* and *white*. However, it is simple to generalize the lower bound to any number of colors. The main idea is to take the lower bound construction for 2 colors, and for each node of the graph and for every additional color, add an extra neighbor with high weight having this color. The process in the resulting graph will behave as if the graph only consisted of the original nodes and the original two colors. A detailed discussion of the technique is available in the full version of the paper. The method allows us to generalize the lower bound not only to any constant number of, but also up to $\Theta(n)$ colors.

Matching upper bound

While the proof of exponential lower bound is quite involved, it is straightforward to show an exponential upper bound on stabilization time in sequential models. To discuss this upper bound, we briefly return to the case of edge-weighted graphs, as they can exhibit a wider set of behaviors. Since for each node-weighted graph there exists an edge-weighted graph with the same stabilization time, the upper bound on edge-weighted graphs immediately implies the same upper bound on node-weighted graphs.

In an edge-weighted graph, for each state (i.e., coloring of the graph), we can define a *potential* value as the sum of $w(e)$ for all edges e in the graph that are currently conflicting. In sequential models when only one node switches in one step, this potential strictly decreases after every step, since the incentive of the nodes is exactly to reduce the potential in their neighborhood. This allows for a simple upper bound on stabilization time in sequential models: since each state has a fixed potential value and potential is monotonously decreasing throughout the process, each state can be visited at most once. For the case of 2 colors, there are 2^n distinct possible states, which implies that stabilization time is upper bounded by 2^n.

5 Construction for the Adversarial Case

We first present a graph construction to show the exponential lower bound in model SA.

▶ **Theorem 1.** *For Switching Rule II with any $\lambda < 1$, there exists a class of (sparse) weighted graphs with $2^{\Theta(n)}$ stabilization time in model SA.*

While the theorem holds for any $\lambda < 1$, recall that we present the construction for a concrete value of $\lambda \approx \frac{2}{3}$ (that is, $\Lambda = 5 - \epsilon$ for some small $\epsilon > 0$).

Throughout the presentation of our construction, nodes that are shown vertically higher in figures will always have larger weight than nodes that are placed below. Based on this, we also refer to neighbors of nodes as upper or lower neighbors. We will define the weight of each node in the graph as a function of the weights of the nodes below. As such, one can determine a concrete set of node weights for the construction by following these rules in a bottom-to-top fashion, with the lowermost weights chosen arbitrarily.

The basic idea behind our construction is recursive, and as such, the resulting graph consists of multiple *levels*. Given a construction that exhibits a sequence which switches some specific nodes of the graph s times at least, we show how to extend this graph with a constant number of new nodes (a next level) to obtain another construction where, with the correct choice of sequence, a specific new set of nodes switch $\frac{3}{2}s$ times. With a repeated application of this step, after adding ℓ levels, we obtain a set of nodes that switch $\left(\frac{3}{2}\right)^{\ell} \cdot s$ times. Since each new level consists of only $O(1)$ nodes, our graph can contain linearly many levels, yielding a final construction with $2^{\Theta(n)}$ switches.

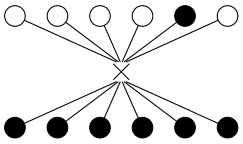


Figure 1 A 6-tuple of base nodes (below) and control nodes (above). The symbol × denotes a complete bipartite connection.

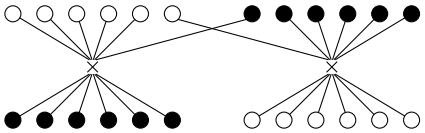


Figure 2 Final structure of a level, with two distinct 6-tuples of base and control nodes.

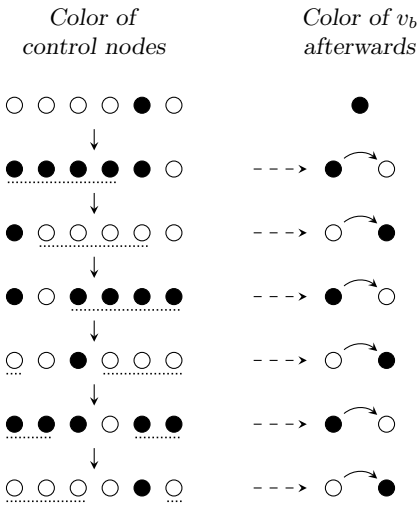


Figure 3 A control sequence of 6 steps, each time switching a 4-node subset of the control nodes (marked by a dotted line). The resulting switch of the base nodes is shown on the right.

The key nodes of our graph are the *base nodes*, which appear in 6-tuples with the same weight and same initial color. Each 6-tuple of base nodes has 6 common upper neighbors, known as the *control nodes* for these base nodes, forming a complete bipartite graph. The two 6-tuples together comprise a level of our construction (see Figure 1).

The 6 control nodes in a level also all have the same weight; let us denote this weight by $w(v_c)$. The main idea of the construction is to choose $w(v_c)$ sufficiently large such that 5 of the 6 control nodes already dominate each of the base nodes below. Assuming that one of the base node v_b has further (lower) neighbors of weight w_L altogether, this requires $5 \cdot w(v_c) \geq \Lambda \cdot (w(v_c) + w_L)$ to hold, which can be ensured by a choice of $w(v_c) \geq \frac{5-\epsilon}{\epsilon} \cdot w_L$ for our current $\Lambda = 5 - \epsilon$. Thus we can select sufficiently large weights such that a base node v_b is indeed switchable whenever 5 out of 6 control nodes have the same color as v_b.

Note that from the initial state shown in Figure 1, we only need to switch 4 of the 6 control nodes (from white to black) in order to force a base node v_b below to switch to white. In fact, we can specify a sequence of 4-node subsets of the control nodes such that every time we switch the next subset in the sequence, we once again have 5 control nodes with the same color that v_b currently has, and therefore v_b can be switched again. A possible such sequence is shown in Figure 3; we refer to this as the *control sequence*. The sequence has a couple of convenient properties: each control node is switched exactly 4 times throughout the sequence, and each control node (and also v_b) returns to its initial color at the end of the sequence.

This is exactly the technique that allows us to increase the number of switches by a factor of $\frac{3}{2}$ within each level of the construction. If the upper levels provide a way to switch each of the 6 control nodes in the current level s times, then this allows us to execute the control sequence $\frac{s}{4}$ times, and each such execution switches the base nodes in the current level 6 times, adding up to $\frac{6}{4}s$ switches for each of the 6 base nodes.

It only remains to connect the different levels of our recursive construction. It comes as a natural first idea that the 6-tuple of base nodes in a level could also directly take the role of the control nodes in the level below. The first difficulty to overcome with this approach

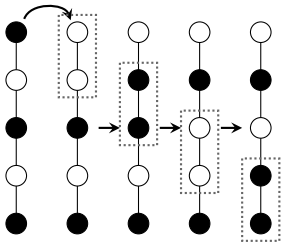


Figure 4 When a conflict is created at the top of the chain, then switching the nodes one by one propagates this conflict down through the chain.

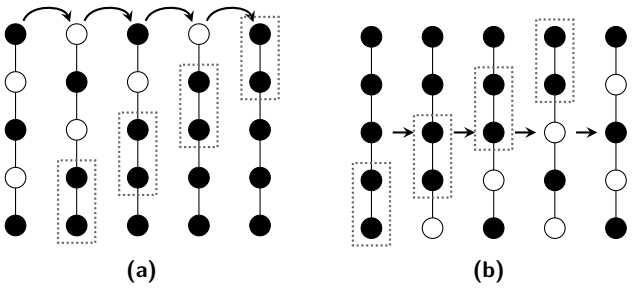


Figure 5 When charging (a), we propagate each new conflict to the next position (Figure 4 shows the first step of (a) in detail). When unloading (b), we always propagate the lowermost stored conflict to the bottom.

is the color of the nodes in question: while all 6 base nodes of a level have the same color (say, initially black), the control nodes initially have mixed color (5 white and 1 black) in the control sequence. We can overcome this by duplicating the structure in Figure 1 in the opposite initial color, and redefining a level as these two bipartite graphs together. Since a level now consists of 12 base nodes, 6 white and 6 black initially, we can reorganize these nodes into two appropriate groups (5 white + 1 black, 5 black + 1 white) to act as the control nodes of the next level (see Figure 2).

There is a further problem with using the base nodes directly as the control nodes of the level below: our level design only provides a way to switch a 6-tuple of base nodes *together* (that is, consecutively in any order). However, in order to execute the control sequence, we need to be able to switch specific subsets of the control nodes. For example, in the sequence of Figure 3, the second node from the left has already switched twice before the rightmost node ever switches. Thus, the fact that we can switch both 6-tuples of base nodes s times does not yet imply that we can switch specific 4-node subsets of them in the given order, as needed for the control sequence.

To provide a way to switch the control nodes in any order of our choice, we connect the levels of the construction with tools known as *storage chains*. A storage chain is a path of 5 nodes, initially colored in an alternating fashion. The weights of the nodes in the chain are chosen such that each node is a follower node of its upper neighbor (this can be ensured by defining node weights in a bottom-to-top fashion, always choosing sufficiently large weight for the next node). The uppermost and lowermost nodes may have other upper and lower neighbors outside of the chain, respectively.

Assume now that the topmost node in the chain is switched by some external condition (i.e., its upper neighbors outside of the chain). This introduces a conflict into the chain between the uppermost two nodes, as shown in Figure 4. However, recall that by our definition of node weights, the second node (from the top) is a follower of the uppermost node, and therefore this conflict makes the second node switchable. Switching the second node (to black) resolves the original conflict, but creates a new conflict between the second and third nodes instead (now making the third node switchable). Generally, whenever there is a conflicting pair of subsequent nodes above an alternating-colored (part of the) chain, we are able to switch the lower node, and thus move the conflict down to the next node pair in the chain. We can use this method to move a conflict down to any point in the chain, as shown in the figure; we refer to this process as *propagating down* the conflict in the chain.

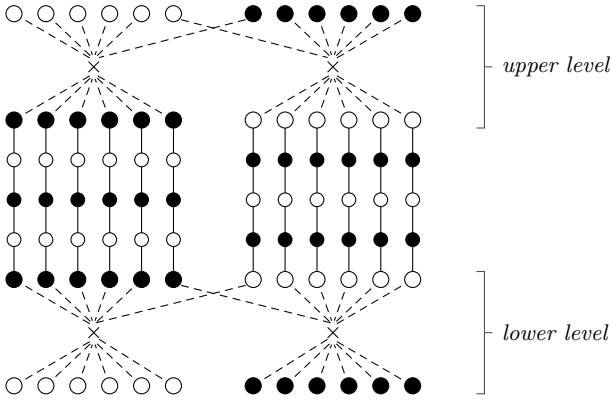


Figure 6 Two levels of the construction, connected by storage chains (edges within a level are shown in dashed). For simpler illustration, the two sides of the lower level are horizontally swapped.

With this technique, we can accumulate and store conflicts in the chain "for later use". If the uppermost node is forced to switch 4 times, then we can propagate down each of the emerging conflicts to a different position (i.e., pair of nodes) in the chain, ending up with 4 conflicts in a completely monochromatic chain. This process (see Figure 5a) is referred to as *charging* the chain. In another sequence of steps, we can then *unload* the chain and propagate these conflicts one by one to the bottom of the chain, essentially using the stored conflicts to switch the lowermost node 4 times in a timing of our own choice (see Figure 5b). When the sequence is finished, each node in the chain once again has its original color.

We use such storage chains to connect subsequent levels of our construction, with the base nodes and control nodes being the uppermost and lowermost nodes in the chains, respectively, as shown in Figure 6. This way, every time after the 6-tuple of base nodes in the upper level switch (together), we can execute the next step in charging each of the storage chains. After each of the base nodes switch 4 times, each of the storage chains are charged. Then, by unloading each chain in 4 steps in the order of our choice, we can switch each of the control nodes below 4 times, in any preferred order; this enables us to execute the control sequence on the lower level. Thus, if the upper-level base nodes are switched 4 times, we can indeed switch the lower-level base nodes 6 times.

For a high-level overview of the process, the execution of the adversarial sequence on a given level L could be summarized by the following recursive pseudocode:

Function ProcessLevel(L)
  For each of the 6 steps of the control sequence:
    On both sides, switch the next subset of 4 control nodes
    Switch all $6 + 6$ base nodes
    Propagate down the conflict in each chain as far as possible
    If the chains below are fully charged:
      Call ProcessLevel($L + 1$) (*execution continues on level below*)
  Return (*execution continues on level above*)

Even with the storage chain connections, the addition of each new level increases the number of nodes only by a constant value. This implies that a graph on n nodes can contain $\Theta(n)$ levels, and thus each node in the lowermost level indeed switches $2^{\Theta(n)}$ times.

There is one more detail to discuss: for convenience, we assumed that the number of switches s in an upper level is always divisible by 4. However, s switches in each control node in fact allows for only $\lfloor \frac{s}{4} \rfloor$ complete executions of the control sequence, and hence $\lfloor \frac{s}{4} \rfloor \cdot 6$ switches for the base nodes. Nonetheless, this still implies exponential increase for s large enough (for example, $\lfloor \frac{s}{4} \rfloor \cdot 6 \geq \frac{6}{5}s$ holds if $s \geq 20$). Thus to overcome this problem, we ensure that the control nodes in the uppermost level already switch 20 times; this is achieved by adding an initially charged storage chain of 21 nodes above each uppermost control node. Unloading the chains allows us to switch these top-level control nodes 20 times in the preferred order, and thus the exponential increase of switches is guaranteed.

This proves our lower bound in model SA for the case of Rule II with $\Lambda = 5 - \epsilon$ for any $\epsilon > 0$. However, the construction is straightforward to generalize to any other odd integer Λ_B: for most of the analysis, one only needs to replace the value 4 by $(\Lambda_B - 1)$ and the value 6 by $(\Lambda_B + 1)$. This provides a construction with $(\Lambda_B + 1)$-tuples of control and base nodes, and a $\frac{\Lambda_B+1}{\Lambda_B-1}$ factor of increase in switches for every new level. The control sequence can also be generalized for other Λ_B values; details of the generalization are discussed in the full version of the paper.

6 Benevolent Case

It is significantly more difficult to show an exponential lower bound for benevolent models, since such a construction needs to guarantee that every possible sequence lasts for an exponential number of steps. We overcome this problem by heavily restricting the set of selectable sequences in the graph. Specifically, we start from the construction of Section 5, and we show how to add a set of extra nodes which ensure that the previously defined sequence is the only possible sequence the benevolent player can choose. In this section, we outline the main ideas of this benevolent construction; a detailed discussion of the technique is provided in the full version of the paper.

▶ **Theorem 2.** *For Switching Rule II with any $\lambda < 1$, there exists a class of (sparse) weighted graphs that have $2^{\Theta(n)}$ stabilization time in the benevolent models (models SB and CB).*

We basically use two tools (gadgets) to ensure that the player, when selecting the sequence, has to follow the procedure described in the pseudocode above. On the one hand, we show how to build logical AND *gates* and OR *gates*, in order to check that a given step of the procedure is reached, and use these gates to allow the player to proceed to the next step of the procedure. On the other hand, we devise a *state chain* in order to keep track of the current phase of the procedure, which can then be used as a condition in the logical gates that control the execution of the procedure.

With the appropriate combination of these two gadgets, we can ensure that the benevolent player has no other option than to switch the control nodes, base nodes and storage chain nodes in the order described by the recursive procedure. We add a separate such combination of these gadgets to each level of the construction of Section 5. However, since in our recursive procedure, each level of the graph executes the same sequence of steps multiple times (the lower levels exponentially many times), the design of these gadgets also needs to ensure that the gadget can execute its task multiple times. This is achieved through introducing a method to repeatedly "reset" the gadgets to their initial state.

For the purpose of resetting these gadgets, we introduce another tool, the third main ingredient of our benevolent construction, known as a *pacer system*. The main idea of the resetting technique is to connect each gadget (logical gate or state chain) to so-called *pacer*

nodes higher in the graph, and to ensure that each such pacer node switches at least twice between two consecutive times of using the gadget. The gadgets are designed in a way which guarantees that this pacer node switching twice results in the gadget being reset to its default state (i.e., each node to its initial color).

Such a pacer node essentially "recharges" the gadget with conflicts: since the weighted sum of conflicts in the graph monotonically decreases, the gadget can only return to the same (initial) state repeatedly if it "acquires" new conflicts from some other part of the graph. This is achieved through the connection to the pacer node, which is in a higher level of the graph (with larger weights), and thus has significantly more conflicts to "push down" into the gadget as a byproduct of its switching.

The simplest way to add pacer nodes to our construction is to place a pair of them between a set of control and base nodes, as shown in Figure 7. In this modified level version, the steps of the control sequence do not switch the base nodes directly. Instead, this happens indirectly: after 5 of the 6 control nodes are black, first the upper pacer node, and then the lower pacer node switches, followed by the base nodes in the end. Thus, the addition of pacer nodes leaves the general behavior of the level unchanged: the base nodes will still switch eventually after each step of the control sequence. However, in this new level construction, the newly added pacer nodes will also both switch in each of these steps.

The actual pacer systems used in our construction are more sophisticated constructions based on this idea. They consist of multiple pacer nodes in order to be able to recharge gadgets of both colors, and they are also responsible for checking that the recharging process has indeed been executed on the connected gadgets.

Given the technique to reset gadgets, it only remains to briefly present the behavior of the two gadgets (logical gates and state chains), and to outline how they are used in the construction. For the convenient description of gadgets, we first introduce two special kinds of node concepts. Essentially, these are methods to carefully select the weight of some specific neighbors of nodes such that they fulfill the following roles:

- *Observer node*: given a set of nodes U_0, we can add a new common neighbor v_o to these nodes such that the behavior of v_o depends on the nodes in U_0, but the behavior of U_0 is unaffected by the addition of v_o
- *Enabler node*: given a node u_1 dominated by another node u_d, we can add a new neighbor v_e to u_1, such that u_1 is no longer dominated by $\{u_d\}$, but it is dominated by the subset $\{u_d, v_e\}$

Given a set of input nodes U_0 and an output node u_1, we can use these concepts to build an AND gate which only enables the switching of u_1 if all nodes in U_0 are colored with a given color. This gadget connects to each of the input nodes in U_0 through a common observer node, and connects to the output u_1 through an enabler node. Besides the observer and enabler node, the gadget only requires an extra relay node (and an appropriate choice of weights) to connect these two nodes, and an extra upper neighbor for each node in order to connect the gadget to a pacer system which resets it after use. A brief illustration of the gadget is available in Figure 8. In a very similar fashion, we can also create AND gates for inputs of the other color, OR gates, or even multi-layer gates that allow us to combine different conditions.

Besides logical gates, the other key gadget in our benevolent construction is the state chain. For each level of the construction, we add a separate state chain in order to indicate the current state (i.e., point in the execution) of the procedure on this level. Essentially, a state chain is a vertical chain of nodes, where every node in the chain is dominated by its upper neighbor, similarly to the case of a storage chain. However, while storage chains are

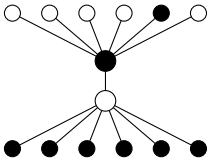


Figure 7 Adding a pair of pacers between a layer of control nodes and base nodes.

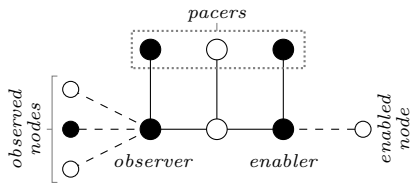


Figure 8 Logical (e.g. AND) gate.

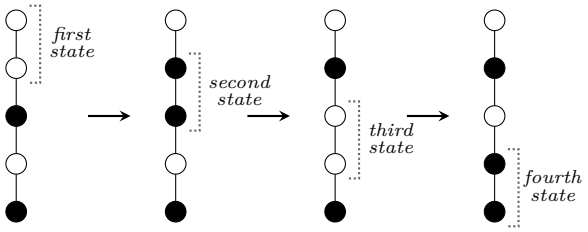


Figure 9 Simplified illustration of a state chain on 4 states (see the full version of the paper for a more detailed illustration). The position of the conflict in the chain shows our current state of the procedure. When propagating the conflict down by one step, the chain proceeds to the next state.

used to accumulate conflicts, a state chain will, on the other hand, always contain exactly one conflict, which we propagate down step by step. The different possible positions of the conflict can then correspond to different states of the procedure, and at a given point in time, our current state in the procedure is determined by the current position of the conflict in the chain (as illustrated in Figure 9).

One such state chain is added to each level of our benevolent construction. The node pairs in the chain that express a state are included in the conditions of the logical gates that control the execution of the recursive procedure on the level, ensuring that certain steps are only available to the benevolent player at certain points in the process. Furthermore, the nodes in the state chain are also connected to enabler nodes, and thus proceeding to the next state is always based on a given condition. Therefore, the benevolent player has no other option than to simultaneously proceed through the steps of the recursive process and the states of the state chain, in the appropriate order. With a couple of auxiliary nodes at the top of the chain, we can also connect the state chain to a pacer node, allowing us to reset the chain and jump back to the first state whenever the last state of the chain is reached.

Given these gadgets, let us now briefly reflect on the states and conditions we need to encode in order to ensure that the player has to follow the recursive sequence. The main idea is to use the logical gates to control the flow of execution within a given level: through the enabler nodes of the gates, we ensure that the switching of the next 2×4 control nodes (i.e., the next step of the control sequence) is only enabled after the previous switching of the base nodes is finished. In practice, this means that, after the base nodes have switched, when the newly added conflicts are propagated down far enough in each of the 2×6 storage chains below, the gates enable the further down-propagation of the appropriate 2×4 conflicts in the storage chains above, which will in turn make the next subset of 2×4 control nodes switchable. That is, the input (observer) nodes of these logical gates are connected to specific nodes of the storage chains below the level, while their output (enabler) nodes are connected to nodes of the storage chains above the level.

However, recall that charging the storage chains below takes 4 steps, while executing the control sequence above consist of 6 steps, so the two processes do not remain in synchrony. Thus in different phases of the procedure, the same set of storage chain nodes below have to enable different subsets of the control nodes above. Because of this, our construction encodes these different phases of the procedure as states in the state chain, and the appropriate state is also included in the condition of the logical gate that enables the next set of control nodes. When a cycle is finished (i.e., the two processes return to their default state at the same time), the state chain is reset and iteration starts again from the first state of the chain.

Furthermore, note that throughout the recursion, execution repeatedly leaves the current level and continues on the level above (or below), so the state chain of each level also has specific states indicating that the execution is currently on a level above (or below).

Altogether, these benevolent-case modifications only add constantly many gadgets (each of constant-size) to each level of the construction. Therefore, the modified construction still has only $O(1)$ nodes in a level, allowing for $\Theta(n)$ levels and thus $2^{\Theta(n)}$ stabilization time. This establishes our lower bound for model SB. By design, the construction only has a few (at most constantly many) switchable nodes at every point in time, and thus even in model CB, it allows for only very limited concurrency for the benevolent player. Specifically, since there are concrete nodes in the construction that switch $2^{\Theta(n)}$ times, the number of steps is still exponential in model CB.

Also, note that even with the gadgets added in the benevolent case, each node of the graph still has a constant degree, and thus our bound is also valid for sparse graphs.

References

1 Ron Aharoni, Eric C Milner, and Karel Prikry. Unfriendly partitions of a graph. *Journal of Combinatorial Theory, Series B*, 50(1):1–10, 1990.

2 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. On the existence and determination of satisfactory partitions in a graph. In *International Symposium on Algorithms and Computation*, pages 444–453. Springer, 2003.

3 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Complexity and approximation of satisfactory partition problems. In *International Computing and Combinatorics Conference*, pages 829–838. Springer, 2005.

4 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. The satisfactory partition problem. *Discrete applied mathematics*, 154(8):1236–1245, 2006.

5 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *European Journal of Operational Research*, 206(2):271–280, 2010.

6 Olivier Bodini, Thomas Fernique, and Damien Regnault. Quasicrystallization by stochastic flips. *HAL online archives*, 2009.

7 Olivier Bodini, Thomas Fernique, and Damien Regnault. Stochastic flips on two-letter words. In *2010 Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 48–55. SIAM, 2010.

8 Henning Bruhn, Reinhard Diestel, Agelos Georgakopoulos, and Philipp Sprüssel. Every rayless graph has an unfriendly partition. *Combinatorica*, 30(5):521–532, 2010.

9 Zhigang Cao and Xiaoguang Yang. The fashion game: Network extension of matching pennies. *Theoretical Computer Science*, 540:169–181, 2014.

10 Jacques Demongeot, Julio Aracena, Florence Thuderoz, Thierry-Pascal Baum, and Olivier Cohen. Genetic regulation networks: circuits, regulons and attractors. *Comptes Rendus Biologies*, 326(2):171–188, 2003.

11 Silvio Frischknecht, Barbara Keller, and Roger Wattenhofer. Convergence in (social) influence networks. In *International Symposium on Distributed Computing*, pages 433–446. Springer, 2013.

12 Michael U Gerber and Daniel Kobler. Algorithmic approach to the satisfactory graph partitioning problem. *European Journal of Operational Research*, 125(2):283–291, 2000.

13 Michael U Gerber and Daniel Kobler. Classes of graphs that can be partitioned to satisfy all their vertices. *Australasian Journal of Combinatorics*, 29:201–214, 2004.

14 Sandra M Hedetniemi, Stephen T Hedetniemi, KE Kennedy, and Alice A Mcrae. Self-stabilizing algorithms for unfriendly partitions into two disjoint dominating sets. *Parallel Processing Letters*, 23(01):1350001, 2013.

15 Barbara Keller, David Peleg, and Roger Wattenhofer. How Even Tiny Influence Can Have a Big Impact! In *International Conference on Fun with Algorithms*, pages 252–263. Springer, 2014.

16 Jeremy Kun, Brian Powers, and Lev Reyzin. Anti-coordination games and stable graph colorings. In *International Symposium on Algorithmic Game Theory*, pages 122–133. Springer, 2013.

17 Damien Regnault, Nicolas Schabanel, and Éric Thierry. Progresses in the Analysis of Stochastic 2D Cellular Automata: A Study of Asynchronous 2D Minority. In Luděk Kučera and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2007*, pages 320–332. Springer Berlin Heidelberg, 2007.

18 Damien Regnault, Nicolas Schabanel, and Éric Thierry. On the analysis of "simple" 2d stochastic cellular automata. In *International Conference on Language and Automata Theory and Applications*, pages 452–463. Springer, 2008.

19 Jean-Baptiste Rouquier, Damien Regnault, and Éric Thierry. Stochastic minority on graphs. *Theoretical Computer Science*, 412(30):3947–3963, 2011.

20 Khurram H Shafique and Ronald D Dutton. On satisfactory partitioning of graphs. *Congressus Numerantium*, pages 183–194, 2002.

21 Saharon Shelah and Eric C Milner. Graphs with no unfriendly partitions. *A tribute to Paul Erdös*, pages 373–384, 1990.

Finite Sequentiality of Unambiguous Max-Plus Tree Automata

Erik Paul
Institute of Computer Science, Leipzig University, 04109 Leipzig, Germany
epaul@informatik.uni-leipzig.de

Abstract

We show the decidability of the finite sequentiality problem for unambiguous max-plus tree automata. A max-plus tree automaton is called unambiguous if there is at most one accepting run on every tree. The finite sequentiality problem asks whether for a given max-plus tree automaton, there exist finitely many deterministic max-plus tree automata whose pointwise maximum is equivalent to the given automaton.

2012 ACM Subject Classification Theory of computation → Quantitative automata; Theory of computation → Tree languages

Keywords and phrases Weighted Tree Automata, Max-Plus Tree Automata, Finite Sequentiality, Decidability, Ambiguity

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.55

Funding This work was supported by Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg 1763 (QuantLA).

1 Introduction

A max-plus automaton is a finite automaton which assigns real numbers to words over a given alphabet. The transitions of a max-plus automaton each carry a weight from the real numbers. To every run of the automaton, a weight is associated by summing over the weights of the transitions which constitute the run. The weight of a word is given by the maximum over the weights of all runs on this word.

More generally, max-plus automata and their min-plus counterparts are weighted automata [31, 30, 22, 5, 11] over the max-plus or min-plus semiring. Min-plus automata were originally introduced by Imre Simon as a means to show the decidability of the *finite power property* [34, 35]. Since their introduction, max-plus and min-plus automata enjoy a continuing interest [21, 14, 18, 10, 12, 6] and they have been employed in many different contexts. To only name some examples, they can be used to determine the star height of a language [13], to prove the termination of some string rewriting systems [36], and to model certain discrete event systems [19]. Additionally, they appear in the context of natural language processing [24], where for reasons of numerical stability, probabilities are often computed in the min-plus semiring as negative log-likelihoods.

A very prominent open question about max-plus automata is the *sequentiality problem*, the problem of deciding whether for an arbitrary max-plus automaton there exists a deterministic equivalent. A max-plus automaton is called *deterministic* or *sequential* if for each pair of a state and an input symbol, there is at most one valid transition into a next state. Although the decidability of this problem is unknown for max-plus automata in general, it is known to be decidable for the subclasses of *unambiguous* [24], *finitely ambiguous* [18], and even *polynomially ambiguous* [17] automata. A max-plus automaton is called *unambiguous* if there exists at most one accepting run on every word. It is called *finitely ambiguous* if the number of runs on each word is bounded by a global constant. If on every word the number of accepting runs is bounded polynomially in the length of the word, the automaton is said to

be *polynomially ambiguous*. Note that the ambiguity of a max-plus automaton is a decidable property, as it is easily reduced to deciding the ambiguity of a finite automaton. Deciding the sequentiality of a finite automaton is trivial, polynomial time algorithms for deciding the unambiguity, the finite ambiguity, and the polynomial ambiguity of a finite automaton can be found in [7, 37, 33]. Furthermore, the classes of functions definable by deterministic, unambiguous, finitely ambiguous, polynomially ambiguous, and arbitrary max-plus automata form a strictly ascending hierarchy [18, 15, 23].

A decidability problem which is closely related to the sequentiality problem is the *finite sequentiality problem*. The finite sequentiality problem asks whether a given max-plus automaton can be represented as a pointwise maximum of deterministic max-plus automata. In [14], it was left as an open question to determine the decidability of the finite sequentiality problem for finitely ambiguous max-plus automata. It was shown only recently that for the classes of unambiguous as well as finitely ambiguous automata, the finite sequentiality problem is decidable [3, 2]. The class of functions which allow a finitely sequential representation by max-plus automata lies strictly between the classes of functions definable by deterministic and by finitely ambiguous max-plus automata, and it is incomparable to the class of functions definable by unambiguous max-plus automata [18].

In this paper, we show that the finite sequentiality problem is decidable for unambiguous max-plus tree automata. Max-plus tree automata are a generalization of max-plus automata and operate on trees instead of words. Applications for max-plus tree automata include proving the termination of certain term rewriting systems [20], and they are also commonly employed in natural language processing [27] in the form of *probabilistic context-free grammars*. Our approach to show the decidability of the finite sequentiality problem employs ideas from [3]. In [3], the *fork property* is shown to be a decidable criterion to determine the existence of a finitely sequential equivalent. More precisely, a max-plus word automaton is shown to possess a finitely sequential representation if and only if it does not satisfy the fork property. It is shown elementarily that an automaton satisfying the fork property cannot possess a finitely sequential equivalent. The proof for the existence of a finitely sequential representation in case that the fork property is not satisfied, on the other hand, relies on the construction of finitely many unambiguous max-plus automata whose pointwise maximum is equivalent to the original automaton, and which all satisfy the *twins property*. It was shown by Mohri [24] that an unambiguous max-plus automaton which satisfies the twins property is determinizable. A finitely sequential representation is thus found by determinizing the unambiguous automata.

For tree automata, we generalize the fork property to the *tree fork property* by adding a condition which accounts for the nonlinear structure of trees. We then prove that an unambiguous max-plus tree automaton possesses a finitely sequential representation if and only if it does not satisfy the tree fork property. As in the word case, the most challenging part of the proof is to show the existence of a finitely sequential representation whenever the tree fork property is not satisfied. Like in the proof for word automata, we construct finitely many unambiguous max-plus tree automata which possess a deterministic equivalent. However, we need to take a different approach in order to obtain these automata. In [3], a modified *Schützenberger covering* [32, 29] is first constructed from the unambiguous max-plus automaton, from which in turn an automaton is constructed which monitors the occurrence of certain states of the modified Schützenberger covering. This latter automaton is then decomposed into the finitely many unambiguous automata. This approach, however, is not applicable to trees, as the monitoring of states requires all relevant states to occur linearly. This happens trivially for word automata due to the inherent linear structure of words, but for

tree automata examples can be found where relevant states occur nonlinearly. The approach we use here relies on constructing a max-plus automaton which tracks certain pairs of states of the original automaton. When applied to word automata, this immediately yields an automaton which can be decomposed into the desired unambiguous automata. Unfortunately, for tree automata this tracking of pairs of states again fails due to states occurring nonlinearly. Surprisingly however, our construction can be applied to the Schützenberger covering of the original tree automaton, as the states relevant for tracking all occur pairwise linearly in the Schützenberger covering. The most difficult part of our proof is to show that the Schützenberger covering indeed has the property we just indicated.

2 Preliminaries

For a set X, we denote the power set of X by $\mathcal{P}(X)$ and the cardinality of X by $|X|$. For two sets X and Y and a mapping $f\colon X \to Y$, we call X the *domain* of f, denoted by $\mathrm{dom}(f)$, and Y the *range* of f, denoted by $\mathrm{range}(f)$. For a subset $X' \subseteq X$, we call the set $f(X') = \{y \in Y \mid \exists x \in X'\colon f(x) = y\}$ the *image* or *range of X' under f*. For an element $y \in Y$, we call the set $f^{-1}(y) = \{x \in X \mid f(x) = y\}$ the *preimage of y under f*. For a second mapping $g\colon X \to Y$, we write $f = g$ if for all $x \in X$ we have $f(x) = g(x)$.

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. By $\mathbb{N}^*$ we denote the set of all finite words over $\mathbb{N}$. The empty word is denoted by ε, and the length of a word $w \in \mathbb{N}^*$ by $|w|$. The set $\mathbb{N}^*$ is partially ordered by the prefix relation $\leq_p$ and totally ordered with respect to the lexicographic ordering $\leq_l$. Two words from $\mathbb{N}^*$ are called *prefix-dependent* if they are in prefix relation, and otherwise they are called *prefix-independent*.

A *ranked alphabet* is a pair $(\Gamma, \mathrm{rk}_\Gamma)$, often abbreviated by Γ, where Γ is a finite set and $\mathrm{rk}_\Gamma\colon \Gamma \to \mathbb{N}$ a mapping which assigns a rank to every symbol. For every $m \geq 0$ we define $\Gamma^{(m)} = \mathrm{rk}_\Gamma^{-1}(m)$ as the set of all symbols of rank m. The rank of Γ is defined as $\mathrm{rk}(\Gamma) = \max\{\mathrm{rk}_\Gamma(a) \mid a \in \Gamma\}$. The set of (*finite, labeled, and ordered*) Γ-*trees*, denoted by T_Γ, is the set of all pairs $t = (\mathrm{pos}(t), \mathrm{label}_t)$, where $\mathrm{pos}(t) \subset \mathbb{N}^*$ is a finite non-empty prefix-closed set of *positions*, $\mathrm{label}_t\colon \mathrm{pos}(t) \to \Gamma$ is a mapping, and for every $w \in \mathrm{pos}(t)$ we have $wi \in \mathrm{pos}(t)$ iff $1 \leq i \leq \mathrm{rk}_\Gamma(\mathrm{label}_t(w))$. We write $t(w)$ for $\mathrm{label}_t(w)$ and $|t|$ for $|\mathrm{pos}(t)|$. We also refer to the elements of $\mathrm{pos}(t)$ as *nodes*, to ε as the *root* of t, and to prefix-maximal nodes as *leaves*. The *height* of t is defined as $\mathrm{height}(t) = \max_{w \in \mathrm{pos}(t)} |w|$. For a leaf $w \in \mathrm{pos}(t)$, the set $\{v \in \mathrm{pos}(t) \mid v \leq_p w\}$ is called a *branch* of t.

Now let $s, t \in T_\Gamma$ and $w \in \mathrm{pos}(t)$. The *subtree of t at w*, denoted by $t{\restriction_w}$, is a Γ-tree defined as follows. We let $\mathrm{pos}(t{\restriction_w}) = \{v \in \mathbb{N}^* \mid wv \in \mathrm{pos}(t)\}$ and for $v \in \mathrm{pos}(t{\restriction_w})$, we let $\mathrm{label}_{t{\restriction_w}}(v) = t(wv)$. The *substitution of s into w of t*, denoted by $t\langle s \to w\rangle$, is a Γ-tree defined as follows. We let $\mathrm{pos}(t\langle s \to w\rangle) = \{v \in \mathrm{pos}(t) \mid w \not\leq_p v\} \cup \{wv \mid v \in \mathrm{pos}(s)\}$. For $v \in \mathrm{pos}(t\langle s \to w\rangle)$, we let $\mathrm{label}_{t\langle s \to w\rangle}(v) = s(u)$ if $v = wu$ for some $u \in \mathrm{pos}(s)$, and otherwise $\mathrm{label}_{t\langle s \to w\rangle}(v) = t(v)$.

For $a \in \Gamma^{(m)}$ and trees $t_1, \ldots, t_m \in T_\Gamma$, we also write $a(t_1, \ldots, t_m)$ to denote the tree t with $\mathrm{pos}(t) = \{\varepsilon\} \cup \{iw \mid i \in \{1, \ldots, m\}, w \in \mathrm{pos}(t_i)\}$, $\mathrm{label}_t(\varepsilon) = a$, and $\mathrm{label}_t(iw) = t_i(w)$.

For a ranked alphabet Γ, a tree over the alphabet $\Gamma_\diamond = (\Gamma \cup \{\diamond\}, \mathrm{rk}_\Gamma \cup \{\diamond \mapsto 0\})$ is called a Γ-*context*. Let $t \in T_{\Gamma_\diamond}$ be a Γ-context and let $w_1, \ldots, w_n \in \mathrm{pos}(t)$ be a lexicographically ordered enumeration of all leaves of t labeled $\diamond$. Then we call t an n-Γ-*context* and define $\Diamond_i(t) = w_i$ for $i \in \{1, \ldots, n\}$. For an n-Γ-context t and contexts $t_1, \ldots, t_n \in T_{\Gamma_\diamond}$, we define $t(t_1, \ldots, t_n) = t\langle t_1 \to \Diamond_1(t)\rangle \ldots \langle t_n \to \Diamond_n(t)\rangle$ by substitution of $t_1, \ldots, t_n$ into the $\diamond$-leaves of t. A 1-Γ-context is also called a Γ-*word*. For a Γ-word s, we define $s^1 = s$ and $s^{n+1} = s(s^n)$ for $n \geq 1$.

A *commutative semiring* is a tuple $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$, abbreviated by K, with operations sum $\oplus$ and product $\odot$ and constants $\mathbb{0}$ and $\mathbb{1}$ such that $(K, \oplus, \mathbb{0})$ and $(K, \odot, \mathbb{1})$ are commutative monoids, multiplication distributes over addition, and $\kappa \odot \mathbb{0} = \mathbb{0} \odot \kappa = \mathbb{0}$ for every $\kappa \in K$. In this paper, we only consider the *max-plus semiring* $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ where the sum and the product operations are max and $+$, respectively, extended to $\mathbb{R} \cup \{-\infty\}$ in the usual way.

A *max-plus weighted bottom-up finite state tree automaton (short: max-plus-WTA) over* Γ is a tuple $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ where Q is a finite set (of states), Γ is a ranked alphabet (of input symbols), $\mu\colon \bigcup_{m=0}^{\mathrm{rk}(\Gamma)} Q^m \times \Gamma^{(m)} \times Q \to \mathbb{R}_{\max}$ (the function of transition weights), and $\nu\colon Q \to \mathbb{R}_{\max}$ (the function of final weights). We define $\Delta_\mathcal{A} = \mathrm{dom}(\mu)$. A tuple $(\bar{p}, a, q) \in \Delta_\mathcal{A}$ is called a *transition* and $(\bar{p}, a, q)$ is called *valid* if $\mu(\bar{p}, a, q) \neq -\infty$. A state $q \in Q$ is called *final* if $\nu(q) \neq -\infty$.

For a tree $t \in T_\Gamma$, a mapping $r\colon \mathrm{pos}(t) \to Q$ is called a *quasi-run of* $\mathcal{A}$ *on* t. For a quasi-run r on t and a position $w \in \mathrm{pos}(t)$ with $t(w) = a \in \Gamma^{(m)}$, the tuple $\mathbb{t}(t, r, w) = (r(w1), \ldots, r(wm), a, r(w))$ is called the *transition at* w. The quasi-run r is called a *(valid) run* if for every $w \in \mathrm{pos}(t)$ the transition $\mathbb{t}(t, r, w)$ is valid with respect to $\mathcal{A}$. We call a run r *accepting* if $r(\varepsilon)$ is final. By $\mathrm{Run}_\mathcal{A}(t)$ and $\mathrm{Acc}_\mathcal{A}(t)$ we denote the sets of all runs and all accepting runs of $\mathcal{A}$ on t, respectively. Similar to trees, we define restrictions of runs as follows. Let $t \in T_\Gamma$, $r \in \mathrm{Run}_\mathcal{A}(t)$, and $w \in \mathrm{pos}(t)$. We define $r\restriction_w \in \mathrm{Run}_\mathcal{A}(t\restriction_w)$ by $r\restriction_w(v) = r(wv)$ for $v \in \mathrm{pos}(t\restriction_w)$.

For $r \in \mathrm{Run}_\mathcal{A}(t)$, the *weight of* r is defined by $\mathrm{wt}_\mathcal{A}(t, r) = \sum_{w \in \mathrm{pos}(t)} \mu(\mathbb{t}(t, r, w))$. The *behavior of* $\mathcal{A}$, denoted by $[\![\mathcal{A}]\!]$, is the mapping defined for every $t \in T_\Gamma$ by $[\![\mathcal{A}]\!](t) = \max_{r \in \mathrm{Acc}_\mathcal{A}(t)}(\mathrm{wt}_\mathcal{A}(t, r) + \nu(r(\varepsilon)))$, where the maximum over the empty set is $-\infty$ by convention.

For a max-plus-WTA $\mathcal{A} = (Q, \Gamma, \mu, \nu)$, a run of $\mathcal{A}$ on a Γ-context t is a run of the max-plus-WTA $\mathcal{A}' = (Q, \Gamma_\diamond, \mu', \nu)$ on t, where $\mu'(\diamond, q) = 0$ for all $q \in Q$ and $\mu'(d) = \mu(d)$ for $d \in \Delta_\mathcal{A}$. We denote $\mathrm{Run}^\diamond_\mathcal{A}(t) = \mathrm{Run}_{\mathcal{A}'}(t)$ and for $r \in \mathrm{Run}^\diamond_\mathcal{A}(t)$ write $\mathrm{wt}^\diamond_\mathcal{A}(t, r) = \mathrm{wt}_{\mathcal{A}'}(t, r)$. For a Γ-word s, we write $p \xrightarrow{s|x} q$ if there exists a run $r \in \mathrm{Run}^\diamond_\mathcal{A}(s)$ with $r(\Diamond_1(s)) = p$, $r(\varepsilon) = q$, and $\mathrm{wt}^\diamond_\mathcal{A}(s, r) = x$. In this case, r is said to *realize* $p \xrightarrow{s|x} q$. Note that $r \in \mathrm{Run}^\diamond_\mathcal{A}(s)$ implies $x \neq -\infty$.

For a max-plus-WTA $\mathcal{A}$, we define a relation $\leq$ on Q by $q \leq p$ iff $p \xrightarrow{s|x} q$ for some Γ-word $s \in T_{\Gamma_\diamond}$. We call $\mathcal{A}$ *trim* if for every $p \in Q$ there exists $t \in T_\Gamma$, $r \in \mathrm{Acc}(t)$, and $w \in \mathrm{pos}(t)$ with $r(w) = p$. The *trim part of* $\mathcal{A}$ is the automaton obtained by removing all states $p \in Q$ for which no such t, r, and w exist. This process obviously has no influence on $[\![\mathcal{A}]\!]$.

A max-plus-WTA $\mathcal{A}$ is called *deterministic* or *sequential* if for every $m \geq 0$, $a \in \Gamma^{(m)}$, and $\bar{p} \in Q^m$, there exists at most one $q \in Q$ with $\mu(\bar{p}, a, q) \neq -\infty$. We call $\mathcal{A}$ *unambiguous* if $|\mathrm{Acc}_\mathcal{A}(t)| \leq 1$ for every $t \in T_\Gamma$. We call the behavior $[\![\mathcal{A}]\!]$ of $\mathcal{A}$ *finitely sequential* if there exist deterministic max-plus-WTA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ over Γ with $[\![\mathcal{A}]\!] = \max_{i=1}^n [\![\mathcal{A}_i]\!]$, where the maximum is taken pointwise.

3 Main Result

We will show that for an unambiguous max-plus-WTA $\mathcal{A}$, it is decidable whether its behavior $[\![\mathcal{A}]\!]$ is finitely sequential. Moreover, if it is finitely sequential, we will obtain that the deterministic max-plus-WTA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ can be effectively constructed. For this, we follow ideas from [3], where the decidability of the finite sequentiality problem was proved for unambiguous max-plus word automata. The general outline of our proof is similar to that of [3] and presents itself as follows. We introduce the *tree fork property* and show that it

is decidable whether an unambiguous max-plus-WTA $\mathcal{A}$ satisfies this property. Then we show that the behavior of an unambiguous max-plus-WTA is finitely sequential if and only if it does not satisfy the tree fork property. In conclusion, we obtain the decidability of the finite sequentiality problem for unambiguous max-plus-WTA. Elementary proof methods can be used to show that $[\![\mathcal{A}]\!]$ is not finitely sequential if $\mathcal{A}$ satisfies the tree fork property. On the other hand, if $\mathcal{A}$ does not satisfy the tree fork property, we show how to construct finitely many unambiguous max-plus-WTA whose pointwise maximum is $[\![\mathcal{A}]\!]$, and which all satisfy the *twins property* [24]. Every unambiguous max-plus-WTA which satisfies the twins property possesses an effectively constructable deterministic equivalent [9]. Thus, we obtain finitely many deterministic max-plus-WTA whose pointwise maximum is $[\![\mathcal{A}]\!]$, which is hence finitely sequential.

In the following, we recall the twins property and introduce the tree fork property. Let Γ be a ranked alphabet. We begin with the concepts of *siblings* and *twins*. Intuitively, two states are called *siblings* if they can be "reached" by the same tree. Two siblings are called *twins* if for every Γ-word which can "loop" in both states, the maximal weight for the loop is the same in both states.

▶ **Definition 1.** *Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a max-plus-WTA. Two states $p, q \in Q$ are called* siblings *if there exists a tree $u \in T_\Gamma$ and runs $r^p, r^q \in \mathrm{Run}_\mathcal{A}(u)$ with $r^p(\varepsilon) = p$ and $r^q(\varepsilon) = q$. We recall that $\mathrm{Run}_\mathcal{A}(u)$ contains only valid runs.*

Two siblings p, q are called twins *if for every Γ-word s and weights*

$$x = \max_{\substack{r \in \mathrm{Run}^\diamond_\mathcal{A}(s) \\ r(\varepsilon) = r(\Diamond_1(s)) = p}} \mathrm{wt}^\diamond_\mathcal{A}(s, r) \qquad y = \max_{\substack{r \in \mathrm{Run}^\diamond_\mathcal{A}(s) \\ r(\varepsilon) = r(\Diamond_1(s)) = q}} \mathrm{wt}^\diamond_\mathcal{A}(s, r),$$

we have $x = y$ whenever $x \neq -\infty$ and $y \neq -\infty$ holds.

A max-plus-WTA is said to satisfy the *twins property* if all of its siblings are twins. For unambiguous max-plus-WTA, the twins property is a criterion to decide the sequentiality problem. An unambiguous max-plus-WTA possesses a deterministic equivalent if and only if it satisfies the twins property. For words, this result is due to [24, Theorem 12], for trees, we cite the following theorem.

▶ **Theorem 2** ([9, Lemma 5.10], [26, Lemma 17])**.** *Let $\mathcal{A}$ be a trim unambiguous max-plus-WTA. There exists a deterministic max-plus-WTA $\mathcal{A}'$ with $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ if and only if $\mathcal{A}$ satisfies the twins property. If it exists, it can be effectively constructed.*

It is intuitive that the twins property is a necessary condition if we consider the following *Lipschitz property* which every deterministic max-plus word automaton $\mathcal{A}$ satisfies [18, End of Section 2.4][24, Section 3.2]. If $\mathcal{A}$ is deterministic and L is the largest weight, in terms of absolute value, occurring in $\mathcal{A}$ (excluding $-\infty$), then for two words $w_1 = uv_1$ and $w_2 = uv_2$ which have an accepting run in $\mathcal{A}$, the difference between $[\![\mathcal{A}]\!](w_1)$ and $[\![\mathcal{A}]\!](w_2)$ can be at most $|L|(|v_1| + |v_2| + 2)$. This is clear since the unique runs of $\mathcal{A}$ on w_1 and w_2 will be identical on the prefix u, and then with every remaining letter of each word the difference between both runs cannot grow more than $|L|$.

If an unambiguous max-plus word automaton $\mathcal{A}$ does not satisfy the twins property, we can find states p and q which are siblings and not twins. We assume that our witnesses for this are u and s as above. We consider words of the form $w_1 = us^N v_p$ and $w_2 = us^N v_q$, where v_p and v_q are two fixed words which lead from p and q, respectively, to some final state. For every fixed L, we can choose N sufficiently large to ensure that $|[\![\mathcal{A}]\!](w_1) - [\![\mathcal{A}]\!](w_2)| > |L|(|v_p| + |v_q| + 2)$. It is thus not possible to determinize $\mathcal{A}$ if it does not satisfy the twins property.

The twins property is decidable for both max-plus word automata [1, 4, 24, 25, 16] and max-plus tree automata [8, Section 3]. Deciding whether a max-plus word automaton satisfies the twins property is PSPACE-complete [16]. For max-plus tree automata, the problem is thus PSPACE-hard, but no upper complexity bound is stated in [8]. Note that in general, it is undecidable whether two given siblings are twins [16], but for unambiguous max-plus automata, it was shown to be decidable on both words [1, Section 4] and trees [8, Section 3].

There exist unambiguous max-plus automata which cannot be determinized, but whose behavior is finitely sequential [18, Section 3.1], see also Figure 1. Thus, for the finite

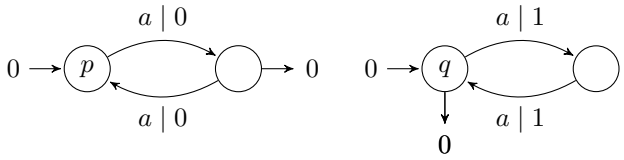


Figure 1 A max-plus word automaton $\mathcal{A}$ over the alphabet $\{a\}$ which is unambiguous, whose behavior is finitely sequential, but which does not satisfy the twins property as p and q are siblings but not twins. The behavior $[\![\mathcal{A}]\!]$ of $\mathcal{A}$ assigns 0 to all words of odd length and $|w|$ to all words w of even length.

sequentiality problem we inevitably have to deal with unambiguous automata in which not all siblings are twins. In the following, we will call two such states *rivals*. For unambiguous automata, which are the only type of max-plus-WTA we consider in this paper, the following definition is equivalent to being siblings and not twins.

▶ **Definition 3.** *Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a max-plus-WTA. Two states $p, q \in Q$ are called* rivals *if there exists a tree $u \in T_\Gamma$, runs $r^p, r^q \in \mathrm{Run}_\mathcal{A}(u)$ with $r^p(\varepsilon) = p$ and $r^q(\varepsilon) = q$, and a Γ-word s such that $p \xrightarrow{s|x} p$ and $q \xrightarrow{s|y} q$ with $x \neq y$.*

We do not have to consider a maximum over runs here since $\mathcal{A}$ is unambiguous. Also note that by our definition of $\mathrm{Run}^\diamond_\mathcal{A}(s)$, we have $x \neq -\infty$ and $y \neq -\infty$ above.

We now come to the tree fork property which, as we will show, is satisfied by an unambiguous max-plus-WTA if and only if its behavior is not finitely sequential. The property consists of two separate conditions. The first condition intuitively states that there exist two rivals p and q and a Γ-word t which can loop in p, and which can also lead from p to q. The second condition states that there exist two rivals which can occur at prefix-independent positions.

▶ **Definition 4.** *Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a max-plus-WTA. We say that $\mathcal{A}$ satisfies the* tree fork property *if at least one of the following two conditions is satisfied.*

- **(i)** *There exist rivals $p, q \in Q$ and a Γ-word t with $p \xrightarrow{t|z_p} p$ and $p \xrightarrow{t|z_q} q$ for some weights $z_p, z_q \in \mathbb{R}$. In this case, t is also called a p-q-*fork*.*
- **(ii)** *There exist rivals $p, q \in Q$, a 2-Γ-context $t \in T_{\Gamma_\diamond}$, and a run $r \in \mathrm{Run}^\diamond_\mathcal{A}(t)$ with $r(\Diamond_1(t)) = p$ and $r(\Diamond_2(t)) = q$.*

The tree fork property can be regarded as an extension of the *fork property* which was introduced in [3] and which for max-plus word automata plays the same role as the tree fork property does for max-plus tree automata. Condition (i) is essentially a tree version of the fork property. Casually put, if we take only condition (i) and replace "Γ-word" by "word", we obtain the fork property. The automaton depicted in Figure 2 is unambiguous and satisfies the fork property. Condition (ii) is new and possesses no counterpart in the fork property. We have the following theorem which relates the tree fork property to the finite sequentiality problem.

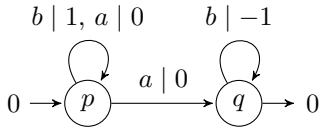


Figure 2 An unambiguous max-plus word automaton $\mathcal{A}$ over the alphabet $\{a, b\}$ which satisfies the fork property. With $u = a$ and $s = b$, we see that p and q are rivals, and a is a p-q-fork. All b's after the last a in a word are treated differently from the b's before the last a. A deterministic automaton cannot "guess" which a is the last in the word, and since there may be arbitrarily many a's in a word, even finitely many deterministic automata cannot compensate this inability to guess.

▶ **Theorem 5.** *Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a trim unambiguous max-plus-WTA over Γ. Then there exist deterministic max-plus-WTA $\mathcal{A}_1, \dots, \mathcal{A}_n$ over Γ with $[\![\mathcal{A}]\!] = \max_{i=1}^{n} [\![\mathcal{A}_i]\!]$ if and only if $\mathcal{A}$ does not satisfy the tree fork property. In particular, the finite sequentiality problem is decidable for unambiguous max-plus-WTA.*

Proof. Here, we only show that it is decidable whether $\mathcal{A}$ satisfies the tree fork property. The rest of the proof is deferred to Sections 4 and 5, where we show that the behavior of $\mathcal{A}$ is finitely sequential if and only if $\mathcal{A}$ does not satisfy the tree fork property.

To decide whether $\mathcal{A}$ satisfies condition (i), we first show that if there exists a p-q-fork t for two rivals p and q, then there exists a p-q-fork t' of height at most $|Q|^2$. If t is a p-q-fork with $\mathrm{height}(t) > |Q|^2$ and r_p and r_q are runs that realize $p \xrightarrow{t|z_p} p$ and $p \xrightarrow{t|z_q} q$ for some weights $z_p, z_q \in \mathbb{R}$, then by pigeon hole principle there are positions $w_1 <_p w_2$ in s with $r_p(w_1) = r_p(w_2)$ and $r_q(w_1) = r_q(w_2)$. Thus, by removing the part of t between w_1 and w_2, we obtain that $t' = t\langle t|_{w_2} \to w_1\rangle$ is a p-q-fork as well. Iterating this process, we obtain a p-q-fork of height at most $|Q|^2$.

Next, we identify all pairs of rivals, which is possible since for unambiguous max-plus tree automata, we can decide for every pair of states whether they are siblings and not twins [8, Section 3]. Then, for every pair of rivals p, q and all Γ-words t of height at most $|Q|^2$, we check whether t is a p-q-fork. If this yields no p-q-fork, $\mathcal{A}$ does not satisfy condition (i).

In order to decide whether $\mathcal{A}$ satisfies condition (ii), we first compute the relation $\leq$ on Q. This is possible since Q is a finite set and $\leq$ is the smallest transitive and reflexive relation satisfying $\mu(q_1, \dots, q_m, a, q_0) \neq -\infty \to q_0 \leq q_i$ for all transitions $(q_1, \dots, q_m, a, q_0) \in \Delta_{\mathcal{A}}$ and $i \in \{1, \dots, m\}$. Then, by the trimness of $\mathcal{A}$, condition (ii) is satisfied if and only if there exist two rivals p and q, a transition $(q_1, \dots, q_m, a, q_0) \in \Delta_{\mathcal{A}}$ with $\mu(q_1, \dots, q_m, a, q_0) \neq -\infty$, and indices $i, j \in \{1, \dots, m\}$ with $i \neq j$, $q_i \leq p$, and $q_j \leq q$. ◀

The following two sections are dedicated to completing the proof of Theorem 5.

4 Necessity

In this section, we show that if an unambiguous max-plus-WTA $\mathcal{A}$ satisfies either condition (i) or condition (ii) of the tree fork property, then its behavior $[\![\mathcal{A}]\!]$ is not finitely sequential. For condition (i), we adapt the corresponding proof from the word case [3, Theorem 2]. The proof relies on the Lipschitz property of deterministic max-plus automata and its approach is similar to the above outline that the twins property is a necessary condition for determinizability. We omit the proof here as it is a straightforward generalization of the proof from [3].

▶ **Theorem 6.** *Let $\mathcal{A}$ be a trim unambiguous max-plus-WTA over Γ. If $\mathcal{A}$ satisfies condition* (i) *of the tree fork property, then there do not exist deterministic max-plus-WTA $\mathcal{A}_1, \dots, \mathcal{A}_n$ over Γ with $[\![\mathcal{A}]\!] = \max_{i=1}^{n} [\![\mathcal{A}_i]\!]$.*

We consider condition (ii) of the tree fork property. On words, states cannot occur in prefix-independent positions. Thus, this condition is new for the tree case. Intuitively, the reason that the behavior of an unambiguous max-plus-WTA $\mathcal{A}$ cannot be finitely sequential if it satisfies condition (ii) is as follows. Assume we have a 2-Γ-context t and two rivals p and q as in condition (ii) and let u and s be as in the definition of rivals. Then we can construct trees of the form $t(s^n(u), s^n(u))$ such that, by increasing n, the difference between the weights on the two subtrees $s^n(u)$ is arbitrarily large. However, a deterministic automaton necessarily assigns the same weight to both subtrees.

▶ **Theorem 7.** *Let $\mathcal{A}$ be a trim unambiguous max-plus-WTA over Γ. If $\mathcal{A}$ satisfies condition* (ii) *of the tree fork property, then there do not exist deterministic max-plus-WTA $\mathcal{A}_1, \dots, \mathcal{A}_n$ over Γ with $[\![\mathcal{A}]\!] = \max_{i=1}^n [\![\mathcal{A}_i]\!]$.*

Proof (Sketch). For contradiction, we assume that $\mathcal{A}$ satisfies condition (ii) of the tree fork property and that there exist deterministic max-plus-WTA $\mathcal{A}_1, \dots, \mathcal{A}_n$ over Γ with $[\![\mathcal{A}]\!] = \max_{i=1}^n [\![\mathcal{A}_i]\!]$. We write $\mathcal{A}_i = (Q_i, \Gamma, \mu_i, \nu_i)$ and let $N = \max_{i=1}^n |Q_i|$. Let p, q, t be as in condition (ii) of the tree fork property and for the rivals p and q, let u and s be as in the definition of rivals.

By assumption, u can reach both p and q, and s can loop both in p and in q. In particular, the tree $s^N(u)$ can reach p by looping s in p and q by looping s in q. Due to our assumption on t, there hence exists a run on the tree $t' = t(s^N(u), s^N(u))$ which loops s in p on the left branch and in q on the right branch. Since $\mathcal{A}$ is trim, we may even assume that this run is accepting, as on top of t we can always add a Γ-word which leads to a final state.

We assume that $[\![\mathcal{A}]\!](t') = \max_{i=1}^n [\![\mathcal{A}_i]\!](t')$, so there must be some j with $[\![\mathcal{A}]\!](t') = [\![\mathcal{A}_j]\!](t')$. As $\mathcal{A}_j$ is deterministic, the unique accepting run of $\mathcal{A}_j$ on t' is identical on both $s^N(u)$-subtrees. Furthermore, since $N \geq |Q_j|$, we find that by pigeon hole principle some sub-Γ-word s^m of s^N loops in a state of $\mathcal{A}_j$ in the subtrees $s^N(u)$, say with weight z.

We let x and y be the weights such that $\mathcal{A}$ loops s^m in p with weight x and in q with weight y. By choice of s, we have $x \neq y$. We may assume that $x < y$. We consider two cases. First, if $z \geq \frac{x+y}{2}$, then for the tree $t^+ = t(s^{N+m}(u), s^N(u))$ we obtain

$$\max_{i=1}^n [\![\mathcal{A}_i]\!](t^+) \geq [\![\mathcal{A}_j]\!](t^+) = [\![\mathcal{A}_j]\!](t') + z \geq [\![\mathcal{A}_j]\!](t') + \frac{x+y}{2} > [\![\mathcal{A}_j]\!](t') + x = [\![\mathcal{A}]\!](t^+).$$

Note that this follows because $\mathcal{A}$ and $\mathcal{A}_j$ are both unambiguous, i.e., if we construct an accepting run on a given tree, we know that the weight of this run must be the weight assigned to the tree by the automaton. For the other case, namely that $z \leq \frac{x+y}{2}$, we see that for the tree $t^- = t(s^N(u), s^{N-m}(u))$ we obtain

$$\max_{i=1}^n [\![\mathcal{A}_i]\!](t^-) \geq [\![\mathcal{A}_j]\!](t^-) = [\![\mathcal{A}_j]\!](t') - z \geq [\![\mathcal{A}_j]\!](t') - \frac{x+y}{2} > [\![\mathcal{A}_j]\!](t') - y = [\![\mathcal{A}]\!](t^-).$$

In both cases, we see that $[\![\mathcal{A}]\!] = \max_{i=1}^n [\![\mathcal{A}_i]\!]$ does not hold, which is a contradiction. ◀

Together, Theorems 6 and 7 show that if a trim unambiguous max-plus-WTA satisfies the tree fork property, then its behavior is not finitely sequential.

5 Sufficiency

In this section, we show that the behavior of an unambiguous max-plus-WTA $\mathcal{A}$ which does not satisfy the tree fork property is finitely sequential. For simplicity, we begin with a description of our method of proof on max-plus word automata and compare it to the proof method of Bala and Koniński [3].

Both proofs work by distributing the runs of $\mathcal{A}$ across a finite set of unambiguous max-plus word automata such that all of these automata satisfy the twins property. This distribution essentially has the aim of separating the rivals of $\mathcal{A}$. By Theorem 2, these unambiguous automata can then be determinized. The major difference between our approach and that of [3] lies in way we obtain these unambiguous automata. To understand our approach, let p and q be two rivals of $\mathcal{A}$. Furthermore, let $u = u_1 \cdots u_n$ be a word for which there exist valid runs $r^p = p_0 \xrightarrow{u_1} p_1 \xrightarrow{u_2} \cdots \xrightarrow{u_{n-1}} p_{n-1} \xrightarrow{u_n} p$ and $r^q = q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} \cdots \xrightarrow{u_{n-1}} q_{n-1} \xrightarrow{u_n} q$ of $\mathcal{A}$ on u.

We now show that the first occurrence of either p or q in the runs r^p and r^q serves as a "distinguisher" between the two runs. We let i be the smallest index with the property that $p_i \in \{p, q\}$. Similarly, we let j be the smallest index with the property that $q_j \in \{p, q\}$. We obtain valid runs $p_i \xrightarrow{u_{i+1} \cdots u_n} p$ and $q_j \xrightarrow{u_{j+1} \cdots u_n} q$.

Now assume it would hold that $i = j$ and $p_i = q_j$, i.e., the first occurrence is at the same position in the word, and also the state at this position is the same in both runs. Then with $t = u_{i+1} \cdots u_n$, we see that we have valid runs $p_i \xrightarrow{t} p$ and $p_i \xrightarrow{t} q$, where $p_i \in \{p, q\}$. Thus, $\mathcal{A}$ would satisfy the fork property. Since our assumption is that $\mathcal{A}$ does not satisfy the fork property, we have either $i \neq j$ or $p_i \neq q_j$.

This fundamental property is also used in the corresponding proof of [3], but our way of exploiting it differs from [3]. In their proof for word automata, Bala and Koniński use this property implicitly to show that certain states of a modified Schützenberger covering of $\mathcal{A}$ occur at most once in every run [3, Lemma 6]. They can therefore construct a new max-plus automaton which for each run keeps a record of all occurrences of these states. The above mentioned unambiguous automata are then obtained by separating runs with differing records into different automata. For tree automata, the number of these occurrences is unfortunately not bounded, for reasons which we will also indicate below.

For now, we continue outlining our new approach, which is to construct an automaton which adds a distinguishing marker to every run when first encountering one of the rivals p or q. This marker consists of a number, which is used to distinguish occurrences at different positions, and the state from $\{p, q\}$ which was visited first. Whenever reading a letter which causes some valid run to visit p or q for the first time, the automaton selects the smallest marker which was not used by any valid run on the prefix read so far, and annotates it to the run. For example, assume that neither p nor q occur in any valid run the word u, but that our run r on ua leads to p. Then r obtains the marker 1_p. Now assume there is a valid run on uaa which leads to p and which visited neither p nor q before that. Then this run obtains the marker 2_p, since 1_p is already assigned to r. Next, assume that after reading $uaaa$ another marker for p has to be assigned, and that r cannot be extended to a valid run on uaa. Then we assign the marker 1_p, as now no valid run on uaa exists to which the marker 1_p is assigned. See Figure 3 for an example of this annotation process on the word aaa for the automaton depicted there.

With this procedure, runs like r^p and r^q above receive different markers since either one run obtains a marker later than the other, and therefore a different marker, or at least the states they visit first are different, which also leads to a different marker. To separate the rivals of $\mathcal{A}$, we can thus make a copy of $\mathcal{A}$ for every marker, and only allow runs which carry the respective automaton's marker. Whenever a different marker would be assigned, the execution of the run is blocked.

Note here that the number of markers we need for this annotation process is bounded. Since the automaton $\mathcal{A}$ is unambiguous, the number of *valid* runs on every given word is bounded by the number of states in $\mathcal{A}$. If this were not the case, there would exist two distinct

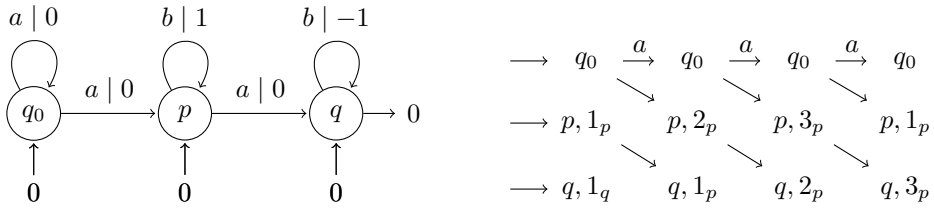


Figure 3 On the left, an unambiguous max-plus word automaton over the alphabet $\{a, b\}$ which does not satisfy the twins property but whose behavior is finitely sequential. On the right, an illustration of the runs of the automaton on the words ε, a, aa, and aaa together with appropriate markers. Arrows indicate a transition. The states p and q are rivals with witnesses $u = \varepsilon$ and $s = b$.

valid runs on the same word which lead to the same state, from which a counterexample to the unambiguity of $\mathcal{A}$ could be constructed. In particular, the number of markers assigned at any given "time" is bounded by the number of states of $\mathcal{A}$.

All of this can easily be generalized to the situation where there is more than one pair of rivals. Then, runs simply obtain a marker for each pair of rivals of the automaton, and the copies of $\mathcal{A}$ allow a distinguished marker for each of these pairs.

Unfortunately, these ideas do not translate to trees as easily. For example, consider the runs in Figure 4. Intuitively, both runs should obtain the marker 1_p. However, since p and q are rivals, this marker does not serve the purpose of distinguishing runs as it does in the word case. The first p occurs in different subtrees of both runs, thus the annotation of distinct markers is not possible. Also, it is easy to construct an automaton where a rival p can occur at arbitrarily many pairwise prefix-independent positions, thus a simple lexicographic distinction is not possible. This is also the reason why the approach from [3] does not work for tree automata.

$$\begin{aligned} \nu(q) &= 0 \\ \mu(p, a, q) &= 0 \\ \mu(p, b, p) &= 1 \\ \mu(q, b, q) &= -1 \\ \mu(p, q_0, c, p) = \mu(q_0, p, c, q) &= 0 \\ \mu(d, p) = \mu(d, q_0) &= 0 \end{aligned}$$

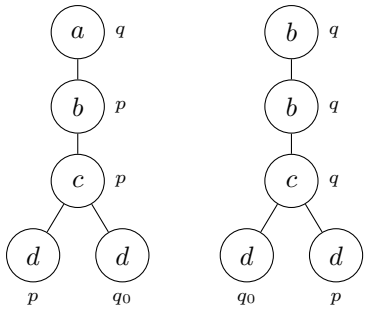


Figure 4 Two accepting runs of the max-plus tree automaton $\mathcal{A} = (\{q_0, p, q\}, \Gamma, \mu, \nu)$ over the ranked alphabet $\Gamma = \{a, b, c, d\}$ where $c \in \Gamma^{(2)}, a, b \in \Gamma^{(1)}, d \in \Gamma^{(0)}$. All unspecified weights are assumed to be $-\infty$. The states p and q are rivals.

Our solution is to distribute not the runs of the automaton $\mathcal{A}$, but the runs of its *Schützenberger covering*. The Schützenberger covering of a max-plus automaton $\mathcal{A}$ is a max-plus automaton which possesses the same behavior as $\mathcal{A}$. It has already been employed in a number of decidability results for max-plus automata [18, 3, 2, 26]. Its construction is inspired by a paper of Schützenberger [32] and was made explicit by Sakarovitch in [29].

To better explain the idea behind its construction, we first point out a certain aspect of the classical powerset construction for finite automata [28]. Assume that $\mathcal{D}$ is the result of applying the powerset construction to an NFA $\mathcal{B}$. Then we might say that for a word

$w = w_1 w_2$, the state which $\mathcal{D}$ is in after reading the prefix w_1 is the set of all states which $\mathcal{B}$ *could* be in after reading w_1. Similarly, the Schützenberger covering of a max-plus automaton $\mathcal{A}$ annotates to every state of a run of $\mathcal{A}$ on a word w the set of all states which "$\mathcal{A}$ could be in" at this point, i.e., which can be reached by some valid run on the considered prefix of w. Like the powerset construction, these ideas easily carry over to trees.

The reason we consider the Schützenberger covering of $\mathcal{A}$ is that each pair $\mathbf{p}, \mathbf{q}$ of its rivals satisfies the following property. For every tree t, either (1) $\mathbf{p}$ and $\mathbf{q}$ do not occur together in any run on t or (2) $\mathbf{p}$ and $\mathbf{q}$ occur only linearly, i.e., there is a distinguished branch of t such that for every run on t, all occurrences of $\mathbf{p}$ and $\mathbf{q}$ lie on this branch. In particular, the situation of Figure 4 is not possible. All pairs which satisfy the first condition can simply be separated into different automata, all pairs which satisfy the second condition can be handled like in the word case. The proof of this is non-trivial and needs some preparation. We begin with the formal definition of the Schützenberger covering.

For the rest of this section, let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a trim unambiguous max-plus-WTA which does not satisfy the tree fork property.

▶ **Definition 8** (Schützenberger covering, [29]). *The Schützenberger covering $\mathcal{S} = (Q_\mathcal{S}, \Gamma, \mu_\mathcal{S}, \nu_\mathcal{S})$ of $\mathcal{A}$ is the trim part of the max-plus-WTA $(Q \times \mathcal{P}(Q), \Gamma, \mu', \nu')$ defined for $a \in \Gamma$ with $\mathrm{rk}_\Gamma(a) = m$ and $(p_0, P_0), \dots, (p_m, P_m) \in Q \times \mathcal{P}(Q)$ by*

$$\mu'((p_1, P_1), \dots, (p_m, P_m), a, (p_0, P_0)) =$$
$$\begin{cases} \mu(p_1, \dots, p_m, a, p_0) & \text{if } P_0 = \{q_0 \in Q \mid \exists (q_1, \dots, q_m) \in P_1 \times \dots \times P_m \text{ with} \\ & \mu(q_1, \dots, q_m, a, q_0) \neq -\infty\} \\ -\infty & \text{otherwise} \end{cases}$$
$$\nu'(p_0, P_0) = \nu(p_0).$$

We let $\pi_1 \colon Q \times \mathcal{P}(Q) \to Q$, $(p, P) \mapsto p$ and $\pi_2 \colon Q \times \mathcal{P}(Q) \to \mathcal{P}(Q)$, $(p, P) \mapsto P$ be the projections.

It is elementary to show that for a run of $\mathcal{S}$ on a tree t, the second entry of the state at a position w consists of all states of $\mathcal{A}$ which can be reached by a valid run of $\mathcal{A}$ on $t|_w$. In particular, two runs on the same tree coincide on their second entries. Furthermore, projecting all states of a run of $\mathcal{S}$ to their first coordinate yields a run of $\mathcal{A}$, and the weights of these runs coincide. It follows that $\mathcal{S}$ is unambiguous and satisfies $[\![\mathcal{S}]\!] = [\![\mathcal{A}]\!]$. Also, $\mathcal{S}$ is trim by definition.

We can make the following observation about the rivals of $\mathcal{S}$. Let $\mathbf{p}$ and $\mathbf{q}$ be rivals of $\mathcal{S}$ and let u and s be as in the definition of rivals. Since all runs of $\mathcal{S}$ on u coincide on the second entry of the state at the root, $\mathbf{p}$ and $\mathbf{q}$ also coincide on their second entry. Moreover, as projecting the runs of $\mathcal{S}$ on u and s to their first entries yields runs of $\mathcal{A}$ on u and s, respectively, we additionally see that the first entries of $\mathbf{p}$ and $\mathbf{q}$ are rivals in $\mathcal{A}$. Thus, if two states $\mathbf{p}, \mathbf{q} \in Q_\mathcal{S}$ are rivals in $\mathcal{S}$, then $\mathbf{p} = (p, P)$ and $\mathbf{q} = (q, P)$ for some set $P \subseteq Q$ and two states $p, q \in Q$ which are rivals in $\mathcal{A}$.

In order to prove some deeper results about the rivals of $\mathcal{S}$, we need two preparatory lemmata. As a first simplification, we show that we may assume that two rivals p and q of $\mathcal{A}$ are always comparable with respect to the relation $\leq$. To see this, note that by condition (ii) of the tree fork property, p and q may not occur in prefix-independent positions in a run. If in addition, p and q can also not appear in prefix-dependent positions in a run, they never appear in the same run of $\mathcal{A}$. Thus, we can create two copies of $\mathcal{A}$, one in which we remove p and one in which we remove q, and the pointwise maximum of these two automata will be equivalent to the behavior of $\mathcal{A}$.

▶ **Lemma 9.** *For all rivals $p, q \in Q$, we may assume that either $p \leq q$ or $q \leq p$, or both.*

Next, we note an elementary statement about self-maps $f\colon X \to X$. Namely, if X is a finite set and $f\colon X \to X$ a mapping, then for every $a \in X$ there exists some element $b \in X$ and an integer $n \geq 1$ such that after n iterations of f, both a and b are mapped to b. To see this, consider the elements $a, f(a), f^2(a), \ldots, f^{|X|}(a)$. By pigeon hole principle, there are numbers $0 \leq m_1 < m_2 \leq |X|$ with $f^{m_1}(a) = f^{m_2}(a)$. Then if we choose $n \geq m_1$ as a multiple of $m_2 - m_1$ and $b = f^n(a)$, we see that $f^n(a) = b = f^n(b)$.

▶ **Lemma 10.** *Let X be a finite set and $f\colon X \to X$ a mapping. Then for every $a \in X$, there exists an element $b \in X$ and an integer $n \geq 1$ with $f^n(a) = b = f^n(b)$. Here, f^n is the n-th iterate of f, i.e., $f^0 = \mathrm{id}_X$ and $f^{m+1} = f \circ f^m$.*

We come to the first important property which all rivals of $\mathcal{S}$ satisfy. Namely, if $P \subseteq Q$ is the second entry of *some* rival, then it cannot occur in the form of a "triangle" in any valid run of $\mathcal{S}$. More precisely, if we have a run r and positions w, wv_1, and wv_2 such that the second entry of $r(w)$, $r(wv_1)$, and $r(wv_2)$ is P, then wv_1 and wv_2 are prefix-dependent.

▶ **Lemma 11.** *Let $(p, P), (q, P) \in Q_{\mathcal{S}}$ be rivals in $\mathcal{S}$. Furthermore, let $t' \in T_\Gamma$ be a tree, $r' \in \mathrm{Run}_{\mathcal{S}}(t')$ a run of $\mathcal{S}$ on t', and $w_1, w_2 \in \mathrm{pos}(t')$ be positions in t'. If $\pi_2 \circ r'(\varepsilon) = \pi_2 \circ r'(w_1) = \pi_2 \circ r'(w_2) = P$, then w_1 and w_2 are prefix-dependent.*

Proof (Sketch). We proceed by contradiction and assume that t', r', w_1, w_2 as in the statement of the lemma exist such that w_1 and w_2 are prefix-independent. We show that then, $\mathcal{A}$ satisfies condition (i) of the tree fork property. For the rivals (p, P) and (q, P), let u and s be as in the definition of rivals and let $v = \Diamond_1(s)$.

By assumption, u can reach (p, P) and s can loop in (p, P), thus the trees $s^{|P|}(u)$ and $s^{|P|^{|P|}}(u)$ can reach (p, P). Due to the construction of $\mathcal{S}$, this means both of these trees can also reach the states of r' at w_1 and w_2. In particular, there exists a run of $\mathcal{S}$ on the tree $t = t'\langle s^{|P|}(u) \to w_1\rangle\langle s^{|P|^{|P|}}(u) \to w_2\rangle$ and for this run, the second entry of every state at the beginning or end of an s-loop is P. In addition, t leads to a state with second entry P, so there in fact exist $|P|$ runs of $\mathcal{S}$ on t, one for each state in P. We let $r_1, \ldots, r_{|P|}$ be the projections of these runs to their first entry and obtain $|P|$ runs of $\mathcal{A}$ on t where the state at the root and all states at the beginning or end of an s-loop are from P.

By pigeonhole principle, there is some subloop s^n below w_2 which loops in all runs at the same time, i.e., where for some n_1 we have $r_i(w_2 v^{n_1}) = r_i(w_2 v^{n_1+n})$ for all runs r_i. For each r_i, we let $q_i = r_i(w_2 v^{n_1}) \in P$ be the state which r_i loops in and let x_i be the weight of this loop.

If $x_i \neq x_j$ for some i and j, the states q_i and q_j are rivals in $\mathcal{A}$ with witnesses u and s^n. By Lemma 9, we may therefore assume $q_i \leq q_j$. Again by pigeon hole principle, the run r_i loops below w_1 in s^m for some $m \geq 1$ with some state $p_i \in P$, say with weight y_i. Due to $x_i \neq x_j$, we have $mx_i \neq ny_i$ or $mx_j \neq ny_i$. Since u can reach every state from P, the state p_i is thus a rival of q_i or q_j with witnesses u and s^{nm}. From the existence of r_i and the assumption that $q_i \leq q_j$, we see that p_i can occur prefix-independently both from q_i and from q_j. This is a contradiction to the assumption that $\mathcal{A}$ does not satisfy the tree fork property. It must therefore hold that $x_1 = \ldots = x_{|P|}$.

We let x and y be the weights such that $\mathcal{A}$ loops s in p with weight x and in q with weight y. Then from $x \neq y$ it follows that $nx \neq x_1$ or $ny \neq x_1$, so the states q_i are either all rivals of p or all rivals of q with witnesses u and s^n. We assume all q_i to be rivals of p and apply Lemma 10 to the mapping $f\colon P \to \{q_1, \ldots, q_{|P|}\}, r_i(\varepsilon) \mapsto q_i$ with $a = p$ to obtain $q_j \in P$ and $m \geq 1$ such that $f^m(p) = q_j = f^m(q_j)$. Then with $\tilde{s} = t\langle \diamond \to w_2 v^{n_1}\rangle$, we see that the Γ-word $\tilde{s}^m$ is a q_j-p-fork, i.e., $\mathcal{A}$ satisfies condition (i) of the tree fork property. ◀

The previous lemma showed that if P is the second entry of some rival from $\mathcal{S}$, states with second entry P do not occur in the form of a triangle. The next lemma shows that even prefix-independent occurrences are restricted to a certain degree. Namely, if we have two rivals (p, P) and (q, P) with $p \leq q$, then all occurrences of P are prefix-dependent on (p, P).

▶ **Lemma 12.** *Let $(p, P), (q, P) \in Q_{\mathcal{S}}$ be rivals in $\mathcal{S}$ with $p \leq q$. Furthermore, let $t' \in T_\Gamma$ be a tree, $r' \in \mathrm{Run}_{\mathcal{S}}(t')$ a run of $\mathcal{S}$ on t', and $w_1 \in \mathrm{pos}(t')$ a position in t' with $r'(w_1) = (p, P)$. Then all positions $w_2 \in \mathrm{pos}(t')$ with $\pi_2 \circ r'(w_2) = P$ are prefix-dependent on w_1.*

Proof (Sketch). We proceed by contradiction and take $(p, P), (q, P), t', r', w_1$ as in the statement of the lemma and assume that there exists a position $w_2 \in \mathrm{pos}(t')$ which is prefix-independent from w_1 and for which $\pi_2 \circ r'(w_2) = P$. We show that under these assumptions, $\mathcal{A}$ satisfies condition (ii) of the tree fork property. For the rivals (p, P) and (q, P), let u and s be as in the definition of rivals.

As we have seen in the proof of Lemma 11, the tree $s^{|P|}(u)$ can reach (p, P), so due to the construction of $\mathcal{S}$, it can also reach the state of r' at w_2. Thus, there exists a run of $\mathcal{S}$ on the tree $t = t'\langle s^{|P|}(u) \to w_2\rangle$ for which the state at w_1 is (p, P) and for which the second entry of every state at the beginning or end of an s-loop is P. We let r be the projection of this run to the first entries of the states.

By pigeonhole principle, we find some subloop s^n below w_2 in r which loops in a state $p' \in P$. Let z be the weight of this loop and let x and y be the weights such that $\mathcal{A}$ loops s in p with weight x and in q with weight y. Due to $x \neq y$, we have $nx \neq z$ or $ny \neq z$. Since u can reach every state from P, the state p' is a rival of p or q with witnesses u and s^n. From the fact that $r(w_1) = p$ and the assumption that $p \leq q$, we see that p' can occur prefix-independently both from p and from q. This is a contradiction to the assumption that $\mathcal{A}$ does not satisfy the tree fork property. ◀

We can now prove that every run of $\mathcal{S}$ satisfies at least one of the following two conditions. If (p, P) and (q, P) are rivals with $p \leq q$, then for every run r on a tree t either (i) (p, P) does not occur in r or (ii) all states with second entry P occur along a distinguished branch of t. This property enables us to apply the idea from the word case of using markers to indicate the first visit of a rival in a run. If u is a witness for (p, P) and (q, P) to be siblings, there is in particular a run on u which leads to (p, P). This run then satisfies condition (ii) and since the second entries of runs on the same tree coincide, *all* states with second entry P occur along a distinguished branch of u in *every* run of $\mathcal{S}$ on u. This is true in particular for the two rivals (p, P) and (q, P).

▶ **Theorem 13.** *Let $(p, P), (q, P) \in Q_{\mathcal{S}}$ be rivals in $\mathcal{S}$ with $p \leq q$. Then for every tree $t \in T_\Gamma$ and every run $r \in \mathrm{Run}_{\mathcal{S}}(t)$ of $\mathcal{S}$ on t, at least one of the following two conditions holds.*

- **(i)** *The state (p, P) does not occur in r, i.e., $r(w) \neq (p, P)$ for all $w \in \mathrm{pos}(t)$.*
- **(ii)** *All states with second entry P occur linearly in r, i.e., for all $w_1, w_2 \in \mathrm{pos}(t)$ with $\pi_2 \circ r(w_1) = \pi_2 \circ r(w_2) = P$ we have $w_1 \leq_p w_2$ or $w_2 \leq_p w_1$.*

Proof. Let $(p, P), (q, P), t, r$ be as in the statement of the theorem. Assume that (i) does not hold, i.e., there is a position $w \in \mathrm{pos}(t)$ with $r(w) = (p, P)$. Let $w_1, w_2 \in \mathrm{pos}(t)$ be two positions with $\pi_2 \circ r(w_1) = \pi_2 \circ r(w_2) = P$. By Lemma 12, we see that then w_1 and w_2 are prefix-dependent on w. From the definition of the prefix relation, we see that if either $w_1 \leq_p w$ or $w_2 \leq_p w$, then all three positions are in prefix relation. We thus consider the case that $w \leq_p w_1$ and $w \leq_p w_2$. In this case, we see from Lemma 11 that w_1 and w_2 are prefix-dependent. ◀

We now construct the automaton which tracks the first occurrences of rivals, and whose runs we distribute across multiple automata in order to separate all rivals.

▶ **Construction 14.** Let $R_1, \dots, R_n \subseteq Q_{\mathcal{S}}$ be an enumeration of all (unordered) pairs of rivals of $\mathcal{S}$, i.e., for all $i \in \{1, \dots, n\}$ we have $R_i = \{(p_i, P_i), (q_i, P_i)\}$ such that (p_i, P_i) and (q_i, P_i) are rivals in $\mathcal{S}$ and for every two rivals $(p, P), (q, P) \in Q_{\mathcal{S}}$, we have $R_i = \{(p, P), (q, P)\}$ for some $i \in \{1, \dots, n\}$. Since by Lemma 9, we may assume that all rivals in $\mathcal{A}$ are in $\leq$-relation, we assume in the following that p_i and q_i are named such that $p_i \leq q_i$ for all $i \in \{1, \dots, n\}$.

For each pair of rivals R_i, we define a set of markers by $I_i = \{0, |Q|+1\} \cup (\{1, \dots, |Q|\} \times R_i)$. The set of all combined records of markers is defined by $I = I_1 \times \ldots \times I_n$. For $\bar{a} \in I$, we denote by $\bar{a}[i]$ the i-th entry of $\bar{a}$.

Intuitively, the states of our new automaton will consist of a state from $\mathcal{S}$ together with a record of markers from I. However, in order to properly update markers, we need to know in each step the records of *all* other runs as well. Thus, our states will be from $Q_{\mathcal{S}} \times I \times \mathcal{P}(Q_{\mathcal{S}} \times I)$. In order to define the transition function of our new automaton, we first define how markers are updated. Assume we transition into the state $\mathbf{q} \in Q_{\mathcal{S}}$, we have m subtrees below our current position in the tree, the runs we consider on these subtrees have obtained markers $\bar{a}_1, \dots, \bar{a}_m \in I$, and the sets of states we *could* be in on these trees, together with their markers, are given by $A_1, \dots, A_m \subseteq Q_{\mathcal{S}} \times I$.

Every pair $(\mathbf{p}, \bar{a}) \in A_k$ corresponds to exactly one run of $\mathcal{S}$ on the k-th subtree together with its markers. Since $\mathcal{S}$ is unambiguous, we can therefore assume that $|A_k| \leq |Q|$. Also, since $\bar{a}_k$ is the marker of a run on the k-th subtree, we may assume that $(Q_{\mathcal{S}} \times \{\bar{a}_k\}) \cap A_k \neq \emptyset$.

For $k \in \{1, \dots, m\}$ and $i \in \{1, \dots, n\}$, we define the sets of unassigned counters $B_k[i] \subseteq \{1, \dots, |Q|\}$ by $B_k[i] = \{1, \dots, |Q|\} \setminus \{j \mid \exists (\mathbf{p}, \bar{a}) \in A_k \text{ with } \bar{a}[i] \in \{j\} \times R_i\}$. Then if for all $k \in \{1, \dots, m\}$ we have $|A_k| \leq |Q|$ and $(Q_{\mathcal{S}} \times \{\bar{a}_k\}) \cap A_k \neq \emptyset$, we define the record of markers $\bar{b}$ for our current position by (explanations below)

$$\bar{b}[i] = \begin{cases} 0 & \text{if } m = 0 \text{ and } \mathbf{q} \notin R_i \\ (1, \mathbf{q}) & \text{if } m = 0 \text{ and } \mathbf{q} \in R_i \\ \bar{a}_k[i] & \text{if } k \in \{1, \dots, m\} \text{ satisfies: } \bar{a}_l[i] = 0 \text{ for all } l \neq k \text{ and} \\ & \text{either } \bar{a}_k[i] \neq 0 \text{ or } \mathbf{q} \notin R_i \\ (\min B_k[i], \mathbf{q}) & \text{if } \mathbf{q} \in R_i \text{ and } k \in \{1, \dots, m\} \text{ satisfies:} \\ & \bar{a}_k[i] = 0 \text{ and for all } l \neq k \text{ and all } (\mathbf{p}, \bar{a}) \in A_l \colon \bar{a}[i] = 0 \\ |Q| + 1 & \text{otherwise} \end{cases}$$

for $i \in \{1, \dots, n\}$. If $|A_k| > |Q|$ or $Q_{\mathcal{S}} \times \{\bar{a}_k\} \cap A_k = \emptyset$ for some k, we let $\bar{b}[1] = \ldots = \bar{b}[n] = |Q| + 1$. Note that $\min B_k[i]$ in above case distinction always exists since $|A_k| \leq |Q|$, $(Q_{\mathcal{S}} \times \{\bar{a}_k\}) \cap A_k \neq \emptyset$, and in the case in question we have $\bar{a}_k[i] = 0$. We define $\mathcal{I}(\mathbf{q}, \bar{a}_1, \dots, \bar{a}_m, A_1, \dots, A_m) = \bar{b}$.

Case 1 of the definition above means our current position is a leaf and $\mathbf{q}$ is not from R_i, so we assign the dummy marker 0. Case 2 means our current position is a leaf and $\mathbf{q}$ is from R_i, so we assign the marker $(1, \mathbf{q})$. Case 3 means that either (1) there is exactly one subtree below our current position which already obtained a marker different from 0 and we keep this marker for our current position, or (2) the markers of all subtrees are 0 and $\mathbf{q}$ is also not from R_i, so we continue with the dummy marker 0.

Case 4 means the markers of all subtrees below our current position are 0, the state $\mathbf{q}$ is from R_i, and there is at most one subtree on which runs exist that obtained a marker for R_i. Then, we take the smallest number which is not already used in a marker for R_i in any run on this subtree, and use this number together with $\mathbf{q}$ as the marker for our current position.

Case 5, the "otherwise-case", applies in two situations. This case means that either (1) two distinct subtrees below our current position have already obtained a marker, or that (2) all markers below our current position are 0 and $\mathbf{q}$ is from R_i, but we cannot apply case 4 as there are two distinct subtrees on which runs exist which obtained markers for R_i. In other words, markers were assigned nonlinearly, and our run satisfies only condition (i) of Theorem 13. In this case, we assign the dummy marker $|Q|+1$.

The extra case covers the situation where in case 4, the set $B_k[i]$ would be empty. This case is necessary to ensure our definition is formally complete, but in our applications of the operator $\mathcal{I}$ it will not actually occur.

We define our "run-marking" max-plus-WTA $\mathcal{B} = (\tilde{Q}, \Gamma, \tilde{\mu}, \tilde{\nu})$ as follows. We let $\tilde{Q}' = Q_{\mathcal{S}} \times I \times \mathcal{P}(Q_{\mathcal{S}} \times I)$ and let $\mathcal{B}$ be the trim part of the automaton $\mathcal{B}' = (\tilde{Q}', \Gamma, \tilde{\mu}', \tilde{\nu}')$ defined for $a \in \Gamma$ with $\mathrm{rk}_\Gamma(a) = m$ and $(\mathbf{p}_0, \bar{a}_0, A_0), \dots, (\mathbf{p}_m, \bar{a}_m, A_m) \in Q_{\mathcal{S}} \times I \times \mathcal{P}(Q_{\mathcal{S}} \times I)$ by

$$\tilde{\mu}'((\mathbf{p}_1, \bar{a}_1, A_1), \dots, (\mathbf{p}_m, \bar{a}_m, A_m), a, (\mathbf{p}_0, \bar{a}_0, A_0)) =$$
$$\begin{cases} \mu_{\mathcal{S}}(\mathbf{p}_1, \dots, \mathbf{p}_m, a, \mathbf{p}_0) & \text{if } \bar{a}_0 = \mathcal{I}(\mathbf{p}_0, \bar{a}_1, \dots, \bar{a}_m, A_1, \dots, A_m) \text{ and} \\ & A_0 = \{(\mathbf{q}_0, \bar{b}_0) \in Q_{\mathcal{S}} \times I \mid \exists((\mathbf{q}_1, \bar{b}_1), \dots, (\mathbf{q}_m, \bar{b}_m)) \in \\ & A_1 \times \ldots \times A_m \text{ with } \mu_{\mathcal{S}}(\mathbf{q}_1, \dots, \mathbf{q}_m, a, \mathbf{q}_0) \neq -\infty \text{ and} \\ & \bar{b}_0 = \mathcal{I}(\mathbf{q}_0, \bar{b}_1, \dots, \bar{b}_m, A_1, \dots, A_m)\} \\ -\infty & \text{otherwise} \end{cases}$$
$$\tilde{\nu}'(\mathbf{p}_0, \bar{a}_0, A_0) = \nu_{\mathcal{S}}(\mathbf{p}_0).$$

The automaton $\mathcal{B}$ is trim, unambiguous, and satisfies $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$. Furthermore, if $(\mathbf{p}, \bar{a}, A)$ and $(\mathbf{q}, \bar{b}, B)$ are rivals in $\mathcal{B}$, we have $\bar{a}[i] \neq \bar{b}[i]$ for some $i \in \{1, \dots, n\}$.

We come to our final construction where we distribute the runs of $\mathcal{B}$ across multiple automata. For every record of markers $\bar{c} \in I$, we construct one automaton $\mathcal{B}_{\bar{c}}$ which for each pair of rivals R_i admits only runs using the markers 0 and $\bar{c}[i]$. All runs in which rivals occur nonlinearly are covered by admitting the marker $|Q|+1$. All other runs are covered by admitting an appropriate marker from $\{1, \dots, |Q|\} \times R_i$.

▶ **Construction 15.** For every tuple $\bar{c} \in I$, we define a max-plus-WTA $\mathcal{B}_{\bar{c}} = (\tilde{Q}_{\bar{c}}, \Gamma, \tilde{\mu}, \tilde{\nu})$ by removing states from $\mathcal{B}$ through

$$\tilde{Q}_{\bar{c}} = \{(\mathbf{p}, \bar{a}, A) \in \tilde{Q} \mid \text{for all } i \in \{1, \dots, n\} \text{ it holds: if } \bar{c}[i] = |Q|+1 \text{ then } \mathbf{p} \neq (p_i, P_i),$$
$$\text{and if } \bar{c}[i] \neq |Q|+1 \text{ then } \bar{a}[i] \in \{0, \bar{c}[i]\}\}.$$

The automata $\mathcal{B}_{\bar{c}}$ are unambiguous, their pointwise maximum is equivalent to the behavior of $\mathcal{A}$, and they all satisfy the twins property, which means that they can be determinized.

▶ **Theorem 16.** *We have* $[\![\mathcal{A}]\!] = \max_{\bar{c} \in I} [\![\mathcal{B}_{\bar{c}}]\!]$ *and for every* $\bar{c} \in I$*, the automaton* $\mathcal{B}_{\bar{c}}$ *is unambiguous and satisfies the twins property.*

We now obtain a finitely sequential representation of $\mathcal{A}$ by applying Theorem 2 to the automata $\mathcal{B}_{\bar{c}}$. In particular, we see that the behavior of a trim unambiguous max-plus-WTA is finitely sequential if it does not satisfy the tree fork property. This concludes the proof of Theorem 5. ◀

References

1 Cyril Allauzen and Mehryar Mohri. Efficient Algorithms for Testing the Twins Property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.

2 Sebastian Bala. Which Finitely Ambiguous Automata Recognize Finitely Sequential Functions? In Krishnendu Chatterjee and Jiří Sgall, editors, *38th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 8087 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2013.

3 Sebastian Bala and Artur Koniński. Unambiguous Automata Denoting Finitely Sequential Functions. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *7th International Conference on Language and Automata Theory and Applications (LATA)*, volume 7810 of *Lecture Notes in Computer Science*, pages 104–115. Springer, 2013.

4 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

5 Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Springer, 1988.

6 Johanna Björklund, Frank Drewes, and Niklas Zechner. An Efficient Best-Trees Algorithm for Weighted Tree Automata over the Tropical Semiring. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *9th International Conference on Language and Automata Theory and Applications (LATA)*, volume 8977 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2015.

7 Meera Blattner and Tom Head. Automata That Recognize Intersections of Free Submonoids. *Information and Control*, 35(3):173–176, 1977.

8 Matthias Büchse and Anja Fischer. Deciding the Twins Property for Weighted Tree Automata over Extremal Semifields. In Frank Drewes and Marco Kuhlmann, editors, *Proceedings of the Workshop on Applications of Tree Automata Techniques in Natural Language Processing (ATANLP)*, pages 11–20. Association for Computational Linguistics, 2012.

9 Matthias Büchse, Jonathan May, and Heiko Vogler. Determinization of Weighted Tree Automata Using Factorizations. *Journal of Automata, Languages and Combinatorics*, 15(3/4):229–254, 2010.

10 Laure Daviaud, Pierre Guillon, and Glenn Merlet. Comparison of Max-Plus Automata and Joint Spectral Radius of Tropical Matrices. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

11 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, 2009.

12 Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. On delay and regret determinization of max-plus automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE Computer Society, 2017.

13 Kōsaburō Hashiguchi. Algorithms for Determining Relative Star Height and Star Height. *Information and Computation*, 78(2):124–169, 1988.

14 Kōsaburō Hashiguchi, Kenichi Ishiguro, and Shūji Jimbo. Decidability of The Equivalence Problem for Finitely Ambiguous Finance Automata. *International Journal of Algebra and Computation*, 12(3):445–461, 2002.

15 Daniel Kirsten. A Burnside Approach to the Termination of Mohri's Algorithm for Polynomially Ambiguous Min-Plus-Automata. *Informatique Théorique et Applications*, 42(3):553–581, 2008.

16 Daniel Kirsten. Decidability, undecidability, and PSPACE-completeness of the twins property in the tropical semiring. *Theoretical Computer Science*, 420:56–63, 2012.

17 Daniel Kirsten and Sylvain Lombardy. Deciding Unambiguity and Sequentiality of Polynomially Ambiguous Min-Plus Automata. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 589–600. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

18 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.

19 Jan Komenda, Sébastien Lahaye, Jean-Louis Boimond, and Ton van den Boom. Max-plus algebra in the history of discrete event systems. *Annual Reviews in Control*, 45:240–249, 2018.

20 Adam Koprowski and Johannes Waldmann. Max/Plus Tree Automata for Termination of Term Rewriting. *Acta Cybernetica*, 19(2):357–392, 2009.

21 Daniel Krob. The Equality Problem for Rational Series with Multiplicities in the tropical Semiring is Undecidable. *International Journal of Algebra and Computation*, 4(3):405–426, 1994.

22 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Springer, 1986.

23 Filip Mazowiecki and Cristian Riveros. Pumping Lemmas for Weighted Automata. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

24 Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, 1997.

25 Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 6, pages 213–254. Springer, 2009.

26 Erik Paul. The Equivalence, Unambiguity and Sequentiality Problems of Finitely Ambiguous Max-Plus Tree Automata are Decidable. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

27 Slav Petrov. Latent Variable Grammars for Natural Language Parsing. In *Coarse-to-Fine Natural Language Processing*, chapter 2, pages 7–46. Springer, 2012.

28 Michael O. Rabin and Dana S. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

29 Jacques Sakarovitch. A Construction on Finite Automata that has Remained Hidden. *Theoretical Computer Science*, 204(1-2):205–231, 1998.

30 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer, 1978.

31 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270, 1961.

32 Marcel-Paul Schützenberger. Sur les Relations Rationnelles Entre Monoïdes Libres. *Theoretical Computer Science*, 3(2):243–259, 1976.

33 Helmut Seidl. On the Finite Degree of Ambiguity of Finite Tree Automata. *Acta Informatica*, 26(6):527–542, 1989.

34 Imre Simon. Limited Subsets of a Free Monoid. In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–150. IEEE Computer Society, 1978.

35 Imre Simon. Recognizable Sets with Multiplicities in the Tropical Semiring. In Michal Chytil, Ladislav Janiga, and Václav Koubek, editors, *Mathematical Foundations of Computer Science (MFCS)*, volume 324 of *Lecture Notes in Computer Science*, pages 107–120. Springer, 1988.

36 Johannes Waldmann. Weighted Automata for Proving Termination of String Rewriting. *Journal of Automata, Languages and Combinatorics*, 12(4):545–570, 2007.

37 Andreas Weber and Helmut Seidl. On the Degree of Ambiguity of Finite Automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

Paging with Dynamic Memory Capacity

Enoch Peserico
Univ. Padova, Italy
enoch@dei.unipd.it

Abstract

We study a generalization of the classic paging problem that allows the amount of available memory to vary over time – capturing a fundamental property of many modern computing realities, from cloud computing to multi-core and energy-optimized processors.

It turns out that good performance in the "classic" case provides no performance guarantees when memory capacity fluctuates: roughly speaking, moving from static to dynamic capacity can mean the difference between optimality within a factor 2 in space and time, and suboptimality by an arbitrarily large factor. More precisely, adopting the competitive analysis framework, we show that some online paging algorithms, despite having an optimal $(h, k)-$competitive ratio when capacity remains constant, are not $(3, k)-$competitive for any arbitrarily large k in the presence of minimal capacity fluctuations.

In this light it is surprising that several classic paging algorithms perform remarkably well even if memory capacity changes adversarially – in fact, even without taking those changes into explicit account! In particular, we prove that LFD still achieves the minimum number of faults, and that several classic online algorithms such as LRU have a "dynamic" $(h, k)-$competitive ratio that is the best one can achieve without knowledge of future page requests, *even if one had perfect knowledge of future capacity fluctuations*. Thus, with careful management, knowing/predicting future memory resources appears far less crucial to performance than knowing/predicting future data accesses.

We characterize the optimal "dynamic" $(h, k)-$competitive ratio exactly, and show it has a somewhat complex expression that is almost but not quite equal to the "classic" ratio $\frac{k}{k-h+1}$, thus proving a strict if minuscule separation between online paging performance achievable in the presence or absence of capacity fluctuations.

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

Keywords and phrases paging, cache, adaptive, elastic, online, competitive, virtual, energy

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.56

Related Version Some of these results were presented at the poster session of SIGMetrics'13 [25]. An older version of this work can be found on arXiv [26].

Acknowledgements We are indebted to Federica Bogo and Michele Scquizzato for their relentless encouragement and a number of fruitful discussions on this topic; and to the anonymous referees of STACS 2019 for their insightful, in-depth review of our work.

1 Introduction

This work examines a generalization of the classic paging problem that allows the amount of available memory to vary over time. After briefly reviewing the paging problem (Subsection 1.1) this section motivates paging with dynamic capacity (Subsection 1.2) and provides an overview of our results and of the organization of the rest of the article (Subsection 1.3).

1.1 The paging problem

The memory/data storage system of modern computing devices is almost always organized as a hierarchy of several layers of progressively larger capacity but also higher access cost (in terms of both time and energy); efficiently orchestrating the flow of information across

the memory hierarchy is crucial for performance. The most widely used model for studying this *paging* problem is that of a two-layer system: a smaller *memory* layer with a capacity of k *pages* (data blocks), and a larger layer of infinite capacity whose pages can only be accessed by first copying them into memory – an operation termed a *(page) fault*. Given any sequence of pages that must be accessed in order, an algorithm for the paging problem must choose which page(s) to "evict" from memory, whenever a new page must be copied into it, so as to minimize the total number of faults.

The simple algorithm LFD (Longest Forward Distance) that evicts the page accessed furthest in the future has long been known to be optimal [4]. However, paging is often studied as an *online* problem, i.e. an algorithm can decide evictions only on the basis of past requests. A popular framework for evaluating the performance of online paging algorithms is that of *competitive analysis* [20]. A paging algorithm is said to have an $(h, k)-$*competitive ratio* of (no more than) ρ if, for every request sequence, it incurs in expectation with a memory of capacity k at most ρ times as many faults as an optimal offline algorithm incurs with a memory of capacity $h \leq k$, plus a number of faults independent of the request sequence. The ratio $\frac{k}{h}$ is called the *resource augmentation*. Resource augmentation and competitive ratio capture, respectively, the space and access cost overheads incurred by an online algorithm.

Many simple, deterministic algorithms including LRU[1], FIFO[2], FWF[3] and CLOCK[4] have an (h, k)-competitive ratio of $\frac{k}{k-h+1}$ [30, 8]; and the same ratio holds for RAND[5] [29]. This ratio is optimal for deterministic algorithms, and even for randomized ones if page requests can depend on previous choices of the paging algorithm (the "adaptive adversary" model [29] which we adopt throughout the article[6]). Since $\frac{k}{k-k/2+1} < 2$, many simple online algorithms never fare worse than the optimal offline algorithm would on a memory system with half the capacity and twice the access cost. This justifies the use of competitive analysis for preliminary performance evaluation of paging algorithms. Its "worst-case" approach may be somewhat pessimistic, but it is not overly so for many popular online paging algorithms – for which it provides guarantees of performance within a factor 2 of the optimal *under any workload* (in terms of faults and required memory capacity). In contrast, the finer granularity evaluation provided by experimental benchmarking is inevitably tied to specific workloads.

[8] and [10] provide two excellent surveys of the many variants of competitive analysis for the paging problem: these include somehow limiting the choice of the adversarial request sequence [9, 13, 14, 21], amortizing the performance evaluation over a spectrum of sequences [1, 2, 5] or of memory capacities [32], considering pages of different size and access cost [19, 32], and accounting for the non-zero cost of non-fault requests [31].

---

[1] Least Recently Used – evict the least recently accessed page.
[2] First In First Out – evict the page brought least recently into memory.
[3] Flush When Full – evict all pages whenever memory is full and space is needed.
[4] Mark any page accessed; to evict a page, cycle through pages, unmarking those found marked and evicting the first found unmarked.
[5] Evict a page chosen uniformly at random.
[6] More precisely, in the *online* adaptive adversary model, the choices of the reference offline algorithm can depend only on the *past* random choices of the online algorithm; in the *offline* adaptive adversary model they can depend on the random choices of the online algorithm over the entire request sequence. The bounds above – and in fact all the bounds in this article – hold for both models, with one exception: the upper bound on the competitive ratio of RAND above, and the corresponding upper bound we provide for RAND in Theorem 10, only hold in the adaptive online model.

1.2 Memory capacity often varies over time

In all variants of the paging problem, until a few years ago memory capacity was always fixed throughout the request sequence. This no longer reflects many computing realities. In a cloud computing environment, the amount of physical memory available to an individual virtual machine varies considerably over time depending on the virtual machine's load and on the number, load and relative class of service of other virtual machines hosted on the same hardware. Even on a simple PC, most modern operating systems have and use the option of declaring some critical virtual pages temporarily "unswappable", pinning them in main memory and thus reducing the amount of main memory available to user processes. Memory fluctuations also take place when considering the cache-RAM interface – in which case memory represents cache memory and pages represent cache lines. In many multi-core processor designs cache capacity is partitioned *dynamically* between different cores [28]. And low-power chip designs can often dynamically disable underutilized portions of the cache to save energy [18], again resulting in a capacity that can vary over time.

Note that, although there exists a large body of work on servicing the same request sequence with a policy that is simultaneously "good" on memories of different [32] and perhaps unknown [15] *but unchanging* capacity, allowing capacity to vary dynamically during the course of the computation is an entirely different problem; as we shall see, a solution to the former does not guarantee even an approximate solution to the latter.

A seminal work considering memory fluctuations in a hierarchical memory system is [3], that presents efficient algorithms for problems such as sorting or FFT assuming each algorithm can explicitly choose which pages to keep in memory. A slightly different approach is that taken by the recent literature on *cache-adaptive* algorithms for sorting, FFT and many other problems [6, 7], where management of the memory is devolved to an "automatic optimal" page replacement policy; the choice is justified by showing that a LRU policy is $O(1)$-competitive with $O(1)$ resource augmentation.

Our work instead focuses on the page replacement policy itself, assuming as in classic paging that the request sequence is provided by an adversary. The adversary also controls how memory capacity fluctuates between 1 and k pages; these fluctuations may be either known beforehand to the paging algorithm (an "offline" problem), or unknown until they take place (an "online" problem). We stress that, when comparing different paging algorithms, we take a slightly different approach from that of [7]: we assume identical/proportional memory capacity when servicing the same page request, whereas [7] assumes identical/proportional capacity after an identical/proportional number of faults. As [7] itself notes when referencing this work (the preliminary version of which predates [7]), both models are natural and reflect a different emphasis on what may initiate capacity changes.

Another related, but fundamentally different, paradigm is that of *RAM rental* [11, 23], where memory capacity fluctuates *under control of the paging algorithm* and the goal is to minimize a linear combination of average capacity and fault rate over time. In practice there are very strong constraints on the set of admissible capacity values, on how they can change over time, and on their relative costs (which may themselves fluctuate). Also, a number of architectural approaches (e.g. [12]) decouple the portion of the system responsible for page replacement from that responsible for capacity allocation. Then, assuming as we do that capacity fluctuations are not controlled by the paging algorithm (in fact, that they may be unknown beforehand and even chosen adversarially) leads to a more robust evaluation of page replacement policies.

1.3 Our results

The rest of this article is organized as follows. Section 2 introduces some formalism and terminology. In particular, it extends the notion of "online vs. offline" problem to encompass the extra dimension of future memory capacity, and it extends the notion of $(h, k)-$resource augmentation to the dynamic capacity scenario (in a nutshell, restricting the offline algorithm to at most a fraction $\frac{h}{k}$ of the online algorithm's *current* memory capacity).

Section 3 shows the existence of online paging algorithms that have an (optimal) (h, k)-competitive ratio of $\frac{k}{k-h+1}$ in the "classic" paging model, and yet are no longer $(3, k)-$competitive *for any arbitrarily large* k if their memory capacity is subject to single page fluctuations. This very negative result provides strong justification for our inquiry, as one cannot infer performance in the presence of (even minimal) memory fluctuations from performance in their absence.

In this light, it is surprising that many well-known algorithms perform remarkably well in the presence of memory fluctuations *even if those fluctuations are chosen adversarially.* In Section 4 we show that the classic LFD algorithm remains strictly optimal for all possible memory capacity fluctuations even though it does not explicitly take those fluctuations into account (i.e. it is an online algorithm in terms of memory fluctuations). We also show that in the dynamic capacity framework every online algorithm that is either marking [14] like LRU, FWF or MARK[7], or *dynamically conservative* (a simple refinement of the notion of "conservative algorithm"[33]), like LRU, CLOCK or FIFO, has an $(h, k)-$competitive ratio no larger than $\rho_{EL}(h, k) = \max_{k' \leq k, k' \in \mathcal{N}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$[8]. Exactly the same bound holds for RAND (against an online adaptive adversary).

Section 5 analyses $\rho_{EL}(h, k)$. We show that it is a lower bound to the $(h, k)-$competitive ratio achievable by any online paging algorithm in the presence of memory fluctuations, proving the optimality of marking and dynamically conservative algorithms. We also show that $\rho_{EL}(h, k)$ almost, but not quite, matches the "classic" bound of $\frac{k}{k-h+1}$ on the $(h, k)-$competitive ratio. More precisely, $\rho_{EL}(h, k)$ is always less than $(1 + \frac{1}{k})$ times the classic $\frac{k}{k-h+1}$ ratio – and if $h > k - \sqrt{k}$ the two quantities actually coincide. However, $\rho_{EL}(h, k)$ is also at least $1 + (\frac{1}{k} - \frac{2}{k^2})$ times as large as $\frac{k}{k-h+1}$ for any odd h and $k = 2h$, which proves a strict if minuscule separation between performance achievable in the presence and absence of memory fluctuations.

Section 6 briefly looks at the implications of our results for the RAM rental problem. In a nutshell, since many simple replacement are near optimal regardless of capacity fluctuations, RAM rental is simplified into the problem of just choosing a "good" capacity sequence without worrying about replacement.

Finally, Section 7 summarizes our results and looks at their significance and at possible directions of future work.

2 Some formalism/terminology

We can easily extend the notion of request sequence $\sigma = r_1, r_2, \ldots$ to the case of memory fluctuations. We simply assume that, interleaved with standard page requests, it is possible to have two additional types of requests, *growths* and *shrinks*. On a growth, memory capacity increases by 1 page; on a shrink, it decreases by 1 – and if the memory was full a page must

[7] Mark any page accessed; evict a random unmarked page, first unmarking all pages if all are marked.
[8] The "EL" in ρ_{EL} stood for "elastic" in an early, poster version of this work titled "Elastic paging" [25].

be evicted. We assume that initially memory capacity is 0. Throughout the rest of the article, we denote a growth request by the symbol + and a shrink request by the symbol −, and we denote k consecutive growths / shrinks by $+^k$ and $-^k$. Thus, a standard request sequence $p_1, \ldots, p_n$ on a memory of capacity k simply becomes $+^k, p_1, \ldots, p_n$ in the more general dynamic capacity framework.

The request sequence automatically induces a *page sequence* $\pi =< p_1, p_2, \cdots >$ (the sequence of requested pages $p_1, p_2, \ldots$, as in the classic paging problem) and a *capacity sequence* $\mu = m_1, m_2, \ldots$ where m_i is the memory capacity immediately before the request for p_i (i.e. it is equal to the number of +s minus the number of −s in the request prefix ending with p_i). Note that the presence of growths and shrinks introduces a second aspect of "onlineness". More formally:

▶ **Definition 1.** *A paging algorithm ALG is online relative to the page sequence if its eviction choices before servicing a request are independent of any future page requests; otherwise it is offline relative to the page sequence. Similarly, ALG is online relative to the capacity sequence if its eviction choices before servicing a request are independent of any subsequent growths and shrinks; otherwise it is offline relative to the capacity sequence. ALG is a fully online, partially offline and fully offline paging algorithm if it is online relative to (respectively) both, one, or neither of the page and the capacity sequence.*

Thus, in the dynamic capacity model, all well-known paging algorithms such as LRU, FIFO, FWF, CLOCK, RAND and MARK are fully online, and LFD is partially offline, being offline relative to the page sequence but online relative to the capacity sequence.

We can easily extend the notion of $(h, k)-$competitive ratio to the dynamic capacity model by comparing the cost (i.e. number of faults) incurred by an online algorithm whose memory capacity never exceeds k to the cost incurred by an offline algorithm whose memory capacity never exceeds $\frac{h}{k}$ times that of the online algorithm. More formally, denote by OPT the optimal offline algorithm, and by $c_{ALG}(\pi, \mu)$ the cost incurred by an algorithm ALG when servicing a page sequence $\pi = p_1, \ldots, p_n$ with a capacity sequence $\mu = m_1, \ldots, m_n$. Also, given a capacity sequence $\mu = m_1, \ldots, m_n$ and a non-negative number a, denote by $\lfloor a \cdot \mu \rfloor$ the capacity sequence $m'_1, \ldots, m'_n$ with $m'_i = \lfloor a \cdot m_i \rfloor$. Then:

▶ **Definition 2.** *A paging algorithm ALG has a* dynamic $(h, k)-$*competitive ratio of (at most) ρ if there exists some constant d such that, for any page sequence $\pi = p_1, \ldots, p_n$ and any capacity sequence $\mu = m_1, \ldots, m_n$ such that, $\forall i$, $m_i \leq k$:*

$$c_{ALG}(\pi, \mu) \leq \rho \cdot c_{OPT}(\pi, \lfloor \frac{h}{k} \cdot \mu \rfloor) + d$$

Note that the dynamic $(h, k)-$competitive ratio of an algorithm is always an upper bound to its $(h, k)-$competitive ratio. Thus online paging with dynamic capacity is in some sense "harder" than classic online paging, and no online algorithm can have a dynamic $(h, k)-$competitive ratio lower than the "classic" ratio $\frac{k}{k-h+1}$.

3 Minimal capacity fluctuations can lead to arbitrarily large performance degradation

This section shows that there exist online paging algorithms *that do not depend explicitly on memory capacity*, and that have an optimal (h, k)-competitive ratio in the classic setting of fixed memory capacity, but are not competitive at all, even with arbitrary resource augmentation, when faced with even slight fluctuations in memory capacity. Consider the

online paging algorithm LFRU (Least Frequently / Recently Used) that starts as LRU and then alternates between LFU and LRU – switching from LRU to LFU after any palindrome subsequence incurring more faults in its second half, and switching from LFU to LRU after any palindrome subsequence incurring more faults in its first half:

Algorithm 1 LFRU: service $p_0, \ldots, p_n$ as follows.

at p_0 POLICY $\leftarrow$ LRU

for $i = 1 \ldots n$ **do**

if at p_i POLICY = LRU AND $\exists j < i$:
$< p_j \ldots p_i >$ is palindrome AND faults$(p_j \ldots p_{\lfloor \frac{i+j}{2} \rfloor}) <$ faults$(p_{\lceil \frac{i+j}{2} \rceil} \ldots p_i)$
then at p_{i+1} POLICY $\leftarrow$ LFU

else if at p_i POLICY = LFU AND $\exists j < i$:
$< p_j \ldots p_i >$ is palindrome AND faults$(p_j \ldots p_{\lfloor \frac{i+j}{2} \rfloor}) >$ faults$(p_{\lceil \frac{i+j}{2} \rceil} \ldots p_i)$
then at p_{i+1} POLICY $\leftarrow$ LRU

end for

We would convince the reader that LFRU, while undoubtedly artificial and difficult to implement in practice, is not too different from many real-world paging heuristics designed for static memory capacity (note that the behaviour of LFRU, like that of LRU and LFU, does not depend explicitly on memory capacity). In fact, pure LRU tends to be outperformed in practice by various LRU/LFU hybrids [22, 24]. The main reason is the common coexistence of "local" or "temporal" computations sporting a high degree of temporal locality and data reuse, and "streaming" computations that access long sequences of sequential data without any temporal locality. In such cases, under LRU and similar policies such as CLOCK, streaming data not only gain no benefit from being kept in the fast memory layer (since every new access is a fault) but actively *pollute* it, forcing the eviction of temporal data and preventing the temporal computation from deriving more than a minimal benefit from the fast memory layer. One possible solution is to combine LRU with eviction schemes biased, like LFU, against data that have no reuse history even if their last (and only) access was very recent. And since LRU performs best when future requests are a "mirror image" of the past, it may seem reasonable to switch to it when such palindrome sequences sport good data-reuse behaviour, and switch to LFU when such palindrome sequences exhibit sport poor data-reuse behaviour – which is what LFRU does.

It turns out that LFRU has an optimal $(h, k)-$competitive ratio in the classic paging model where memory capacity is fixed. At the same time, even if faced with capacity fluctuations of just a single page, and even if allowed the use of an arbitrarily large amount of memory, LFRU's fault rate can be arbitrarily larger than that of an offline algorithm running with just 3 pages of memory. More formally we prove:

▶ **Theorem 3.** *LFRU has an $(h, k)-$competitive ratio equal to $\frac{k}{k-h+1}$ if memory capacity is constant, but has no finite dynamic $(h, k)-$competitive ratio for any $h \geq 3$ and any arbitrarily large k.*

Proof. Let us first prove that LFRU has an $(h, k)-$competitive ratio equal to $\frac{k}{k-h+1}$ if memory maintains an arbitrary but fixed capacity k. We need only prove that, as long as LFRU keeps behaving as LRU, on no page request sequence a palindrome subsequence incurs more faults in its second half: then LFRU keeps behaving exactly as LRU and shares its (h, k)-competitive ratio of $\frac{k}{k-h+1}$.

Consider a palindrome page subsequence $\pi = p_{i_1}, \ldots, p_{i_\ell}, p_{i_\ell}, \ldots, p_{i_1}$ of even length 2ℓ, containing $\lambda \leq \ell$ distinct pages $p_1, \ldots, p_\lambda$. Note that, if $\lambda \leq k$, by the end of the first half of π the λ most recently requested pages are $p_i, \ldots, p_\lambda$, which are then in memory and prevent any fault from taking place during the second half of π. Then, we need only consider the case $\lambda > k$.

Let us focus on the first half of π. For each distinct page, we analyse separately the first request to it (which we call a *cold* request), and the remaining requests, if any (which we call *hot* requests). The i^{th} cold request is certainly a fault for any $i > k$, since at least k distinct pages have been requested before it in $p_{i_1}, \ldots, p_{i_\ell}$; so the number of cold requests incurring faults is at least $\ell - k$. Let us now look at hot requests, and let r_i be the number of hot requests of p_i in the first half of π. For $1 \leq i \leq \lambda$ and $1 \leq j \leq r_i$, let D_i^j be the set of distinct pages requested between the j^{th} hot request for p_i and the previous request for p_i, inclusive (so D_i^j always includes p_i). Then the j^{th} hot request to p_i is a fault if and only if $|D_i^j| > k$, and the total number of faults in $p_{i_1}, \ldots, p_{i_\ell}$ is:

$$f_\pi^{\frac{1}{2}} \geq (\ell - k) + |\{(i,j) : |D_i^j| > k\}| \tag{1}$$

Let us now focus on the second half of π. Again, we divide requests for any distinct page into a cold request (the first) and hot requests (subsequent ones, if any). The first k cold requests of $p_{i_\ell}, \ldots, p_{i_1}$ are for the last k distinct pages requested in $p_{i_1}, \ldots, p_{i_\ell}$, which are then present in memory at the beginning of $p_{i_\ell}, \ldots, p_{i_1}$. So in $p_{i_\ell}, \ldots, p_{i_1}$ none of the first k cold requests incurs a fault, yielding and at most $\lambda - k$ faults on cold requests. Let us now look at the hot requests of $p_{i_\ell}, \ldots, p_{i_1}$; those for p_i are obviously r_i, as in the first half of π. For $1 \leq i \leq \lambda$ and $1 \leq j \leq r_i$, let $\bar{D}_j^i$ be set of distinct pages between the j^{th} hot request for p_i and its previous request, including p_i itself; then the j^{th} hot request for p_i is a fault if and only if $|\bar{D}_i^j| > k$, and the total number of faults in $p_{i_\ell}, \ldots, p_{i_1}$ is:

$$\bar{f}_\pi^{\frac{1}{2}} \leq (\ell - k) + |\{(i,j) : |\bar{D}_i^j| > k\}| \tag{2}$$

It is crucial to observe that, since π is palindrome, $\bar{D}_j^i = D_{r_i - j + 1}^i$. Then $|\{(i,j) : |\bar{D}_i^j| > k\}| = |\{(i,j) : |D_i^j| > k\}|$ and $f_\pi^{\frac{1}{2}} \geq \bar{f}_\pi^{\frac{1}{2}}$. The analysis is virtually identical for palindrome subsequences of odd length; and thus with static memory capacity LFRU incurs no more faults on the second half of any palindrome subsequence than in the first half and has an $(h,k)-$competitive ratio equal to $\frac{k}{k-h+1}$.

To prove that LFRU can incur arbitrarily more faults than an optimal offline algorithm OPT when memory capacity fluctuates – even if OPT is limited to a capacity fluctuating between capacity 3 and 2, while LFRU's fluctuates between $3m$ and $3m-1$ for an arbitrarily large m – we show how LFRU can be coaxed into, and kept in, LFU behaviour, and how that behaviour can result in arbitrarily more faults than OPT even with arbitrarily larger capacity.

Denoting by r^m the concatenation of m consecutive copies of r, consider a page sequence formed by a prefix $\pi_1 =< p_1, p_2, p_3^\ell \ldots, p_{3m}^\ell, p_1, p_2, p_3, \ldots, p_{3m}, p_{3m}, \ldots, p_2, p_1 >$ followed by a suffix $p_2 = (p_2, p_1)^{\ell-1}$. Assume LFRU's memory capacity while servicing π_1 remains equal to $3m$ except for the last $3m$ requests, during which it drops by 1 to $3m-1$; and assume it then remains equal to $3m-1$ while servicing π_2. It is immediate to see that when capacity drops p_1 is evicted, and that the last $6m$ requests of π_1 form a palindrome subsequence experiencing a fault (only) on the last request. Thus, on that request, LFRU switches to LFU behaviour, and evicts p_2 (which, like p_1, has experienced $\ell - 1$ fewer requests than every other page p_i, $i \geq 3$). Then LFRU services $\pi_2 = (p_2, p_1)^{\ell-1}$ with memory capacity $3m-1$

by alternatively evicting p_2 and p_1 in turn, since p_1 and p_2 remain until the end of π_2 the two pages having experienced the fewest requests so far; thus LFRU incurs at least $2\ell - 2$ faults to service π_2.

An optimal algorithm (or even just LRU) with memory capacity 3 throughout all but the last $3m$ requests of π_1, and with memory capacity 2 thereafter, would instead incur no more than $3m + 3m + 3m = 9m$ faults during π_1, and no faults at all during π_2 (retaining only p_1 and p_2 in memory). Thus, since ℓ can be chosen arbitrarily larger than m, LFRU cannot have a finite $(3, 3m)-$competitive ratio for any arbitrarily large m. ◀

4 Dealing with adversarial fluctuations efficiently – and "implicitly"

In the light of Theorem 3 it may be surprising that many well-known "good" paging algorithms still perform remarkably well in the dynamic capacity setting – even though they do not take memory fluctuations into explicit account. It is very easy to prove:

▶ **Theorem 4.** *LFD incurs the minimal number of faults on any request sequence.*

Proof. We can safely ignore algorithms leaving unoccupied space in memory after an eviction, as such an eviction could be delayed without incurring additional faults. Let a page be *close* if it will be accessed before another page currently in memory, *far* otherwise. LFD is the algorithm evicting no close pages. We prove the theorem showing that one can always eliminate the earliest close eviction without altering previous evictions or increasing the number of faults.

Let p be the close page evicted earliest, at time t, by an algorithm ALG servicing a request sequence. Consider the algorithm $\overline{ALG}$ that operates as ALG until t, when it instead evicts a far page $\overline{p}$, and then operates as follows. Denote by M and $\overline{M}$ the sets of pages respectively in ALG's and $\overline{ALG}$'s memory. When both ALG and $\overline{ALG}$ must incur an eviction, $\overline{ALG}$ evicts the same page as ALG if possible; otherwise, when $\overline{ALG}$ must incur an eviction, it evicts a page not in M (as soon as $\overline{M} = M$, ALG and $\overline{ALG}$ coincide). After t, let t' be the time of the first request or eviction of either p or $\overline{p}$. Until t' ALG and $\overline{ALG}$ incur exactly the same faults and evictions, and thus $M \setminus \overline{M} = \{\overline{p}\}$ and $\overline{M} \setminus M = \{p\}$. At t' $\overline{ALG}$ evicts p if and only if ALG evicts $\overline{p}$ – in which case M and $\overline{M}$ converge. Otherwise p is requested at t' and ALG, but not $\overline{ALG}$, incurs a fault; and since $\overline{ALG}$ never evicts a page unless ALG also has evicted it, $|\overline{M} \setminus M|$ never increases after t, and drops to 0 no later than the first fault incurred by $\overline{ALG}$ and not by ALG. In both cases $\overline{ALG}$ incurs no more misses than ALG. ◀

It is interesting to note that Theorem 4 yields as an immediate corollary Theorem 4.1 in [17] – in a nutshell, for a given, dynamically changing, partition of the memory space between different processes, using LFD for each process on its own partition yields the minimum *total* number of faults. It is not, however, immediately obvious that the result in [17] implies the thesis of Theorem 4. Furthermore, the result in [17] is only stated, and not proved – the proof is deferred to the full version of the article because of its complexity compared to the "classic" proof of LFD's optimality.

Let us now focus on *online* paging algorithms. It turns out that the dynamic $(h, k)-$competitive ratio achievable by many well-known online algorithms is almost, but not quite, as good as the "plain" $(h, k)-$competitive ratio $\frac{k}{k-h+1}$ – and in particular equal to:

$$\rho_{EL}(h, k) = \max_{k' \leq k, k' \in \mathcal{N}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$$

The formula for ρ_{EL} is more complex, but vaguely reminiscent of the formula for the "classic" (h,k)-competitive ratio; and indeed it is easy to verify that for $h=k$ both equal k. A detailed analysis of the behaviour of ρ_{EL}, including a proof that it is a lower bound on the dynamic $(h,k)-$competitive ratio of any online algorithm, can be found in the following Section 5. The remainder of this section is devoted to proving that a dynamic $(h,k)-$competitive ratio $\rho_{EL}(h,k)$ is indeed achieved by all marking algorithms[9] (including MARK, LRU and FWF), by RAND, and by all *dynamically conservative* algorithms. The latter form a class of algorithms that is slightly narrower than that of conservative algorithms[10][33] but still includes LRU, FIFO and CLOCK. The cornerstone of the analysis lies in the notion of *short subsequence*, which is the "correct" extension of the concept of $k-$phase to dynamic capacity:

▶ **Definition 5.** *Given a generic (sub)sequence of consecutive requests, its* width *is the number of distinct pages in it.*

▶ **Definition 6.** *Consider a generic request sequence σ, and a subsequence σ' of consecutive requests in σ (including page requests, growths and shrinks). σ' is* short *if, for every prefix π of σ', the width of π does not exceed the memory capacity* at the end *of π.*

▶ **Definition 7.** *A* dynamically conservative *algorithm never incurs more than w faults on any short subsequence of width w.*

Note that a dynamically conservative algorithm is also always a conservative algorithm according to the definition of [33] since with a memory of fixed capacity k every short subsequence involves access to at most k pages, and thus incurs at most k faults. The converse is not true: LFRU from Section 3 is conservative but not dynamically conservative. However, we can easily prove:

▶ **Theorem 8.** *LRU, FIFO and CLOCK are dynamically conservative.*

Proof. It is not difficult to verify that all three algorithms have the following property: if a page p is brought into memory at time t, and a page p' already in memory at time t and is never accessed again, then p' will be evicted before p. This holds for LRU because p is more recently accessed than p'. It holds for FIFO because p' entered the memory before p. It holds for CLOCK because after t the unmark/evict process will encounter p' before encountering p – thus either evicting or at least unmarking p' before unmarking p, and thus certainly evicting it before evicting p. Then none of the three algorithms evicts a page accessed during a short sequence before the end of the sequence (since there is always sufficient memory to hold all pages accessed during the sequence), and thus none can incur more faults than the width of the sequence. ◀

The main result of this section is then:

▶ **Theorem 9.** *The dynamic $(h,k)-$competitive ratio of any online paging algorithm that is either marking or dynamically conservative is no larger than $\rho_{EL}(h,k)$.*

Proof. Let us begin with marking algorithms. The proof bears some resemblance to that of the static case, with a number of subtle but profound differences. One such difference is that, instead of partitioning the request sequence into maximal length phases each involving

[9] A marking algorithm marks a page in memory whenever it accesses it, never evicts a marked page, and unmarks all pages if all are marked and one must be evicted (e.g. in response to a fault or a shrink).

[10] A conservative algorithm never incurs more than k faults on a sequence of accesses involving at most k distinct pages and a memory of capacity k.

access to k distinct pages, we partition it into maximal short sequences $\pi_1, \ldots, \pi_n$ where π_i is the longest short sequence beginning immediately after the end of π_{i-1}. Furthermore, the strategy of analysing each short subsequence in isolation does not work, and we can only bound the ratio over the whole sequence, through careful accounting and a potential argument.

Denote by w_i the width of π_i. We can assume without loss of generality that the request sequence ends with a page request, so $w_i > 0 \ \forall i$. Note that, for $i > 1$, the first request $\pi_{i,1}$ of π_i must be either a shrink or a request for a page not in π_{i-1}; in the first case we say that π_{i-1} is *capacity bound*, in the second that it is *page bound*.

It is easy to verify by simultaneous induction the following claims hold for all i:

1. All pages in memory are unmarked when $\pi_{i,1}$ is serviced.
2. Every one of the w_i pages accessed during π_i (and no other page) remains marked and thus in memory until the end of π_i.
3. Immediately before $\pi_{i+1,1}$ is serviced, the memory is full and holds w_i pages, all marked.

Claim 1 holds trivially for $i = 1$. If Claim 1 holds for i, Claim 2 also holds for i, since until the end of π_i the memory is large enough to accommodate all pages accessed so far during π_i, which are the only ones marked. If Claim 2 holds for i, Claim 3 also holds for i, since immediately before $\pi_{i+1,1}$ is serviced the memory capacity exactly matches the number of distinct pages accessed in π_i. If Claim 3 holds for i, Claim 1 holds for $i + 1$ (proving the inductive step), since the first request of π_{i+1} must be either a shrink or a request for a page not in π_i, and thus causes all pages in memory to become unmarked.

From Claim 2 it is obvious that a marking algorithm incurs a number of faults at most equal to w_i during short sequence π_i, for a total number of faults equal to at most:

$$c_{ALG} \leq \sum_{i=1}^{n} w_i \tag{3}$$

Let us compute the number of faults incurred by any other algorithm $\overline{ALG}$ with a memory of capacity at most $\frac{h}{k}$ times that of the marking algorithm, in the interval π'_i from immediately after the first request σ of π_i is serviced, to immediately after the first request of π_{i+1} is serviced or to the end of the request sequence if $i = n$. Let r_i be equal to 1 if π_i is page bound, and to 0 if it is capacity bound, for $1 \leq i < n$, and let $r_0 = 0$ and $r_n = 0$. Remember that the first request of a short phase π_i is a shrink if π_{i-1} is capacity bound, and a page not in π_{i-1} if π_{i-1} is page bound – and a growth if $i = 1$. Denoting by w'_i the number of distinct pages in π'_i after removing the page involved in the first request of π_i if any (i.e. if π_{i-1} is page bound), we can then write for $1 \leq i \leq n$:

$$w'_i = -r_{i-1} + w_i + r_i \tag{4}$$

The subset of these pages in the memory of $\overline{ALG}$ immediately before servicing the first request of π'_i is then at most:

$$m_i = \begin{cases} 0 & \text{if } i = 1, \\ \lfloor \frac{h}{k}(w_{i-1} - 1) \rfloor & \text{if } i > 1. \end{cases} \tag{5}$$

Equation 5 is immediate if $i = 1$ or if π_{i-1} is capacity bound - since then the first request of π_i shrinks the memory available to ALG from w_{i-1} to $w_{i-1} - 1$. If instead π_{i-1} is page bound, of the $\lfloor \frac{h}{k} w_{i-1} \rfloor$ pages $\overline{ALG}$'s memory can hold, one must be the first

page of π_i that has just been requested and that does not contribute to m_i – leaving only $\lfloor \frac{h}{k} w_{i-1} \rfloor - 1 \leq \lfloor \frac{h}{k}(w_{i-1}-1) \rfloor$. Then the total number of faults incurred by $\overline{ALG}$ is at least:

$$\sum (w'_i - m_i) \geq \sum_{i=1}^{n} (-r_{i-1} + w_i + r_i) - \sum_{i=2}^{n} \lfloor \frac{h}{k}(w_{i-1}) - 1) \rfloor \geq \sum_{i=1}^{n} (w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor) \quad (6)$$

Remembering that both w_i and $w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor$ with $h \leq k$ are positive, and that $\forall a, b, c, d > 0$ we have that $\frac{a+b}{c+d} = \frac{c}{c+d} \cdot \frac{a}{c} + \frac{d}{c+d} \cdot \frac{b}{d} \leq \max(\frac{a}{c}, \frac{b}{d})$, the dynamic $(h,k)-$competitive ratio of ALG is at most:

$$\frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} (w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor)} \leq \max_i \frac{w_i}{w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor} \leq \max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \quad (7)$$

This proves the theorem for marking algorithms. The proof for dynamically conservative algorithms proceeds identically, except for the fact that in this case one can immediately obtain, from Definition 7, the bound given by Equation 3 on the cost incurred by the online algorithm. ◀

The analysis of RAND faces similar difficulties similar to those in the proof of Theorem 9 in terms of "compartimentalization of costs"; but they can be addressed in a different way due to the randomized nature of the algorithm, by exploiting its lack of memory. In this sense it may be somewhat surprising that *exactly* the same bound obtained in Theorem 9 also applies to RAND, particularly because in the (very slightly different) cache-adaptive model RAND is not competitive at all, regardless of resource augmentation[11]. Instead, we prove:

▶ **Theorem 10.** *RAND's dynamic $(h,k)-$competitive ratio is no larger than $\rho_{EL}(h,k)$ in the adaptive online adversary model.*

Proof. We cannot apply the "classic" paging analysis of RAND due to the fact that, if $h < k$, cache shrinks may not be "synchronized" and RAND may incur shrinks when the optimal offline algorithm OPT does not. Instead of comparing the number of faults c_{RAND} and c_{OPT} incurred, respectively, by RAND and OPT, we then begin by comparing the number of *page evictions* e_{RAND} and e_{OPT}. For simplicity, assume that, after any given request (for a page, or for a capacity change), the request is served in the following order. OPT performs any page eviction; then it loads into memory any requested page not yet there; then RAND does the same; finally, OPT adjusts its memory capacity, and then RAND does the same. Note that since capacity is adjusted one page at a time (see Section 2), evictions are always performed one page at a time.

Let the *garbage* of RAND at any given point in time be the set G of pages in its memory and not in the memory of OPT. First of all, note that G can increase only when OPT incurs an eviction (and at most by 1 page for each eviction), since RAND never brings into memory a page not requested by OPT – which at that point must then be in OPT's memory.

[11] This can be easily seen by the reader familiar with the cache-adaptive model. Consider, for an arbitrarily large $c \geq 2$, the memory sequence $p_{2c}, \ldots, p_0, (p_1, p_0)^{3c^2}$. OPT can service the sequence with memory 2 incurring only $2c+1$ faults. RAND with memory $2c$ has a non-zero probability of evicting p_1 on every request for p_0 and viceversa, and thus incurring more than $3c^2 > (2c+1)c$ faults – so in the worst case it completes the sequence after OPT even with the advantage of $c-$speed and $c-$memory augmentation. While this simple example leverages the worst-case accounting of the standard cache-adaptive model, even more sophisticated accounting "in expectation" faces similar difficulties – informally because in the cache-adaptive model even a minuscule probability of missing a shrink deadline can be catastrophic.

Immediately before RAND incurs an eviction, its memory must be full; denote by k' and h' the memory capacity of RAND and OPT at that point. If the eviction is the result of a shrink, then the number of pages in RAND's memory that are *not* garbage are at most:

$$h' = \lfloor \frac{h}{k}(k'-1) \rfloor \tag{8}$$

Note that at this point OPT has adjusted its memory capacity to the shrink but RAND has not. If the eviction is the result of a page fault, then the requested page at this point is in OPT's memory but not in RAND's, and the number of pages in RAND's memory that are *not* garbage are at most:

$$h' - 1 = \lfloor \frac{h}{k}k' \rfloor - 1 \leq \lfloor \frac{h}{k}(k'-1) \rfloor \tag{9}$$

Thus the probability that, when RAND incurs an eviction, $|G|$ decreases by 1 is at least:

$$p_{RAND} = \min_{k' \in \{1,\ldots,k\}} \frac{k' - \lfloor \frac{h}{k}(k'-1) \rfloor}{k'} \tag{10}$$

and at any given time we have that, in expectation:

$$|G| \leq e_{OPT} - e_{RAND} \cdot p_{RAND} \tag{11}$$

Appending to any request sequence sufficient shrinks to bring $RAND$'s memory capacity to 0 obviously brings $|G|$ to 0, without increasing the number of *faults* incurred by $RAND$ or OPT. For any algorithm that evicts a single page at a time, when the memory holds no pages the number of faults and evictions incurred must coincide. Setting $|G|$ to 0, as well as $c_{OPT} = e_{OPT}$ and $c_{RAND} = e_{RAND}$, in Equation 11 then yields for RAND a dynamic $(h,k)-$competitive ratio equal at most to:

$$\frac{c_{RAND}}{c_{OPT}} \leq \frac{1}{p_{RAND}} = \max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \tag{12}$$

◀

5 An exact characterization of the competitive ratio

The upper bound $\rho_{EL}(h,k)$ obtained in Section 4 for the dynamic $(h,k)-$competitive ratio of many online paging algorithms is actually tight. In fact, no paging algorithm that is online relative to the page request sequence can achieve a better $(h,k)-$competitive ratio, *even if it has from the start full knowledge of the entire capacity sequence.* More formally we can prove:

▶ **Theorem 11.** *No paging algorithm that is online relative to the page sequence has a dynamic $(h,k)-$competitive ratio (against any online or offline adaptive adversary if randomized) lower than:*

$$\rho_{EL}(h,k) = max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$$

Proof. Let $\overline{k} = argmax_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$, and let ALG be a generic paging algorithm online relative to the page sequence. Consider a request sequence $\sigma_n =< +^{(\overline{k}-1)}, \pi_1, \ldots, \pi_n >$, where:

$$\pi_i =< +^{(k-\overline{k}+1)}, p_{i,1}, -^{(k-\overline{k}+1)}, \ldots, +^{(k-\overline{k}+1)}, p_{i,\overline{k}}, -^{(k-\overline{k}+1)} > \tag{13}$$

and $p_{i,j}$ is any one page, from the set $p_1, \ldots, p_{\overline{k}}$, that is not in ALG's memory just before it is requested – note that immediately before any page request ALG's memory holds at most $\overline{k}-1$ pages, so there always exists one such page. ALG then incurs a fault on every page request, for a total number of faults equal to:

$$c_{ALG}(\sigma_n) = n \cdot \overline{k} \tag{14}$$

Also, note that the capacity sequence associated to σ_n does not depend on ALG, so we can freely assume that the design of ALG incorporates full knowledge of it.

Consider an offline algorithm $\overline{ALG}$ with access to a memory that has at most $\frac{h}{k}$ times the capacity of ALG's at any given time; in particular, $\overline{ALG}$'s memory capacity grows to h immediately before any page request, and immediately afterwards drops to capacity:

$$\lfloor \frac{h}{k}(\overline{k}-1) \rfloor = \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor < h \tag{15}$$

$\overline{ALG}$ can easily maintain in its "permanent" $\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor$ memory locations the $\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor$ pages with most *expected* accesses in σ_n, incurring for each only one initial fault. Note that the total number of accesses to these pages is, in expectation, at least $n\overline{k} \cdot \frac{\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor}{\overline{k}} = n\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor$. Every other page, when requested, is brought into the "temporary" location(s) immediately eliminated by the following shrink. $\overline{ALG}$ then incurs an expected number of faults equal to:

$$c_{\overline{ALG}}(\sigma_n) \leq \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor + n(\overline{k} - \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor) \tag{16}$$

Then the competitive ratio of ALG can be no lower than:

$$\lim_{n\to\infty} \frac{c_{ALG}(\sigma_n)}{c_{\overline{ALG}}(\sigma_n)} = \lim_{n\to\infty} \frac{n\overline{k}}{\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor + n(\overline{k} - \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor)} = \max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \tag{17}$$

It is important to observe that, if ALG is randomized, $\overline{ALG}$ need only know ALG's probabilistic behaviour to choose which pages to keep in its own memory; and it can choose which page to request next based only on ALG's current memory contents. Thus the lower bound we proved holds for deterministic and randomized algorithms both in the adaptive offline and in the adaptive online adversary models. ◀

As noted in Section 4 the expression of the optimal dynamic (h,k)-competitive ratio $\rho_{EL}(h,k)$ appears considerably more complex than, but vaguely reminiscent of, that of the "classic" bound on the $(h,k)-$competitive ratio, $\frac{k}{k-h+1}$. It is natural to ask whether the two are actually different, and if so to what extent. We show that $\rho_{EL}(h,k)$ *is*, in fact, a factor $\approx 1+\frac{1}{k}$ larger for some "natural" values of h and k – though it is never more than a factor $1+\frac{1}{k}$ larger, and actually coincides with $\frac{k}{k-h+1}$ if h is equal or very close to k. This is stated more formally in the following two theorems:

▶ **Theorem 12.** *For any odd h and $k = 2h$, $\rho_{EL}(h,k) \geq (1+\frac{1}{k}-\frac{2}{k^2})\frac{k}{k-h+1}$.*

Proof. For any integer $i \geq 0$, choosing $h = 2i+1$, $k = 2h$, and $k' = k-1$, we obtain immediately:

$$\begin{aligned}\rho_{EL}(h,k) &\geq \frac{4i+1}{4i+1-\lfloor (2i+1)\frac{4i+1}{4i+2} - \frac{2i+1}{4i+2} \rfloor} = \frac{4i+1}{4i+1-\lfloor \frac{4i+1}{2} - \frac{1}{2} \rfloor} = \frac{4i+1}{2i+1} \\ &= \frac{k-1}{\frac{k}{2}} = \frac{k-1}{\frac{k}{2}} \cdot \frac{\frac{k}{2}+1}{k} \cdot \frac{k}{k-h+1} = (1+\frac{1}{k}-\frac{2}{k^2})\frac{k}{k-h+1}\end{aligned} \tag{18}$$

◀

▶ **Theorem 13.** $\frac{k}{k-h+1} \leq \rho_{EL}(h,k) < (1+\frac{1}{k})\frac{k}{k-h+1}$ *for all* $h,k \in \mathbb{Z}^+$ *with* $h \leq k$, *and* $\rho_{EL}(h,k) = \frac{k}{k-h+1}$ *if* $k \geq h > k - \sqrt{k}$.

Proof. It is immediate to verify that, for $k' = k$:

$$\rho_{EL}(h,k) \geq \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} = \frac{k}{k-h+1} \tag{19}$$

And since, if $k' \leq h$, we have that:

$$\frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \leq \frac{k'}{k' - (h\frac{k'}{k} - \frac{h}{k})} = \frac{k'\frac{k}{k'}}{k'\frac{k}{k'} - h\frac{k'}{k}\frac{k}{k'} + \frac{h}{k}\frac{k}{k'}} \leq \frac{k}{k-h+1} \tag{20}$$

then values of $k' \leq h$ can be disregarded in the max operation. To prove that, for all $h \leq k$, $\rho_{EL}(h,k) \leq (1+\frac{1}{k})\frac{k}{k-h+1}$, note that:

$$\begin{aligned} \rho_{EL}(h,k) &= max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \\ &\leq max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - (h\frac{k'}{k} - \frac{h}{k})} = \frac{k}{k - (h\frac{k}{k} - \frac{h}{k})} = \frac{k}{k - h + \frac{h}{k}} \end{aligned} \tag{21}$$

Then we obtain:

$$\frac{\rho_{EL}(h,k)}{\frac{k}{k-h+1}} \leq \frac{k-h+1}{k-h+\frac{h}{k}} = 1 + \frac{\frac{k-h}{k}}{k-h+\frac{h}{k}} = 1 + \frac{1}{k + \frac{h}{k-h}} < 1 + \frac{1}{k} \tag{22}$$

To prove that $\rho_{EL}(h,k)$ coincides with $\frac{k}{k-h+1}$ for $h > k - \sqrt{k}$ let us rewrite h and k' as $h = k - a$ and $k' = k - b$, with $a > b$ and $a, b \in \mathbb{Z}_0^+$. We obtain:

$$\begin{aligned} \rho_{EL}(h > (k-\sqrt{k}), k) &= \max_{\sqrt{k} > a > b} \frac{k-b}{k - b - \lfloor \frac{(k-a)(k-b)}{k} - \frac{k-a}{k} \rfloor} \\ &= \max_{\sqrt{k} > a > b} \frac{k-b}{k - b - \lfloor k - (a+b) + \frac{ab}{k} - 1 + \frac{a}{k} \rfloor} \\ &= \max_{\sqrt{k} > a > b} \frac{k-b}{k - b - k + (a+b) + 1 - \lfloor \frac{a(b+1)}{k} \rfloor} \\ &< \max_{\sqrt{k} > a > b} \frac{k}{k - h + 1 - \lfloor \frac{a(b+1)}{k} \rfloor} = \frac{k}{k-h+1} \end{aligned} \tag{23}$$

where the last equality follows from the fact that, since $a = k - h < \sqrt{k}$ and $b \leq a - 1 < \sqrt{k} - 1$, then $a(b+1) < k$. ◀

The complex expression of $\rho_{EL}(h,k)$ is in part due to the "rounding" of the memory capacity of the optimal offline algorithm. However, it is important to note that this rounding is not sufficient to explain why $\rho_{EL}(h,k)$ can be strictly *larger* than the "classic" ratio $\frac{k}{k-h+1}$ obtained when capacity is fixed at its maximum value: at smaller capacities rounding can only favour the online algorithm, and for any fixed ratio $\frac{k'}{h'}$, $\frac{k'}{k'-h'+1}$ strictly decreases with k', again favouring the online algorithm at smaller capacities. *Capacity fluctuations (rather than simply the choice between different, constant capacities) are then the source of the separation* between $\rho_{EL}(h,k)$ and the "classic" (h,k)-competitive ratio $\frac{k}{k-h+1}$.

6 Decoupling replacement from capacity in RAM rental

The results from Section 4 can be readily applied to the RAM rental problem, in which a paging algorithm ALG can choose the capacity sequence (with maximum capacity k), and the cost it incurs and must minimize on a request sequence σ is:

$$R^k_{ALG}(\sigma) = \sum_{i=1}^{|\sigma|} (\alpha f(i) + \beta w(i)) \tag{24}$$

where $w(i)$ is the capacity when serving the i^{th} request of σ, and $f(i)$ is 1 if that request is a fault and 0 otherwise. The fundamental consequence of our results from Section 4 is that to a large extent the replacement policy can be decoupled from the choice of capacities. More precisely, Theorem 9 yields:

▶ **Corollary 14.** *Consider a paging algorithm ALG, servicing each request σ_i of a sequence σ with capacity $w(i) \leq h$ and an arbitrary (even offline) replacement policy; and a second paging algorithm ALG′ servicing σ_i with capacity $2w(i)$ and a replacement policy that can be any marking or dynamically conservative algorithm. Then, for any choice of α, β and $w(\cdot) \leq h$:*

$$R^{2h}_{ALG'}(\sigma) \leq 2 \cdot R^h_{ALG}(\sigma). \tag{25}$$

which follows immediately from the fact that the sum of all faults incurred by ALG' is at most twice that by ALG as long as ALG' maintains twice the capacity of ALG. *In other words, RAM rental is all about choosing the correct capacity at any given time; and any of the "classic" replacement policies analysed in the previous section will be close to optimal for any choice of α, of β, and of the capacity sequence.*

7 Conclusions

Good performance in the case of constant memory capacity provides no performance guarantees whatsoever in the case of fluctuating memory capacity: moving from a scenario where capacity remains constant to one where it can fluctuate by a single page can mean the difference between performance optimal within a factor 2, and performance suboptimal by an arbitrarily large factor. This suggests the need of extreme caution when evaluating with classic methodologies the performance of paging algorithms meant for memory systems with dynamic capacity.

A counterpoint to this very "negative" result is that several extremely simple classic paging algorithms achieve optimal or nearly optimal performance even in the dynamic capacity framework - which is surprising because none of those algorithms is designed to take memory capacity fluctuations into explicit account. Counterintuitively then, while knowledge of future page requests provides an advantage, knowledge of future memory capacity does not. A practical corollary is that, in the design of memory architectures, one can efficiently decouple the problem of allocating memory resources to different cores/processes/threads from the problem of managing the allocated memory – greatly simplifying system design and analysis and providing a strong (a posteriori!) theoretical justification for the exokernel approach [12].

As in classic paging, in the dynamic capacity framework competitive analysis fails to distinguish between the performance of LRU, of FIFO, and of more naive algorithms such as RAND or FWF – at least without resorting to more sophisticated approaches such as

access graphs. While each of these algorithms is still guaranteed to outperform an optimal offline algorithm (and thus any other online algorithm) whose memory system has half the capacity and twice the access cost, there are probably differences within those factors of 2 that would be important to characterize in practice. It is by no means clear whether the winner in the dynamic capacity scenario would be the same as in the classic one, or whether models designed a posteriori to explain the superiority of e.g. LRU over FIFO would still provide correct predictions.

In this sense we are not aware of any experimental benchmarks specifically designed to assess the impact of memory capacity fluctuations. A fundamental obstacle in their development seems to be the difficulty of characterizing "typical" fluctuation patterns encountered in practice. An interesting line of inquiry would be to investigate whether one can obtain, from the performance numbers of a black box algorithm under a small "basis" of specific fluctuation patterns, sufficient information to compute a good assessment of the algorithm's performance numbers under any other pattern.

Finally, the impact of resource fluctuation on how efficiently a task can be solved is a line of inquiry obviously not limited to memory management. There are a number of other problems where the amount of resources available can realistically vary over time. Examples include call admission [16] (with variable circuit capacity) and the numerous variants of online scheduling [27] (with e.g. variable number or speed of servers). In addition to studying each problem individually, it would be extremely interesting to identify broad classes sharing similar characteristics. For example, which problems can be solved optimally or almost optimally without knowledge of the amount of resources available in the future (as in the case of paging with dynamic memory capacity)?

References

1 Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. Syst. Sci.*, 70(2):145–175, 2005. doi:10.1016/j.jcss.2004.08.002.

2 Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 229–237, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283408.

3 Rakesh D. Barve and Jeffrey Scott Vitter. A Theoretical Framework for Memory-Adaptive Algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 273–284, 1999. doi:10.1109/SFFCS.1999.814599.

4 Laszlo A. Belady. A Study of Replacement Algorithms for Virtual-Storage Computer. *IBM Systems Journal*, 5(2):78–101, 1966. doi:10.1147/sj.52.0078.

5 Shai Ben-David and Allan Borodin. A New Measure for the Study of On-Line Algorithms. *Algorithmica*, 11(1):73–91, 1994. doi:10.1007/BF01294264.

6 Michael A. Bender, Erik D. Demaine, Roozbeh Ebrahimi, Jeremy T. Fineman, Rob Johnson, Andrea Lincoln, Jayson Lynch, and Samuel McCauley. Cache-Adaptive Analysis. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 135–144, 2016. doi:10.1145/2935764.2935798.

7 Michael A. Bender, Roozbeh Ebrahimi, Jeremy T. Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. Cache-Adaptive Algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 958–971, 2014. doi:10.1137/1.9781611973402.71.

8 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

9 Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive Paging with Locality of Reference. *J. Comput. Syst. Sci.*, 50(2):244–258, 1995. doi:10.1006/jcss.1995.1021.

10 Joan Boyar, Sandy Irani, and Kim S. Larsen. A Comparison of Performance Measures for Online Algorithms. *Algorithmica*, 72(4):969–994, 2015. doi:10.1007/s00453-014-9884-6.

11 Marek Chrobak. SIGACT news online algorithms column 17. *SIGACT News*, 41(4):114–121, 2010. doi:10.1145/1907450.1907547.

12 Dawson R. Engler, M. Frans Kaashoek, and James O'Toole. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, SOSP 1995, Copper Mountain Resort, Colorado, USA, December 3-6, 1995*, pages 251–266, 1995. doi:10.1145/224056.224076.

13 Amos Fiat and Anna R. Karlin. Randomized and multipointer paging with locality of reference. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 626–634, 1995. doi:10.1145/225058.225280.

14 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive Paging Algorithms. *J. Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.

15 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-Oblivious Algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012.

16 Juan A. Garay, Inder S. Gopal, Shay Kutten, Yishay Mansour, and Moti Yung. Efficient On-Line Call Control Algorithms. *J. Algorithms*, 23(1):180–194, 1997.

17 Avinatan Hassidim. Cache Replacement Policies for Multicore Processors. In *ICS*, pages 501–509. Tsinghua University Press, 2010.

18 Houman Homayoun, Mohammad A. Makhzan, and Alexander V. Veidenbaum. Multiple sleep mode leakage control for cache peripheral circuits in embedded processors. In *Proceedings of the 2008 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2008, Atlanta, GA, USA, October 19-24, 2008*, pages 197–206, 2008. doi:10.1145/1450095.1450125.

19 Sandy Irani. Page Replacement with Multi-Size Pages and Applications to Web Caching. *Algorithmica*, 33(3):384–409, 2002.

20 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive Snoopy Caching. *Algorithmica*, 3:77–119, 1988. doi:10.1007/BF01762111.

21 Elias Koutsoupias and Christos H. Papadimitriou. Beyond Competitive Analysis. *SIAM J. Comput.*, 30(1):300–317, 2000.

22 Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Computers*, 50(12):1352–1361, 2001.

23 Alejandro López-Ortiz and Alejandro Salinger. Minimizing Cache Usage in Paging. In *WAOA*, volume 7846 of *Lecture Notes in Computer Science*, pages 145–158. Springer, 2012.

24 Nimrod Megiddo and Dharmendra S. Modha. Outperforming LRU with an Adaptive Replacement Cache Algorithm. *IEEE Computer*, 37(4):58–65, 2004. doi:10.1109/MC.2004.1297303.

25 Enoch Peserico. Elastic Paging. *SIGMETRICS Perform. Eval. Rev.*, 41(1):349–350, June 2013. doi:10.1145/2494232.2479781.

26 Enoch Peserico. Paging with dynamic memory capacity. *CoRR*, abs/1304.6007, 2013. arXiv:1304.6007v1.

27 Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007. doi:10.1145/1243401.1243411.

28 Moinuddin K. Qureshi and Yale N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *MICRO*, pages 423–432. IEEE Computer Society, 2006.

29 Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–708, 1994.

30 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

31 Eric Torng. A Unified Analysis of Paging and Caching. *Algorithmica*, 20:194–203, 1998.

32 Neal E. Young. On-Line Caching as Cache Size Varies. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California, USA.*, pages 241–250, 1991. URL: http://dl.acm.org/citation.cfm?id=127787.127832.

33 Neal E. Young. The K-Server Dual and Loose Competitiveness for Paging. *Algorithmica*, 11:525–541, 1994.

Random Noise Increases Kolmogorov Complexity and Hausdorff Dimension

Gleb Posobin
Computer Science department, Columbia University, New York, USA
posobin@gmail.com

Alexander Shen
LIRMM CNRS & University of Montpellier, 161 rue Ada, 34095, Montpellier, France,
On leave from IITP RAS, Moscow
https://www.lirmm.fr/~ashen
alexander.shen@lirmm.fr

Abstract

Consider a bit string x of length n and Kolmogorov complexity αn (for some $\alpha < 1$). It is always possible to increase the complexity of x by changing a small fraction of bits in x [2]. What happens with the complexity of x when we *randomly* change each bit independently with some probability τ? We prove that a linear increase in complexity happens with high probability, but this increase is smaller than in the case of arbitrary change considered in [2]. The amount of the increase depends on x (strings of the same complexity could behave differently). We give exact lower and upper bounds for this increase (with $o(n)$ precision).

The same technique is used to prove the results about the (effective Hausdorff) dimension of infinite sequences. We show that random change increases the dimension with probability 1, and provide an optimal lower bound for the dimension of the changed sequence. We also improve a result from [5] and show that for every sequence ω of dimension α there exists a strongly α-random sequence ω' such that the Besicovitch distance between ω and ω' is 0.

The proofs use the combinatorial and probabilistic reformulations of complexity statements and the technique that goes back to Ahlswede, Gács and Körner [1].

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

Keywords and phrases Kolmogorov complexity, effective Hausdorff dimension, random noise

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.57

Related Version An extended version of the paper is available as https://arxiv.org/abs/1808.04626.

Funding Supported by RaCAF ANR-15-CE40-0016-01 grant.
Gleb Posobin: The work was done when G. Posobin was at the Laboratory of Theoretical Computer Science, National Research University Higher School of Economics, Moscow, Russia, and was supported by Russian Academic Excellence Project 5–100 and MK-5379.2018.1 grant. The preparation of the final version was supported by NSF CAREER Award CCF-1844887 and CCF-1563155 grants.
Alexander Shen: Supported in part by RFBR 19-01-00563A grant. Part of the work done while visiting Toyota Technological University, Chicago.

Acknowledgements Authors are grateful to the participants and organizers of the Heidelberg Kolmogorov complexity program where the question of the complexity increase was raised, and to all colleagues (from the ESCAPE team, LIRMM, Montpellier, Kolmogorov seminar and HSE Theoretical Computer Science Group, and other places) who participated in the discussions, in particular to B. Bauwens, N. Greenberg, K. Makarychev, Yu. Makarychev, J. Miller, A. Milovanov, F. Nazarov, I. Razenshteyn, A. Romashchenko, N. Vereshchagin, L. B. Westrick, and, last but not least, to Peter Gács who explained us how the tools from [1] can be used to provide the desired result about Kolmogorov complexity.

1 Introduction

The Kolmogorov complexity $C(x)$ of a binary string x is defined as the minimal length of a program that generates x, assuming that we use an optimal programming language that makes the complexity function minimal up to an $O(1)$ additive term (see [8, 12] for details). There are several versions of Kolmogorov complexity; we consider the original version, called *plain* complexity. In fact, for our considerations the difference between different versions of Kolmogorov complexity does not matter, since they differ only by $O(\log n)$ additive term for n-bit strings, but we restrict ourselves to plain complexity for simplicity.

The complexity of n-bit strings is between 0 and n (we omit $O(1)$ additive terms). Consider a string x of length n that has some intermediate complexity, say $0.5n$. Imagine that we are allowed to change a small fraction of bits in x, say, 1% of all bits. Can we decrease the complexity of x? Can we increase the complexity of x? What happens if we change randomly chosen 1% of bits?

In other words, consider a Hamming ball with center x and radius $0.01n$, i.e., the set of strings that differ from x in at most $0.01n$ positions. What can be said about the minimal complexity of strings in this ball? the maximal complexity of strings in this ball? the typical complexity of strings in this ball?

The answer may depend on x: different strings of the same complexity may behave differently if we are interested in the complexities of neighbor strings. For example, if the first half of x is a random string, and the second half contains only zeros, the string x has complexity $0.5n$ and it is easy to decrease its complexity by shifting the boundary between the random part and zero part: to move the boundary to $0.48n$ from $0.5n$ we need to change about $0.01n$ bits, and the complexity becomes close to $0.48n$. On the other hand, if x is a random codeword of an error-correcting code with $2^{0.5n}$ codewords of length n that corrects up to $0.01n$ errors, then x also has complexity $0.5n$, but no change of $0.01n$ (or less) bits can decrease the complexity of x, since x can be reconstructed from the changed version.

The question about the complexity *decrease* is studied by algorithmic statistics (see [14] or the survey [13]), and the full answer is known. For each x one may consider the function

$$d \mapsto (\text{the minimal complexity of strings in the } d\text{-ball centered at } x).$$

It starts at $C(x)$ (when $d = 0$) and then decreases, reaching 0 at $d = n/2$ (since we can change all bits to zeros or to ones). The algorithmic statistic tells us which functions may appear in this way (see [13, section 6.2] or [12, theorem 257]).[1]

The question about the complexity *increase* is less studied. It is known that some complexity increase is always guaranteed, as shown in [2]. The amount of this increase may depend on x. If x is a random codeword of an error-correcting code, then the changed version of x contains all the information both about x itself and the places where it was changed. This leads to the maximal increase in complexity. The minimal increase, as shown in [2], happens for x that is a random element of the Hamming ball of some radius with center 0^n. However, the natural question: which functions may appear as $d \mapsto$ (the maximal complexity of strings in the d-ball centered at x), remains open.

In our paper we study the *typical* complexity of a string that can be obtained from x by changing a fraction of bits chosen randomly. Let us return to our example and consider again a string x of length n and complexity $0.5n$. Let us change about 1% of bits in x, changing

[1] Note that algorithmic statistics uses a different language. Instead of a string y in the d-ball centered at x, it speaks about a d-ball centered at y and containing x. This ball is considered as a statistical model for x.

each bit independently[2] with probability 0.01. Does this change increase the complexity of x? It depends on the changed bits, but it turns out that *random change increases the complexity of the string with high probability*: we get a string of complexity at least $0.501n$ with probability at least 99%, for all large enough n (the result is necessarily asymptotic, since the Kolmogorov complexity function is defined up to $O(1)$ terms).

Of course, the parameters above are chosen only as an example, and the following general statement is true. For some $\tau \in (0,1)$ consider the random noise N_τ that changes each position in a given n-bit string independently with probability τ.

▶ **Theorem 1.** *There exists a strictly positive function $\delta(\alpha,\tau)$ defined for $\alpha,\tau \in (0,1)$ with the following property: for all sufficiently large n, for every $\alpha \in (0,1)$, for every $\tau \in (0,1)$, for $\beta = \alpha + \delta(\alpha,\tau)$, and for every x such that $\mathrm{C}(x) \geqslant \alpha n$, the probability of the event "$\mathrm{C}(N_\tau(x)) > \beta n$" is at least $1 - 1/n$.*

▶ Remark 2. We use the inequality $\mathrm{C}(x) \geqslant \alpha n$ (and not an equality $\mathrm{C}(x) = \alpha n$) to avoid technical problems: the complexity $\mathrm{C}(x)$ is an integer, and αn may not be an integer.

▶ Remark 3. One may consider only $\tau \leqslant 1/2$ since reversing all bits does not change Kolmogorov complexity (so τ and $1-\tau$ give the same increase in complexity). For $\tau = 1/2$ the variable $N_\tau(x)$ is uniformly distributed in the Boolean cube $\mathbb{B}^n$, so its complexity is close to n, and the statement is easy (for arbitrary $\beta < 1$).

▶ Remark 4. We use α,τ as parameters while fixing the probability bound as $1 - 1/n$. As we will see, the choice of this bound is not important: we could use a stronger bound (e.g., $1 - 1/n^d$ for arbitrary d) as well.

Now a natural question arises: what is the optimal bound in Theorem 1, i.e., the maximal possible value of $\delta(\alpha,\tau)$? In other words, fix α and τ. Theorem 1 guarantees that there exists some $\beta > \alpha$ such that every string x of length n (sufficiently large) and complexity at least αn is guaranteed to have complexity at least βn after τ-noise N_τ (with high probability). *What is the maximal value of β for which such a statement is true* (for given α and τ)?

Before answering this question, we should note that the guaranteed complexity increase depends on x: for different strings of the same complexity the typical complexity of $N_\tau(x)$ could be different. Here are the two opposite examples (with minimal and maximal increase, as we will see).

▶ **Example 5.** Consider some $p \in (0,1)$ and the Bernoulli distribution B_p on the Boolean cube $\mathbb{B}^n$ (bits are independent; every bit equals 1 with probability p). With high probability the complexity of a B_p-random string is $o(n)$-close to $n\,\mathrm{H}(p)$ (see, e.g., [12, chapter 7]), where $\mathrm{H}(p)$ is the Shannon entropy function $\mathrm{H}(p) = -p\log p - (1-p)\log(1-p)$. After applying τ-noise the distribution B_p is transformed into $B_{N(\tau,p)}$, where $N(\tau,p) = p(1-\tau) + (1-p)\tau = p + \tau - 2p\tau$ is the probability to change the bit if we first change it with probability p and then (independently) with probability τ.[3] The complexity of $N_\tau(x)$ is close (with high probability) to $H(N(\tau,p))n$ since the B_p-random string x and the τ-noise are chosen independently. So in this case we have (with high probability) the complexity increase $\mathrm{H}(p)n \to \mathrm{H}(N(\tau,p))n$. Note that $N(\tau,p)$ is closer to $1/2$ than p, and H is strictly increasing on $[0,1/2]$, so indeed some increase happens.

[2] From the probabilistic viewpoint it is more naturally to change all the bits independently with the same probability 0.01. Then the number of changed bits is not exactly $0.01n$, but is close to $0.01n$ with high probability.

[3] We use the letter N (for "noise") both in $N_\tau(x)$ (random change with probability τ, one argument) and in $N(\tau,p)$ (the parameter of the Bernoulli distribution B_p after applying N_τ, no subscript, two arguments).

▶ **Example 6.** Now consider an error-correcting code that has $2^{\alpha n}$ codewords and corrects up to τn errors (this means that the Hamming distance between codewords is greater than $2\tau n$). Such a code may exist or not depending on the choice of α and τ. The basic result in coding theory, Gilbert's bound, guarantees that such a code exists if α and τ are not too large. Consider some pair of α and τ for which such a code exist; moreover, let us assume that it corrects up to $\tau' n$ errors for some $\tau' > \tau$. We assume also that the code itself (the list of codewords) has small complexity, say, $O(\log n)$. This can be achieved by choosing the first (in some ordering) code with required parameters.

Now take a random codeword of this code; most of the codewords have complexity close to αn. If we randomly change each bit with probability τ, then with high probability we get at most $\tau' n$ errors, therefore, decoding is possible and the pair (x, noise) can be reconstructed from $N_\tau(x)$, the noisy version of x. Then the complexity of $N_\tau(x)$ is close to the complexity of the pair (x, noise), which (due to independence) is close to $\alpha n + \mathrm{H}(\tau)n$ with high probability. So in this case we have the complexity increase $\alpha n \to (\alpha + \mathrm{H}(\tau))n$.

▶ Remark 7. Note that this increase is the maximal possible not only for random independent noise but for any change in x that changes a τ-fraction of bits. See below about the difference between random change and arbitrary change.

Now we formulate the result we promised. It says that the complexity increase observed in Example 5 is the minimal possible: such an increase is guaranteed for every string of given complexity.

▶ **Theorem 8.** *Let $\alpha = \mathrm{H}(p)$ for some $p \leqslant 1/2$. Let τ be an arbitrary number in $(0, 1)$. Let $\beta = \mathrm{H}(N(p, \tau))$. Then for sufficiently large n the following is true: for every string x of length n with $\mathrm{C}(x) \geqslant \alpha n$, we have $\Pr[\mathrm{C}(N_\tau(x)) \geqslant \beta n - o(n)] \geqslant 1 - \frac{1}{n}$.*

Here $o(n)$ denotes some function such that $o(n)/n \to 0$ as $n \to \infty$. This function does not depend on α, β, and τ. As the proof will show, we may take $o(n) = c\sqrt{n}\log^{3/2} n$ for some c.

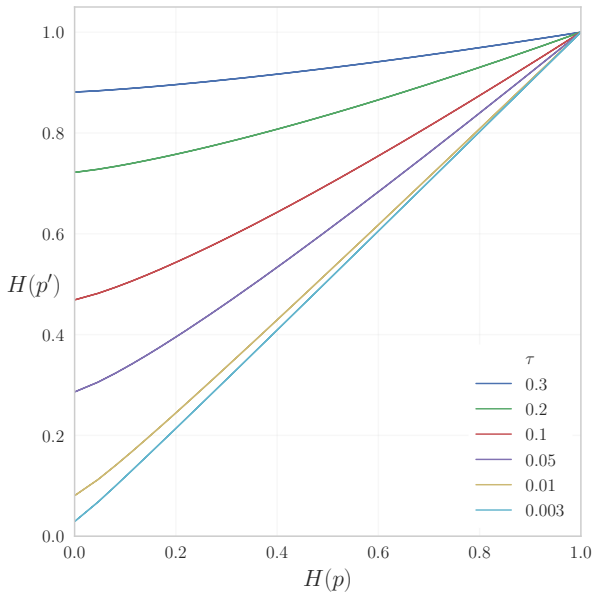


Figure 1 The curves $(\mathrm{H}(p), \mathrm{H}(p'))$ where $p' = N(p, \tau)$. Six different values of τ are shown.

Figure 1 shows the values of (α, β) where Theorem 8 can be applied, for six different values of τ. Example 5 shows that the value of β in this theorem is optimal.

In the next section we explain the scheme of the proof of Theorem 8. Then we explain the details of the proof. It starts with the Shannon information counterpart of our complexity statement that is proven in [15]. In Section 3 we derive two different combinatorial counterparts following [1]. Finally, in Section 4 we consider the details of the conversion of a combinatorial statement to a complexity one and finish the proof. In Section 5 we extend our techniques to infinite sequences and improve some results obtained in [5].

2 Proof sketch

2.1 Three ways to measure the amount of information

Kolmogorov's first paper on algorithmic information theory [7] was called "Three approaches to the Quantitative Definition of Information". These three approaches can be summarized as follows:

- (Combinatorial): an element of a set of cardinality N carries $\log N$ bits of information.
- (Algorithmic): a binary string x carries $\mathrm{C}(x)$ bits of information, where $\mathrm{C}(x)$ is the minimal bit length of a program that produces x.
- (Shannon information theory, or probabilistic approach): a random variable ξ that has k values with probabilities $p_1, \ldots, p_k$, carries $\mathrm{H}(\xi)$ bits of information, where $\mathrm{H}(\xi)$ is the Shannon entropy of ξ, defined as $\mathrm{H}(\xi) = p_1 \log \frac{1}{p_1} + \ldots + p_k \log \frac{1}{p_k}$

One cannot compare directly these three quantities since the measured objects are different (sets, strings, random variables). Still these quantities are closely related, and many statements that are true for one of these notions can be reformulated for other two. Several examples of this type are discussed in [12, chapters 7 and 10], and we use this technique in our proof.

2.2 Arbitrary change

We start by recalling an argument from [2] for the case when we are allowed to change arbitrary bits (only the number of changed bits is bounded) and want to increase complexity. (A similar reduction will be a part of our argument.)

Fix some parameters α (determining the complexity of the original string), τ (the maximal fraction of changed bits), and β (determining the complexity of the changed string). Let us repeat the complexity statement and give its combinatorial equivalent.

- (Complexity version) Every string x of length n and complexity at least αn can be changed in at most τn positions to obtain a string of complexity at least βn.
- (Combinatorial version) For every subset B of the Boolean cube $\mathbb{B}^n$ of cardinality at most $2^{\beta n}$, its τn-interior has cardinality at most $2^{\alpha n}$.

Here by d-interior of a set $X \subset \mathbb{B}^n$ we mean the set of strings $x \in \mathbb{B}^n$ such that the entire ball of radius d centered at x belongs to X. In other words, a string x does *not* belong to the d-interior of X if x can be changed in at most d positions to get a string outside X.

▶ Remark 9. The combinatorial statement can be reformulated in a dual way: for every set $A \subset \mathbb{B}^n$ of cardinality greater than $2^{\alpha n}$, its d-neighborhood has cardinality greater than $2^{\beta n}$.

These two statements (combinatorial and complexity versions) are almost equivalent: one of them implies the other if we allow a small change in α and β (in fact, $O(\log n)/n$ change is enough). Indeed, assume first that the combinatorial statement is true. Consider the set B of all n-bit strings that have complexity less than βn. Then $\#B < 2^{\beta n}$, so we can

apply the combinatorial statement.[4] It guarantees that the τn-interior of B (we denote it by A) has at most $2^{\alpha n}$ elements. The set A can be enumerated given n, βn and τn. Indeed, knowing n and βn, one can enumerate elements of B (by running in parallel all programs of length less than βn; note that there are less than $2^{\beta n}$ of them). Knowing also τn, we may enumerate A (if a ball is contained in B entirely, this will become known at some stage of the enumeration of B). Then every element of A can be encoded by its ordinal number in this enumeration. This guarantees that the complexity of all elements of A does not exceed $\alpha n + O(\log n)$ (the additional $O(\log n)$ bits are needed to encode n, βn, and τn). Therefore, if some x has complexity greater that $\alpha n + O(\log n)$, it is not in A, i.e., x can be changed in at most τn positions to get a string outside B. By the definition of B, this changed string has complexity at least βn, as required. The term $O(\log n)$ can be absorbed by a small change in α.

Let us explain also (though this direction is not needed for our purpose) why the complexity statement implies the combinatorial one. Assume that there are some sets B that violate the combinatorial statement, i.e., contain at most $2^{\beta n}$ strings but have τn-interior of size greater than $2^{\alpha n}$. Such a set can be found by exhaustive search, and the first set B that appears during the search has complexity $O(\log n)$. Its elements, therefore, have complexity $\beta n + O(\log n)$: to specify an element, we need to specify B and the ordinal number of the element in B. From this we conclude, using the complexity statement (and changing β slightly) that all elements of the τn-interior of B have complexity at most αn. Therefore, there are at most $O(2^{\alpha n})$ of them, and the size of the interior is bounded by $2^{\alpha n}$ (again up to a small change in α).

Now we return to the result from [2]. Let x be a string of length n and complexity at least $\alpha n + O(\log n)$, where $\alpha = \mathrm{H}(p)$ for some $p \leqslant 1/2$. Let τ be a real such that $p + \tau \leqslant 1/2$, and $\beta = \mathrm{H}(p+\tau)$. Then x can be changed in at most τn positions to get a string of complexity at least βn. As we have seen, this statement from [2] is a corollary of the following combinatorial result.

▶ **Proposition 10.** *Let $p \leqslant 1/2$ be some number and let $\alpha = \mathrm{H}(p)$. Let τ be some positive number so that $p + \tau \leqslant 1/2$, and let $\beta = \mathrm{H}(p + \tau)$. Let B be an arbitrary subset of $\mathbb{B}^n$ of size at most $2^{\beta n}$. Let A be a subset of $\mathbb{B}^n$, and for every $x \in A$ the Hamming ball of radius τn with center x is contained in B. Then the cardinality of A does not exceed* $\mathrm{poly}(n)2^{\alpha n}$.

This proposition is a direct consequence of Harper's theorem (see, e.g., [4]) that says that for a subset of $\mathbb{B}^n$ of a given size, its *d-interior* (for some fixed d) *is maximal when the subset is a Hamming ball* (formally speaking, is between two Hamming balls of sizes k and $k+1$ for some k). Or, in dual terms, Harper's theorem says that the *d-neighborhood of a set of a given size is minimal if this set is a Hamming ball.* The relation between $2^{\alpha n}$ and $2^{\beta n}$ in the proposition is just the relation between the sizes of balls of radii pn and $(p+\tau)n$ (if we ignore factors that are polynomial in n). Note that $p + \tau \leqslant 1/2$ is needed since otherwise the radius exceeds $n/2$ and then the log-size of the ball is close to n and not to $\mathrm{H}(p+\tau)n$. The $\mathrm{poly}(n)$ factor is needed due to the polynomial factor in the estimate for the ball size in terms of Shannon entropy (the ball of radius γn has size $\mathrm{poly}(n)2^{\mathrm{H}(\gamma)n}$).

We do not go into details here (and do not reproduce the proof of Harper's theorem) since we need this result only to motivate the corresponding relation between combinatorial and complexity statements for the case of a random noise we are interested in.

[4] For simplicity we assume that αn, βn, and τn are integers. This is not important, since we have $O(\log n)$ term anyway.

2.3 Random noise: four versions

For the random noise case we need a more complicated argument. First, we need to consider also the probabilistic version of the statement (in addition to the complexity and combinatorial versions). Second, we need *two* combinatorial versions (strong and weak). Fix some α, β and τ. Here are the four versions we are speaking about; all four statements are equivalent (are true for the same parameters α, τ, and β, up to $o(1)$-changes in the parameters):

- (Shannon information version, [15]) For every random variable P with values in $\mathbb{B}^n$ such that $\mathrm{H}(P) \geqslant \alpha n$, the variable $N_\tau(P)$ that is obtained from P by applying independent noise changing each bit with probability τ, has entropy $\mathrm{H}(N_\tau(P)) \geqslant \beta n$.
- (Complexity version) For every string x of length n and complexity $\mathrm{C}(x) \geqslant \alpha n$, the probability of the event "$\mathrm{C}(N_\tau(x)) \geqslant \beta n$" is at least $1 - 1/n$. (Again, N_τ is random noise that independently changes each bit with probability τ, but now it is applied to the string x and not to a random variable)
- (Strong combinatorial version) For every set $B \subset \mathbb{B}^n$ of size at most $2^{\beta n}$ the set A of all strings x such that $\Pr[N_\tau(x) \in B] \geqslant 1/n$ has size $\#A \leqslant 2^{\alpha n}$.
- (Weak combinatorial version) For every set $B \subset \mathbb{B}^n$ of size at most $2^{\beta n}$ the set A of all strings x such that $\Pr[N_\tau(x) \in B] \geqslant 1 - 1/n$ has size $\#A \leqslant 2^{\alpha n}$.

The difference between weak and strong combinatorial versions is due to the different thresholds for the probability. In the weak version the set A contains only strings that get into B after the noise *almost surely* (with probability at least $1 - 1/n$). In the strong version the set A is bigger and includes all strings that get into B *with non-negligible probability* (at least $1/n$), so the upper bound for $\#A$ becomes a stronger statement.

▶ Remark 11. In the case of arbitrary changes (the result from [2]) we consider the τn-interior of B, the set of points that remain in B after arbitrary change in (at most) τn positions. If a point is *not* in the interior, it can be moved outside B by changing at most τn bits. Now we consider (in the strong version) the set of points that get into B with probability at least $1/n$. If a point is *not* in this set, the random τ-noise will move it outside B almost surely (with probability at least $1 - 1/n$). Again the complexity and (strong) combinatorial versions are equivalent up to $o(1)$ changes in parameters, for the same reasons.

This explains why we are interested in the strong combinatorial statement. The weak one is used as an intermediate step in the chain of arguments. This chain goes as follows:

- First the Shannon entropy statement is proven using tools from information theory (one-letter characterization and inequalities for Shannon entropy); this was done in [15].
- Then we derive the weak combinatorial statement from the entropy statement using a simple coding argument from [1].
- Then we show that weak combinatorial statement implies the strong one, using a tool that is called the "blowing-up lemma" in [1] (now it is more popular under the name of "concentration inequalities").
- Finally, we note that the strong combinatorial statement implies the complexity statement (using the argument sketched above).

2.4 Tools used in the proof

Let us give a brief description of the tools used in these arguments.

To prove the Shannon entropy statement, following [15], fix some τ. Consider the set S_n of all pairs $(\mathrm{H}(P), \mathrm{H}(N_\tau(P)))$ for all random variables with values in $\mathbb{B}^n$. For each n we get a subset of the square $[0, n] \times [0, n]$. For $n = 1$ it is a curve made of all points

$(\mathrm{H}(p), \mathrm{H}(N(p,\tau)))$ (shown in Figure 1 for six different values of τ). We start by showing that this curve is convex (performing some computation with power series). Then we show, using the convexity of the curve and some inequalities for entropies, that for every n the set S_n is above the same curve (scaled by factor n), and this is the entropy statement we need. (See the arxiv version of this paper for details.)

To derive the weak combinatorial statement from the entropy statement, we use a coding argument. Assume that two sets A and B are given, and for every point $x \in A$ the random point $N_\tau(x)$ belongs to B with probability at least $1 - 1/n$. Consider a random variable U_A that is uniformly distributed in A. Then $\mathrm{H}(U_A) = \log \#A$, and if $\#A \geqslant 2^{\alpha n}$, then $\mathrm{H}(U_A) \geqslant \alpha n$ and $\mathrm{H}(N_\tau(U_A)) \geqslant \beta n$ (assuming the entropy statement is true for given α, β, and τ). On the other hand, the variable $N_\tau(U_A)$ can be encoded as follows:

- one bit (flag) says whether $N_\tau(U_A)$ is in B or not;
- if yes, then $\log \#B$ bits are used to encode an element of B;
- otherwise n bits are used to encode the value of $N_\tau(U_A)$ (trivial encoding).

The average length of this code for $N_\tau(U_A)$) does not exceed

$$1 + \left(1 - \frac{1}{n}\right) \log \#B + \frac{1}{n} \cdot n \leqslant \log \#B + O(1).$$

(Note that if the second case has probability less than $1/n$, the average length is even smaller.) The entropy of a random variable $N_\tau(U_A)$ does not exceed the average length of the code. So we get $\beta n \leqslant \mathrm{H}(N_\tau(U_A)) \leqslant \log \#B + O(1)$ and $\log \#B \geqslant \beta n - O(1)$, assuming that $\log \#A \geqslant \alpha n$.

The next step is to derive the strong combinatorial version from the weak one. Assume that two sets $A, B \subset \mathbb{B}^n$ are given, and for each $x \in A$ the probability of the event $N_\tau(x) \in B$ is at least $1/n$. For some d consider the set B_d, the d-neighborhood of B. We will prove (using the concentration inequalities) that for some $d = o(n)$ the probability of the event $N_\tau(x) \in B_d$ is at least $1 - 1/n$ (for each $x \in A$). So one can apply the weak combinatorial statement to B_d and get a lower bound for $\#B_d$. On the other hand, there is a simple upper bound: $\#B_d \leqslant \#B \times$ (the size of d-ball); combining them, we get the required bound for $\#B$. See Section 3 for details.

▶ Remark 12. One may also note (though it is not needed for our purposes) that the entropy statement is an easy corollary of the complexity statement, and therefore all four are equivalent up to small changes in parameters. This can be proven in a standard way. Consider N independent copies of random variable P and independently apply noise to all of them. Then we write the inequality for the typical values of the complexities; in most cases they are close to the corresponding entropies (up to $o(N)$ error). Therefore, we get the inequality for entropies with $o(N)$ precision (for N copies) and with $o(1)$ precision for one copy (the entropies are divided by N). As $N \to \infty$, the additional term $o(1)$ disappears and we get an exact inequality for entropies.

3 Combinatorial version

Recall the entropy bound from [15] discussed above (we reproduce its proof in the arxiv version for reader's convenience):

▶ **Proposition 13.** *Let P be an arbitrary random variable with values in $\mathbb{B}^n$, and let $P' = N_\tau(P)$ be its noisy version obtained by applying N_τ independently to each bit in P. Choose $p \leqslant 1/2$ in such a way that $\mathrm{H}(P) = n\,\mathrm{H}(p)$. Then consider $q = N(p,\tau)$, the probability to get 1 if we apply N_τ to a variable that equals 1 with probability p. Then $\mathrm{H}(P') \geqslant n\,\mathrm{H}(q)$.*

In this section we use this entropy bound to prove the combinatorial bounds. We start with the weak one and then amplify it to get the strong one, as discussed in Section 2. First, let us formulate explicitly the weak bound that is derived from Proposition 13 using the argument of Section 2.

▶ **Proposition 14.** *Let* $\alpha = \mathrm{H}(p)$ *and* $\beta = \mathrm{H}(N(p,\tau))$. *Let* $A, B \subset \mathbb{B}^n$ *and for every* $x \in A$ *the probability of the event "*$N_\tau(x) \in B$*" is at least* $1 - 1/n$. *If* $\log \#A \geqslant \alpha n$, *then* $\log \#B \geqslant \beta n - O(1)$.

In fact, this "$O(1)$" is just 2, but we do not want to be too specific here.

Now we need to extend the bound to the case when the probability of the event $N_\tau(x) \in B$ is at least $1/n$. We already discussed how this is done. Consider for some d (depending on n) the Hamming d-neighborhood B_d of B. We need to show that

$$\Pr[N_\tau(x) \in B] \geqslant \frac{1}{n} \Rightarrow \Pr[N_\tau(x) \in B_d] \geqslant 1 - \frac{1}{n}.$$

for every $x \in \mathbb{B}^n$ (for a suitable d). In fact, x does not matter here: we may assume that $x = 0\ldots0$ (flipping bits in x and B simultaneously). In other terms, we use the following property of the Bernoulli distribution with parameter τ: *if some set* B *has probability not too small according to this distribution, then its neighborhood* B_d *has probability close to* 1. We need this property for $d = o(n)$, see below about the exact value of d.

Such a statement is called a *blowing-up lemma* in [1]. There are several (and quite different) ways to prove statements of this type. The original proof in [1] used a result of Margulis from [9] that says that the (Bernoulli) measure of a boundary of an arbitrary set $U \subset \mathbb{B}^n$ is not too small compared to the measure of a boundary of a ball of the same size. Iterating this statement (a neighborhood is obtained by adding boundary layer several times), we get the lower bound for the measure of the neighborhood. Another proof was suggested by Marton [10]; it is based on the information-theoretical considerations that involve transportation cost inequalities for bounding measure concentration. In this paper we provide a simple proof that uses the McDiarmid inequality [11], a simple consequence of the Azuma–Hoeffding inequality [6]. This proof works for $d = O(\sqrt{n \log n})$.

Let us state the blowing-up lemma in a slightly more general version than we need. Let $X_1, \ldots, X_n$ be (finite) probability spaces. Consider the space $X = X_1 \times \ldots \times X_n$ with the product measure μ (so the coordinates are independent) and Hamming distance d (the number of coordinates that differ). In our case $X = \mathbb{B}^n$ and μ is the Bernoulli measure with parameter τ. The blowing-up lemma says, informally speaking, that if a set is not too small, then its neighborhood has small complement (the size is measured by μ). It can be reformulated in a more symmetric way: *if two sets are not too small, then the distance between them is rather small.* (Then this symmetric statement is applied to the original set and the complement of its neighborhood.) Here is the symmetric statement:

▶ **Proposition 15** (Blowing-up lemma, symmetric version)**.** *Let* B, B' *be two subsets of* $X = X_1 \times \ldots \times X_n$ *with the product measure* μ. *Then*

$$d(B, B') \leqslant \sqrt{(n/2)\ln(1/\mu(B))} + \sqrt{(n/2)\ln(1/\mu(B'))}.$$

To prove the blowing-up lemma, we use the McDiarmid concentration inequality:

▶ **Proposition 16** (McDiarmid's inequality, [11])**.** *Consider a function* $f\colon X_1 \times \ldots \times X_n \to \mathbb{R}$. *Assume that changing the* i*-th coordinate changes the value of* f *at most by some* c_i*:* $|f(x) - f(x')| \leqslant c_i$, *if* x *and* x' *coincide everywhere except for the* i*th coordinate. Then*

$$\Pr[f - \mathbb{E}\, f \geqslant z] \leqslant \exp\left(-\frac{2z^2}{\sum_{i=1}^n c_i^2}\right)$$

for arbitrary $z \geqslant 0$.

Here the probability and expectation are considered with respect to the product distribution μ (the same as in the blowing-up lemma, see above). This inequality shows that f cannot be much larger than its average on a big set. Applying this inequality to $-f$, we get the same bound for the points where the function is less than its average by z or more. (For the reader's convenience we reproduce the proof of the McDiarmid inequality in the arxiv version of this paper.)

Now let us show why it implies the blowing-up lemma (in the symmetric version).

Proof of the blowing-up lemma. Let $f(x) = d(x, B)$ be the distance between x and B, i.e., the minimal number of coordinates that one has to change in x to get into B. This function satisfies the bounded differences property with $c_i = 1$, so we can apply the McDiarmid inequality to it. Let m be the expectation of f. The function f equals zero for arguments in B and therefore is below its expectation at least by m (everywhere in B), so

$$\mu(B) \leqslant \exp\left(-\frac{2m^2}{n}\right), \quad \text{or} \quad m \leqslant \sqrt{(n/2)\ln(1/\mu(B))}$$

On the other hand, the function f is at least $d(B, B')$ for arguments in B', so it exceeds its expectation at least by $d(B, B') - m$ (everywhere in B'), therefore the McDiarmid inequality gives

$$d(B, B') - m \leqslant \sqrt{(n/2)\ln(1/\mu(B'))},$$

and it remains to combine the last two inequalities. ◀

Here is the special case of the blowing-up lemma we need:

▶ **Corollary 17.** *If μ is a distribution on $\mathbb{B}^n$ with independent coordinates, and $B \subset \mathbb{B}^n$ has measure $\mu(B) \geqslant 1/n$, then for $d = O(\sqrt{n \log n})$ we have $\mu(B_d) \geqslant 1 - 1/n$.*

Indeed, we may apply the blowing-up lemma to B and B', where B' is a complement of B_d. If both B and B' have measures at least $1/n$, we get a contradiction for $d \geqslant 2\sqrt{(n/2)\ln n}$ (the distance between B and the complement of its neighborhood B_d exceeds d).

▶ Remark 18. In the same way we get a similar result for probabilities $1/n^c$ and $1 - 1/n^c$ for arbitrary constant c (only the constant factor in $O(\sqrt{n \log n})$ will be different).

Now we are ready to prove the strong combinatorial version:

▶ **Proposition 19.** *Let $\alpha = \mathrm{H}(p)$ and $\beta = \mathrm{H}(N(p, \tau))$. Let $A, B \subset \mathbb{B}^n$ and for every $x \in A$ the probability of the event "$N_\tau(x) \in B$" is at least $1/n$. If $\log \#A \geqslant \alpha n$, then $\log \#B \geqslant \beta n - O(\sqrt{n} \log^{3/2} n)$.*

Proof. As we have seen, the weak combinatorial version (Proposition 14) can be applied to the neighborhood B_d for $d = O(\sqrt{n \log n})$. The size of B_d can be bounded by the size of B multiplied by the size of a Hamming ball of radius d. The latter is $\mathrm{poly}(n) 2^{n\,\mathrm{H}(d/n)}$. Combining the inequalities, we get

$$\log \#B \geqslant \log \#B_d - n\,\mathrm{H}(d/n) - O(\log n) \geqslant \beta n - n\,\mathrm{H}(d/n) - O(\log n).$$

For small p we have $\mathrm{H}(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p} = p \log \frac{1}{p} + p + o(p) = O\left(p \log \frac{1}{p}\right)$. We have $p = d/n = O(\sqrt{\log n / n})$, so $n\,\mathrm{H}(d/n) = nO(\sqrt{\log n / n} \log n) = O(\sqrt{n} \log^{3/2} n)$, as promised. ◀

4 Complexity statement

Now we combine all pieces and prove Theorem 8. It states:

> Let $\alpha = \mathrm{H}(p)$ for some $p \leqslant 1/2$. Let τ be an arbitrary number in $(0,1)$. Let $\beta = \mathrm{H}(N(p,\tau))$. Then for sufficiently large n the following is true: for every string x of length n with $\mathrm{C}(x) \geqslant \alpha n$, we have $\Pr[\mathrm{C}(N_\tau(x)) \geqslant \beta n - o(n)] \geqslant 1 - \frac{1}{n}$.

Here $o(n)$ is actually $O(\sqrt{n}\log^{3/2} n)$.

We already have all the necessary tools for the proof, but some adjustments are needed. We already know how to convert a combinatorial statement into a complexity one. For that we consider the set B of all strings in $\mathbb{B}^n$ that have complexity less than $\beta n - c\sqrt{n}\log^{3/2} n$ for some c (to be chosen later). Then we consider the set A of all x such that $\Pr[N_\tau(x) \in B] \geqslant 1/n$. The combinatorial statement (strong version, Proposition 19) guarantees that $\#A \leqslant 2^{\alpha n}$. We would like to conclude that all elements of A have complexity only slightly exceeding αn. (Then we have to deal with this excess, see below.) For that we need an algorithm that enumerates A. First, we need to enumerate B, and for that it is enough to know n and the complexity bound for elements of B. But now (unlike the case of arbitrary change where we need to know only the maximal number of allowed changes) we need to compute the probability $\Pr[N_\tau(x) \in B]$, and the value of τ may not be computable, and an infinite amount of information is needed to specify τ. How can we overcome this difficulty?

Note that it is enough to enumerate some set that contains A but has only slightly larger size. Consider some rational τ' that is close to τ and the set $A' = \{x\colon \Pr[N_{\tau'}(x) \in B] > 1/2n\}$ The combinatorial statement remains true (as we noted in Remark 18, even $1/n^c$ would be OK, not only $1/2n$), so we may still assume that $\#A' \leqslant 2^{\alpha n}$. We want $A' \supset A$. This will be guaranteed if the difference between $\Pr[N_\tau(x) \in B]$ and $\Pr[N_{\tau'}(x) \in B]$ is less than $1/2n$. To use the coupling argument, let us assume that $N_\tau(x)$ and $N_{\tau'}(x)$ are defined on the same space: to decide whether the noise changes ith bit, we generate a fresh uniformly random real in $[0,1]$ and compare it with thresholds τ and τ'. This comparison gives different results if this random real falls into the gap between τ and τ'. Using the union bound for all bits, we conclude that $\Pr[N_\tau(x) \neq N_{\tau'}(x)]$ in this setting is bounded by $n|\tau' - \tau|$. Therefore, if the approximation error $|\tau' - \tau|$ is less than $1/2n^2$, we get the desired result, and to specify τ' that approximates τ with this precision we need only $O(\log n)$ bits. This gives us the following statement:

> for every string x of length n with $\mathrm{C}(x) \geqslant \alpha n + O(\log n)$, we have $\Pr[\mathrm{C}(N_\tau(x)) \geqslant \beta n - o(n)] \geqslant 1 - \frac{1}{n}$.

The only difference with the statement of Theorem 8 is that we have a stronger requirement $\mathrm{C}(x) \geqslant \alpha n + O(\log n)$ instead of $\mathrm{C}(x) \geqslant \alpha n$. To compensate for this, we need to decrease α a bit and apply the statement we have proven to $\alpha' = \alpha - O(\log n/n)$. Then the corresponding value of β also should be changed, to get a point (α', β') on the curve (Figure 1) on the left of the original point (α, β). Note that the slope of the curve is bounded by 1 (it is the case at the right end where the curve reaches $(1,1)$, since the curve is above the diagonal $\alpha = \beta$, and the slope increases with α due to convexity). Therefore, the difference between β and β' is also $O(\log n/n)$ and is absorbed by the bigger term $O(\sqrt{n}\log^{3/2} n)$.

Theorem 8 is proven.

In the next section we apply our technique to get some related results about infinite bit sequences and their effective Hausdorff dimension. We finish the part about finite strings with the following natural question.

▶ **Question 1.** *Fix some x and apply random noise N_τ. The complexity of $N_\tau(x)$ becomes a random variable. What is the distribution of this variable? The blowing-up lemma implies that it is concentrated around some value. Indeed, if we look at strings below 1%-quantile and above 99%-quantile, the blowing-up lemma guarantees that the Hamming distance between these two sets is at most $O(\sqrt{n})$, and therefore the thresholds for Kolmogorov complexity differ at most by $O(\sqrt{n}\log n)$ (recall that for two strings of length n that differ in i positions, their complexities differ at most by $O(i \log n)$, since it is enough to add information about i positions and each position can be encoded by $\log n$ bits).*

So with high probability the complexity of $N_\tau(x)$ is concentrated around some value (defined up to $O(\sqrt{n}\log n)$ precision). For each τ we get some number (expected complexity, with guaranteed concentration) that depends not only on n and $\mathrm{C}(x)$, but on some more specific properties of x. What are these properties? Among the properties of this type there exists a Vitanyi–Vereshchagin profile curve for balls, the minimal *complexity in the neighborhood as function of the radius (see [12, section 14.4]); is it somehow related?*

As we have mentioned, this question is open also for maximal *complexity in d-balls around x, not only for* typical *complexity after τ-noise.*

5 Infinite sequences and Hausdorff dimension

Let $X = x_1x_2x_3\ldots$ be an infinite bit sequence. The effective Hausdorff dimension of X is defined as $\liminf_{n\to\infty}(\mathrm{C}(x_1\ldots x_n)/n)$. A natural question arises: *what happens with the Hausdorff dimension of a sequence when each its bit is independently changed with some probability τ?* The following result states that the dimension increases with probability 1 (assuming the dimension was less than 1, of course), and the guaranteed increase follows the same curve as for finite sequences.

▶ **Theorem 20.** *Let $p, \tau \in (0, 1/2)$ be some reals, $\alpha = \mathrm{H}(p)$ and $\beta = \mathrm{H}(N(p, \tau))$. Let X be an infinite sequence that has effective Hausdorff dimension at least α. Then the effective Hausdorff dimension of the sequence $N_\tau(X)$ that is obtained from X by applying random τ-noise independently to each position, is at least β with probability* 1.

Proof. It is enough to show, for every $\beta' < \beta$, that the dimension of $N_\tau(X)$ is at least β' with probability 1. Consider $\alpha' < \alpha$ so that the pair (α', β') lies on the boundary curve. By definition of the effective Hausdorff dimension, we know that $\mathrm{C}(x_1\ldots x_n) > \alpha' n$ for all sufficiently large n. Then Theorem 8 can be applied to α' and β'. It guarantees that with probability at least $1 - 1/n$ the changed string has complexity at least $\beta' n - o(n)$. Moreover, as we have said, the same is true with probability at least $1 - 1/n^2$. This improvement is important for us: the series $\sum 1/n^2$ converges, so the Borel–Cantelli lemma says that with probability 1 only finitely many prefixes have complexity less than $\beta' n - o(n)$, therefore the dimension of $N_\tau(X)$ is at least β' with probability 1. ◀

In the next result we randomly change bits with probabilities depending on the bit position. The probability of change in the nth position converges to 0 as $n \to \infty$. This guarantees that with probability 1 we get a sequence that is Besicovitch-close to a given one. Recall that the Besicovitch distance between two bit sequences $X = x_1x_2\ldots$ and $Y = y_1y_2\ldots$ is defined as $\limsup_{n\to\infty}(d(x_1\ldots x_n, y_1\ldots y_n)/n)$, where d stands for the Hamming distance. So $d(X, Y) = 0$ means that the fraction of different bits in the n-bit prefixes of two sequences converges to 0 as $n \to \infty$. The strong law of large numbers implies that if we start with some sequence X and change ith bit independently with probability τ_i with $\lim_n \tau_n = 0$, we get (with probability 1) the sequence X' such that the Besicovitch distance between X and X' is 0. This allows us to prove the following result using a probabilistic argument.

▶ **Theorem 21.** *Let $X = x_1x_2\ldots$ be a bit sequence whose effective Hausdorff dimension is at least γ for some $\gamma < 1$. Let δ_n be a sequence of positive reals such that $\lim_n \delta_n = 0$. Then there exists a sequence $X' = x'_1x'_2\ldots$ such that:*

- *the Besicovitch distance between X and X' is 0;*
- *$\mathrm{C}(x'_1\ldots x'_n)$ is at least $n(\gamma+\delta_n)$ for all sufficiently large n.*

Proof. For this result we use some decreasing sequence $\tau_i \to 0$ and change ith bit with probability τ_i. Since $\tau_i \to 0$, with probability 1 the changed sequence is Besicovitch-equivalent (distance 0) to the original one. It remains to prove that the probability of the last claim (the lower bound for complexities) is also 1 for the changed sequence, if we choose $\tau_i \to 0$ in a suitable way.

To use different τ_i for different i, we have to look again at our arguments. We start with Proposition 13: the proof remains valid if each bit is changed independently with probability $\tau_i \geqslant \tau$ depending on the bit's position (see the arxiv version of the paper for details). Indeed, for every $\tau' \geqslant \tau$ the corresponding τ'-curve is above the τ-curve, so the pairs of entropies (original bit, bit with noise) are above the τ-curve and we may apply the same convexity argument.

The derivation of the combinatorial statement (first the weak one, then the strong one) also remains unchanged. The proof of the weak version does not mention the exact nature of the noise at all; in the strong version we use only that different bits are independent (to apply the McDiarmid inequality and the blowing-up lemma). The only problem arises when we derive the complexity version from the combinatorial one. In our argument we need to know τ (or some approximation for τ) to enumerate A. If for each bit we have its own value of τ, even one bit to specify this value is too much for us.

To overcome this difficulty, let us agree that we start with $\tau_i = 1/2$, then change them to $1/4$ at some point, then to $1/8$ etc. If for nth bit we use $\tau_n = 2^{-m}$, then to specify all the τ_i for $i \leqslant n$ we need to specify $O(m\log n)$ bits (each moment of change requires $O(\log n)$ bits). For $\tau = 2^{-m}$ we choose a pair (α,β) on the τ-curve such that $\alpha < \gamma < \beta$. To decide when we can start using this value of τ, we wait until $\mathrm{C}(x_1\ldots x_n) > \alpha n + O(m\log n)$ becomes true and stays true forever, and also $\gamma + \delta_n < \beta - O(\sqrt{n}\log^{3/2} n)$ becomes and stays true. Note that m is fixed when we decide when to start using $\tau = 2^{-m}$, so such an n can be found. In this way we guarantee that the probability that $x'_1\ldots x'_n$ will have complexity more than $(\gamma+\delta_n)$ is at least $1 - 1/n^2$ (we need a converging series, so we use the bound with n^2), and it remains to apply the Borel–Cantelli lemma. ◀

Theorem 21 implies that for every X that has effective Hausdorff dimension α there exist a Besicovitch equivalent X' that is α-random (due to the complexity criterion for α-randomness, see [5]), and we get the result of [5, Theorem 2.5] as a corollary. Moreover, we can get this result in a stronger version than in [5], since for slowly converging sequence δ_n, for example, $\delta_n = 1/\log n$, we get *strong* α-randomness instead of *weak* α-randomness used in [5]. (For the definition of weak and strong α-randomness and for the complexity criteria for them see [3, Section 13.5].)

Final remarks

In fact, if we are interested only in *some* increase of entropy when applying noise, and do not insist on the optimal lower bound, some simpler arguments (that do not involve entropy arguments and just prove the combinatorial statement with a weaker bound) are enough. One of them uses Fourier transform and was suggested by Fedor Nazarov; one can also use the hypercontractivity argument to improve this bound, but the resulting bound is not optimal either. Both arguments are explained in the arxiv version of the paper.

The arxiv version also contains the proof of the result from [1] (about the increase in entropy caused by random noise) for reader's convenience, and also because we need a slightly more general version for non-constant noise probability. Short (and quite standard) proofs of the McDiarmid inequality as a corollary of the Azuma–Hoeffding inequality, and of the Azuma–Hoeffding inequality itself are also given there to make the paper self-contained.

References

1 Rudolf Ahlswede, Peter Gács, and János Körner. Bounds on conditional probabilities with applications in multi-user communication. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 34(2):157–177, 1976.

2 Harry Buhrman, Lance Fortnow, Ilan Newman, and Nikolai K. Vereshchagin. Increasing Kolmogorov Complexity. In Volker Diekert and Bruno Durand, editors, *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 2005. doi:10.1007/978-3-540-31856-9_34.

3 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Theory and Applications of Computability. Springer, 2010. doi:10.1007/978-0-387-68441-3.

4 Peter Frankl and Zoltán Füredi. A short proof for a theorem of Harper about Hamming-spheres. *Discrete Mathematics*, 34(3):311–313, 1981. doi:10.1016/0012-365X(81)90009-1.

5 Noam Greenberg, Joseph S. Miller, Alexander Shen, and Linda Brown Westrick. Dimension 1 sequences are close to randoms. *Theoretical Computer Science*, 705:99–112, 2018. doi:10.1016/j.tcs.2017.09.031.

6 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

7 Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):3–11, 1965.

8 Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, Third Edition*. Texts in Computer Science. Springer, 2008. doi:10.1007/978-0-387-49820-1.

9 Grigory A. Margulis. Probabilistic properties of highly connected graphs. *Problems of Information Transmission*, 10(2):174–179, 1974.

10 Katalin Marton. A simple proof of the blowing-up lemma. *IEEE Transactions on Information Theory*, 32(3):445–446, 1986. doi:10.1109/TIT.1986.1057176.

11 Colin McDiarmid. *On the method of bounded differences*, pages 148–188. London Mathematical Society Lecture Note Series. Cambridge University Press, 1989. doi:10.1017/CBO9781107359949.008.

12 Alexander Shen, Vladimir A. Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*, volume 220. American Mathematical Society, 2017.

13 Nikolai K. Vereshchagin and Alexander Shen. Algorithmic statistics: forty years later. In *Computability and Complexity. Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, v. 10010*, pages 669–737. Springer, July 2017. arXiv:1607.08077.

14 Nikolai K. Vereshchagin and Paul M. B. Vitányi. Rate Distortion and Denoising of Individual Data Using Kolmogorov Complexity. *IEEE Transactions on Information Theory*, 56(7):3438–3454, 2010.

15 Aaron D. Wyner and Jacob Ziv. A Theorem on the Entropy of Certain Binary Sequences and Applications: Part I. *IEEE Transactions on Information Theory*, 19(6):769–772, 1973. doi:10.1109/TIT.1973.1055107.

A Unified Approach to Tail Estimates for Randomized Incremental Construction

Sandeep Sen
Department of CSE, I.I.T. Delhi, India
ssen@cse.iitd.ac.in

Abstract

By combining several interesting applications of random sampling in geometric algorithms like point location, linear programming, segment intersections, binary space partitioning, Clarkson and Shor [4] developed a general framework of randomized incremental construction (*RIC*). The basic idea is to add objects in a random order and show that this approach yields efficient/optimal bounds on **expected** running time. Even quicksort can be viewed as a special case of this paradigm. However, unlike quicksort, for most of these problems, sharper tail estimates on their running times are not known. Barring some promising attempts in [15, 3, 20], the general question remains unresolved.

In this paper we present a general technique to obtain tail estimates for *RIC* and and provide applications to some fundamental problems like Delaunay triangulations and construction of Visibility maps of intersecting line segments. The main result of the paper is derived from a new and careful application of Freedman's [9] inequality for Martingale concentration that overcomes the bottleneck of the better known Azuma-Hoeffding inequality. Further, we explore instances, where an *RIC* based algorithm may not have inverse polynomial tail estimates. In particular, we show that the *RIC* time bounds for trapezoidal map can encounter a running time of $\Omega(n \log n \log\log n)$ with probability exceeding $\frac{1}{\sqrt{n}}$. This rules out inverse polynomial concentration bounds within a constant factor of the $O(n \log n)$ expected running time.

2012 ACM Subject Classification Theory of Computation → Computational Geometry

Keywords and phrases ric, tail estimates, martingale, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.58

1 Introduction

One of the most natural and elegant paradigm for designing geometric algorithms is randomized incremental construction or *RIC* for short. It can be viewed as generalization of Quicksort and evolved over a sequence of papers [18, 2] eventually culminating in a very general framework of *configuration space* by Clarkson and Shor [4]. The basic procedure is described in Figure 1. Quicksort itself can be viewed through this paradigm as refinement of the current partially ordered set (partitions) by *inserting* the next splitter and updating the partitions.

An abstract configuration space, that we will refer to as $\Pi(S)$ is defined by the given set S of n elements. A configuration σ is a subset of the Euclidean space that is defined by $O(1)$ objects of S denoted by $d(\sigma)$. The *conflict list* of a configuration σ is denoted by $\ell(\sigma) = \sigma \cap \{S - d(\sigma)\}$, i.e. the elements of S that intersect σ, not including $d(\sigma)$. We define $\Pi^i(S) = \{\sigma : |\ell(\sigma)| = i\}$ and $\Pi(S) = \bigcup_{i=0}^{n} \Pi^i(S)$. For analyzing *RIC* , the properties of $\Pi^0(R)$ for a randomly chosen subset $R \subset S$ turns out to be very crucial. In particular, they characterize how the *uninserted* elements of S interact with the current partially constructed structure, denoted by $H(R)$. For notational simplicity, the conflict list of any $\sigma \in \Pi^0(R)$, $\ell(\sigma) = \sigma \cap S$ (instead of $\sigma \cap R$) which will be an important parameter in the analysis.

We illustrate the framework and the notations in the context of quicksort. A configuration σ is an interval defined by $\{x_i, x_j\} \in S$, i.e., $|d(\sigma)| = 2$ and $\ell([x_i, x_j]) = S \cap [x_i, x_j]$, i.e., the points of S that lie in this interval. The configurations space $\Pi(S)$ is the set of intervals

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 58; pp. 58:1–58:16

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

Procedure RIC(S).

$\mathcal{N} = [x_1, x_2 \dots x_n]$: a random permutation of S. ;
$T \leftarrow \phi$, H is some data structure appropriately initialized;
for $i = 1$ *to* n **do**
    $T \leftarrow T \cup \{x_i\}$;
    Update $H(T)$
Return $H(T)$;

Figure 1 Randomized Incremental Construction.

defined by all pairs $x_i, x_j \in S$. Note that $\Pi^0(S)$ consists of intervals defined by the sorted set of points in S. The associated data structure $H(R)$ for a sample $R \subset S$ may be thought of as an *incidence relation* between intervals $\sigma \in \Pi^0(R)$ and $\ell(\sigma) \in S - R$. This is precisely the ordered intervals induced by the points in R and the uninserted points $S - R$ partially ordered by these intervals.
When the next randomly chosen element $x \in S - R$ is added to R, $H(R)$ is updated to $H(R \cup \{x\})$ and this cost contributes to the running time of *RIC* . In [4] , the data-structure is maintained as a *conflict graph* that maintains relation between $\sigma \in \Pi^0(R)$ and $S - R$ as a bipartite graph. Although configurations are created and destroyed, the total cost can be shown to be twice the cost of new configurations created and the ones destroyed can be charged to the cost of its creation. In the case of quicksort, $H(S)$ yields the information about the sorted set since it contains all the ordered intervals. The reader is referred to [4, 17] for further details regarding this framework. We include a brief description in the Appendix.

Although the initial analysis in [4] was somewhat intricate and complex, subsequent papers [1, 19] simplified the analysis using a clever technique called *backward analysis*. In this paper, we will appeal to the simpler analysis. Often the full conflict graph information can be replaced by simpler relations (see [10, 19]. However, the conflict graph approach is very general and works for diverse problems.

A related, but a somewhat distinct approach was developed in the work of Seidel [19, 20, 1] that maintains a solution inductively, that is recomputed from scratch if the next insertion modifies it. For example, the closest pair can be computed in this similar manner ([12]). Although our techniques can be applied to the latter work also, we will focus primarily on the Clarkson-Shor incremental paradigm of a configuration space.

While the primary focus was on deriving bounds on the expected running time of *RIC*, it was clear that obtaining concentration bounds on the expected running time would make the *RIC* more practical and attractive. The conjecture was that the running times are concentrated around their expected values but to the best of our knowledge, there has been little progress in this direction barring some papers related to computing line segment intersections using *RIC* [3, 15] and on fixed dimensional linear programming [20]. For problems like planar hulls, high probability bounds can be proved based on linear ordering that do not extend to higher dimensions. Although, by resampling $\Omega(\log n)$ times, we can obtain inverse polynomial concentration bounds, it comes at the expense of the increasing the running time by an $O(\log n)$ factor.

In this paper, we revisit the problem and present a general methodology to obtain tail bounds for specific problems like *Delaunay triangulation, 3-D convex hulls, and line segment intersections* that are based on *RIC* - these are summarized in Table 1. For the case of

Table 1 Summary of results for some representative problems using our technique.
w.p. : "with probability" $\alpha(n)$: inverse Ackerman function, $\gamma > 0$ some constant
The third result alludes to a version that doesn't use conflict lists explicitly.

Problem	**Expected Running time**	**Tail estimates**
Quicksort	$O(n \log n)$	$O(c\gamma n \log n)$ w.p. $\geq 1 - n^{-c}$
Delaunay Triangulation	$O(n \log n)$	$O(c\gamma n \log n)$ w.p. $\geq 1 - 2^{-c}$
Segment intersections/ Trapezoidal maps	$O(n \log n + m)$ n: segments, m: intersections	$O(n \log n + m)$ w.p. $1 - \exp -(\frac{m+n \log n}{n\alpha(n)})$

finding intersection of line segments, our bounds are not only better than [15] but also distinctly less involved in terms of calculations. Moreover, we develop a unified approach based on a well-known Martingale inequality, unlike the previous approaches that were arguably more ad-hoc.

We also provide some evidence of the non-existence of such generalized tail estimates by constructing an instance of the trapezoidal maps (based on maintaining conflict lists) for which inverse polynomial tail estimates is possible only for running times exceeding $\Omega(n \log n \log \log n)$ and rules out concentration bounds within constant factor of expectation.

▶ Remark. In *RIC* based algorithms, the term *running time* is often interchangeably used with *structural changes* caused by each insertion, particularly when the underlying data structure is a conflict-graph.

1.1 Main techniques and organization

We begin by introducing a useful probabilistic inequality, viz., Freedman's inequality [9] for Martingales that will be used to model the running time of the generic *RIC* algorithms. In the following section, we illustrate the use of this technique for analyzing quicksort that can also be viewed within the framework of *RIC* . The application to quicksort doesn't yield any better result than what was previously known, but it provides a stepping stone to the more complex and general framework. In particular, even the more commonly used Azuma-Hoeffding bound is not known to be effective for quicksort concentration bounds because its dependence on the worst-case bound (sum of bounded differences).

It is unlikely that the previously known techniques for concentration bound of quicksort can be extended to generic *RIC* analysis as the intermediate structures in *RIC* are more complex and can be bound only in an *expected* sense. Starting with quicksort in section 3 which has a predictable intermediate structure consisting of $i + 1$ intervals in stage i, we tackle increasingly complex scenarios. In the case of Delaunay triangulation (in section 4), although the number of triangles in the i-th step is $O(i)$, the number of additional triangles created in the i-th step can be bound only in expectation. In section 5 we consider the case of line segment intersections where the size of the intermediate structure can be bound only in an expected sense and may have a large variance.

In the last section, we construct a family of inputs for the *RIC* for trapezoidal maps to show that inverse polynomial concentration bound is not attainable if we rely on conflict-list based update mechanism.

2 Basic framework and tools

Let $S = \{x_1, x_2 \dots x_n\}$ be a set of n objects. A permutation π of S is a 1-1 function $\pi(i) = j$ where $i, j \in \{1, 2, \dots n\}$ that produces a permutation $x_{\pi(1)}, x_{\pi(2)} \dots x_{\pi(n)}$. A *random* permutation of S is one of the $n!$ permutation function chosen uniformly at random. A *k-prefix* of a permutation π is the sequence of the first k objects and denoted by $\pi^{(k)}$ consisting of $x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)} \dots x_{\pi^{-1}(k)}$. Note that the permutation x_3, x_1, x_2 is defined as $\pi(1) = 2; \pi(2) = 3; \pi(3) = 1$, so the permutation is $x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, x_{\pi^{-1}(3)}$.

Let $X_1, X_2 \dots X_n$ where $X_i = x_{\pi^{-1}(i)}$ corresponding to a random permutation π. Further, let $\bar{X}^{(k)}$ denote a sequence of k random variables that will also be used it to refer to a fixed choice of the k variables, i.e., $\bar{X}^k = \pi^{(k)}$.

Let $(\Omega, \mathcal{U})$ denote the space of all possible permutations of n objects and $\mathcal{U}$ is the uniform probability distribution. For $0 \leq i \leq n$, let $\mathcal{B}_i$ consist of *blocks* of permutations, partitioned into some equivalent classes according to the i-prefixes where $\mathcal{B}_0$ is a single block consisting of all permutations. In this context, let us define $Z_i = \mathbb{E}[Z|\bar{X}^{(i)}]$ for any well-defined random variable Z over the probability space $(\Omega, \mathcal{U})$ where the conditioning is over the blocks of $\mathcal{B}_i$. The sequence Z_i defines a martingale sequence that is widely known as a *Doob Martingale* [5, 7, 8]. The reader may recognize that these blocks form the basis of a nested *filter* sequence that formally defines a martingale sequence.

It is more intuitive to visualize the above process as a tree τ, where the level i nodes correspond to *blocks* of $\mathcal{B}_i$ with arity $n - i$ and each sub-block is connected to its parent block by an edge directed from the parent. Any node in the j-th level of this tree can be labelled by the (unique) sequence $X^{(j)}$ leading to it. An edge is labelled by the (random) choice made at that level and also has an associated weight $w(\tau(\bar{X}^{(i-1)}), \tau(\bar{X}^{(i)}))$ that corresponds to the cost of the i-th incremental step. Here $\tau(\bar{X}^{(i)})$ represents the node of the tree τ after adding/deleting a sequence of random variable $\bar{X}^{(i)}$, we have We will use $\bar{w}$ to denote an upper bound of $w()$ in the context of specific algorithms. Let $Y = \sum_{j=0}^{j=n} w(\tau(\bar{X}^{(j-1)}), \tau(\bar{X}^{(j)}))$ be a random variable that corresponds to the sum of the cost of the edges on a path that corresponds to the cost of the RIC. Let $Y_i = \mathbb{E}[Y|\mathbb{F}_i] = \mathbb{E}[Y|\bar{X}^{(i)}]$.

Even if we interpret the sequence as *deletion* sequence starting from $\{x_1, x_2, x_3\}$, the reader can easily verify that this preserves the nesting property of blocks, and hence a valid filter. For example, $\mathcal{B}_1$ would contain the blocks $(x_1x_2x_3, x_1, x_3, x_2)$ corresponding to deletion of x_1 and $\mathcal{B}_2$ would contain the block (x_1, x_2, x_3) when x_2 is the next element deleted and so on. Since every insertion sequence has a unique backward deletion sequence, this interpretation amounts to running the *RIC* in the *reverse* direction with each step associated with the same cost as the forward direction. This perspective is is very similar to the idea of *backward analysis* [21].

In the *Doob's martingale* sequence, Y_0 denotes the expected running time of the RIC. We would like to bound the deviation $|Y_n - Y_0|$ for any run of the algorithm with *high probability*, which in the context of this paper will be inverse polynomial, unless otherwise mentioned. Likewise the random deletion sequence also defines a *Doob's martingale* on the subsets which will be referred to as the *backward-sequence* martingale (BSM henceforth).

The following martingale tail bound is the basis of many later results in this paper which is distinct from Azuma's inequality and referred to as the *Method of bounded variance*.

▶ **Theorem 1** (Freedman [9]). *Let $X_1, X_2 \dots X_n$ be a sequence of random variables and let Y_k, a function of $X_1 \dots X_k$ be a martingale sequence, i.e., $\mathbb{E}[Y_k|X_1 \dots X_k] = Y_{k-1}$ such that $\max_{1 \leq k \leq n}\{|Y_k - Y_{k-1}|\} \leq M_n$. Let*

$$W_k = \sum_{j=1}^{k} \mathbb{E}[(Y_j - Y_{j-1})^2 | X_1 \dots X_{j-1}] = \sum_{j=1}^{k} Var(Y_j | X_1 \dots X_{j-1})$$

where Var is the variance using $\mathbb{E}[Y_j] = Y_{j-1}$. Then for all λ and $W_n \leq \Delta^2$, $\Delta^2 > 0$,

$$\Pr[|Y_n - Y_0| \geq \lambda] \leq 2\exp\left(-\frac{\lambda^2}{2(\Delta^2 + M_n \cdot \lambda/3)}\right)$$

Note that the term Δ^2 can be bound by $\sum_{j=1}^{n} \max_{X_1, X_2 \ldots X_j} Var(Y_j | X_1 \ldots X_{j-1})$ i.e., the worst case bounds over all choices of length j prefix $X^{(j)}$. If the inner term can be bound by some function of j, say, $\omega(j)$, then we may obtain an upper bound on the probability of deviation for any sequence $\bar{X}^{(n)}$ as $\sum_{j=1}^{n} \omega(j)$ which can be viewed as a function of n.

Further, we will actually use a minor variation of this result (see [6, 13]). Suppose $\Pr[M_n \geq g(n)] \leq \frac{1}{f(n)}$ for some non-decreasing functions g, f. If we let B denote the event $M_n \geq g(n)$, then the overall bound becomes

$$\Pr[|Y_n - Y_0| \geq \lambda] \leq 2\exp\left(-\frac{\lambda^2}{2(\Delta^2 + g(n)\lambda/3)}\right) + \Pr(B) \tag{1}$$

Similarly it can also be extended to the case where $W_n \leq \Delta^2$ holds with probability $1 - \frac{1}{f(n)}$. Henceforth, in the remaining paper, we will appeal to the following version of Freedman's inequality where the bounds on M_n and W_n hold with high probability. Often the term $\frac{1}{f(n)}$ will be the dominant term, so the final tail bound will effectively be $O(\frac{1}{f(n)})$.

▶ **Corollary 2.** *Let* $\Pr[\max_{1 \leq k \leq n}\{|Y_k - Y_{k-1}|\} \geq M_n, W_n \geq \Delta^2] \leq \frac{1}{f(n)}$, *then,*

$$\Pr[|Y_n - Y_0| \geq \lambda] \leq 2\exp\left(-\frac{\lambda^2}{2(\Delta^2 + M_n \cdot \lambda/3)}\right) + O(1/f(n))$$

3 Application to Quicksort and related problems

Let us consider quicksort in the *RIC* framework and without loss of generality, let the input elements be $\{1, 2 \ldots n\}$. The j-th pivot, $1 \leq j \leq n$, partitions the input into $j + 1$ ordered sets, by splitting some existing partition P. Any element $x \in P$ is charged the cost of comparison with the pivot - any element $x' \notin P$ is not charged. The running time of the algorithm can be bound by the cumulative charges accrued by each element. In this analysis we will bound the charge of each element *with high probability* (w.h.p. henceforth) to denote probability exceeding $1 - 1/n^{\alpha}$ for some appropriate constant $\alpha > 0$, and the overall running time bound follows from multiplying by n.

The associated weight with each edge is either 1 or 0 depending on whether the latest random choice is one of the boundary elements of the interval containing x. We define a random variable $I_j^x = \{1$ if interval containing x changes in step j 0 otherwise$\}$. From backward analysis, the probability of this is at most $\frac{2}{j}$ for a uniformly chosen child node[1]. For completeness, we have included a detailed description of backward analysis in the appendix. We will also omit the superscript x and just use I_j since we will obtain a worst case bound over all choices of x. The reader may note that the bound on $\mathbb{E}[I_j]$ is only a function of j and not $\bar{X}^{(j)}$ over all random choices of *any* prefix of j elements.

[1] Using a simple trick of circular ordering (see [21]), this probability can be made exactly equal to $\frac{2}{j}$. Subsequently, Chernoff bounds can be applied easily by arguing about te independence of I_js.

It will also help to focus on the BSM for quicksort. A random deletion sequence creates a nested sequence of random subsets starting from the all the elements and ending in the empty sequence. An edge of this tree $(K, K - \{y\})$ is given a value 1 for a subset K and an element $y \in K$ if in the (forward) quicksort algorithm, selecting y as a pivot and leading to K (all the pivots selected) forces a comparison between y and x. Clearly two edges from any subset will be given a value 1, so that the expected cost for a random deletion is $\frac{2}{n-j}$ in the j-th level, $n \geq j \geq 0$. Figure 2 gives a depiction of this random variable in the quicksort process.

Consider a path $\mathcal{P} = v_0 v_1 \dots v_n$ from root to a leaf-node in this tree. The cost of this path is given by $w(\mathcal{P}) = \sum_{i=1}^{n} w(v_i, v_{i+1})$. A *random* path corresponds to one where v_{i+1} is a child of v_i chosen uniformly at random among the $n - i$ children. The expected cost of such a random path is given by

$$\mathbb{E}_{random\ \mathcal{P}}[w(\mathcal{P})] = \mathbb{E}[\sum_{i=1}^{n} w(V_i, V_{i+1})] \text{ where } V_{i+1} \text{ is a random child of node } V_i$$

We will follow the convention that small letters will denote fixed choices, i.e., v_i where the capital letters will correspond to random variables, i.e., V_i. Let $\mathbb{E}_j[Z]$ denote $\mathbb{E}_{X_j}[Z|\bar{X}^{(j-1)}]$ for some random variable Z. Note that $\bar{X}^{(j-1)}$ represents a fixed path from the root of the tree τ corresponding to the deletion sequence $X_1 X_2 \dots X_{j-1}$, to a level $j-1$ node, say V_{j-1}. Then,

$$\mathbb{E}_j[Y] = Y_j = \sum_{k=0}^{j-1} w(v_k, v_{k+1}) + \mathbb{E}_j\left[\sum_{k=j}^{n-1} w(V_k, V_{k+1})\right] = \sum_{k=0}^{j-1} w(v_k, v_{k+1}) + \sum_{k=j+1}^{n} \mathbb{E}[I_k]$$

It follows that $Y_0 = 2H_n$ and we want to obtain a tail estimate for $Y_n - Y_0$.

So,

$$Y_j - Y_{j-1} = w(v_{j-1}, v_j) + \left(\sum_{k=j+1}^{n} \mathbb{E}[I'_k]\right) - \left(\sum_{k=j}^{n} \mathbb{E}[I_k]\right) \tag{2}$$

$$= I_j - E[I_j] \text{ assuming } I_j, I'_j\text{'s have the same distribution} \tag{3}$$

To see this, the reader may recall that the probability that a fixed element x is affected by the pivot is a function only of the number of elements deleted (in the backward sequence) and not on the elements themselves or how they are distributed. So $\mathbb{E}_j[(Y_j - Y_{j-1})^2] = \mathbb{E}_j[(I_j - E[I_j])^2]$ which shows that the value of Y_j differs from Y_{j-1} because of the specific choice of the random variable X_j.

For quicksort, we can complete the analysis as follows.

$$\mathbb{E}_j[(I_j - E[I_j])^2] = \mathbb{E}_j[I_j^2] - \mathbb{E}_j^2[I_j] \ \leq \mathbb{E}_j[I_j^2] - \frac{4}{(n-j)^2}$$

$$\leq \frac{2}{n-j} \text{ since } I_j^2 \text{ is also a 0-1 indicator rv}$$

So $\sum_{j=1}^{n} \mathbb{E}_{j-1}[(Y_j - Y_{j-1})^2] \leq \sum_{j=1}^{n} \frac{2}{n-j} \leq 2H_n$where $H_n \leq \log n$. Plugging in $\lambda = 2c \log n$ for some constant c and using Freedman's theorem, we obtain

$$\Pr[|Y_n - Y_0| \geq c \log n] \leq \exp\left(-\frac{4c^2 \log^2 n}{2(\log n + c \log n/3)}\right) \leq \frac{1}{n^c}.$$

Note that $M_n = Y_i - Y_{i-1} \leq 1$.

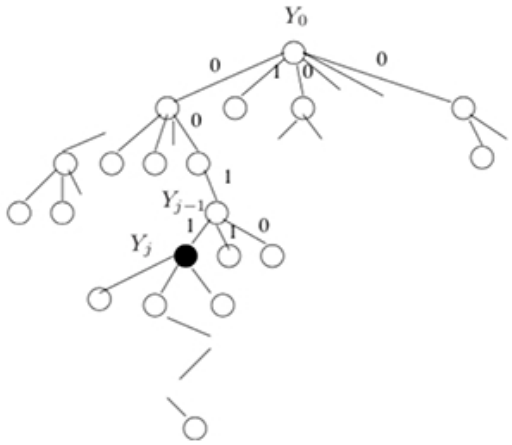


Figure 2 Tree corresponding to the Backward Sequence Martingale corresponding to the comparisons for a fixed element x. The root corresponds to Y_0 which denotes the expected running time. Every edge has cost 0 or 1 depending on whether x and X_i belong to the same interval and a path in this tree reveals the indicator variables I_j.

This shows that a single element incurs at most $O(\log n)$ cost with high probability and therefore quicksort runs in $O(n \log n)$ time with high probability.

▶ Remark. A straightforward application of the classic Azuma-Hoeffding bound [16] $\Pr[|Y_n - Y_0| \geq t] \leq \exp\left(\frac{-t^2}{\sum_{i=1}^{n} c_i^2}\right)$ would not have been effective since the the bound $c_i = M_n = 1$ makes the denominator too large for an $O(\log n)$ deviation bound. In [21], the author obtained a similar bound by using Chernoff bounds for binomial distribution that require *independence* of I_js across different levels. Also note that, there exists a superior bound of $O(n^{-\Omega(\log \log n)})$ for Quicksort obtained in [14].

The above argument can be directly extended to obtain a concentration bound on the *dart throwing* game that has many applications (Mulmuley [18]). Consider throwing n darts randomly in n ordered locations, say numbered $\{1, 2 \ldots n\}$. Let $S(i)$ be a random variable that denotes the *smallest* numbered location among the first i randomly thrown darts. Let $Z(i) = 1$ if $S(i) \neq S(i-1)$ and $Z(1) = 1$. So $Z(i)$ is the number of times $S(i)$ changes among the first i darts thrown. We are interested in $\mathbb{E}[Z(n)]$ which can be shown to be $\sum_{i=1}^{n} \frac{1}{i} = H_n$, the n-th harmonic. This follows from *backward analysis* by observing that among a set of i randomly chosen numbers, the probability of picking the smallest number as the last number is $\frac{1}{i}$. This is related to many visibility problems in geometry as well as the analysis of Trieps. Using the Freedman's inequality, we can easily show the following from the previous argument and looking at the changes in the leftmost interval induced by the darts.

▶ **Corollary 3.**

$$\Pr[|Z(n) - H_n| \geq 0.9 \log n] \leq \exp(-0.7 \log n) \leq \frac{1}{n^{0.7}}$$

This implies that $\Pr[0.1 \log n \leq Z(n) \leq 1.9 \log n] \geq 1 - n^{-0.7}$. The above result has been stated in a slightly weaker manner so that we can claim a lower bound on $Z(n)$ that will be invoked later to show the limitations of *RIC* .

The analysis in this section also extends to problems like constructing trapezoidal maps that can be used for point location (Seidel [19]). Since a trapezoid can be defined by at most 4 segments, the expected work for point location is $\sum_{i=1}^{n} \frac{4}{j} \leq 4 \log n$. Using a straightforward extension of the previous arguments, the following result can be obtained.

▶ **Lemma 4.** *Given a set of n non-intersecting line segments, a trapezoidal map can be constructed using RIC such that for any query point q, the number times the trapezoid containing q changes can be bound by $O(\log n)$ with inverse polynomial probability.*

This result will turn out to be very useful for some later results.

3.1 Extension to more general cost function

The bound obtained in Equation 3 can be extended to a more general situations of *RIC* where a single change can affect multiple "intervals" (more precisely, configurations). More specifically, when $\bar{w}$ not bounded by a constant we have the following generalization as long as $\mathbb{E}[w(V_{j-1}, V_j)]$ are same across all nodes in level $j-1$ for a random choice of the next node. Let $\mathcal{W}_j = w(V_{j-1}, V_j)$, then by generalizing the calculations in Equation 3, we obtain

$$\mathbb{E}_j[(Y_j - Y_{j-1})^2] \leq \mathbb{E}_j[\mathcal{W}_j^2] - \mathbb{E}_j^2[\mathcal{W}_j] \leq \mathbb{E}_j[\mathcal{W}_j^2] \tag{4}$$

Although the generalized analysis of *RIC* is not described here is details, we appeal to the intuitions of the reader that the expected cost of the j-th step depends on j and not the actual choice of the elements - see Equation 5. Note that in the general analysis of *RIC* , we obtain an upper bound on the $\mathbb{E}[\mathcal{W}_j]$ as a function of j. The upper bounds can be considered as the (identical) expected cost of the i-th step and the martingale bounds can be applied on these costs, so the final bounds would still hold as deviation from this (uniform) expected upper-bounds.

3.2 Comparison with an earlier bound

We briefly recall the framework of Mehlhorn, Sharir and Welzl [15] to model the general RIC algorithm. A rooted (n, r) tree T is either a single node for $r = 0$ or (for $r > 0$) the tree has n children which are recursively defined $(n-1, r-1)$ subtrees. Each of the n edges has an associated weight d_i corresponding to the i-th child and $\max_{i=1}^{n} d_i \leq d(n)$ and $\sum_i d_i \leq M(n)$. The expected cost of a path in this recursively defined tree is $A = \sum_{i=1}^{n-1} \frac{M(n-i)}{(n-i)}$. One of the main results in the paper is the following tail bound (Theorem 1 in [15]).

$$\Pr(X \geq B) \leq \left(\frac{e}{1 + B/A}\right)^{B/d(n)} \text{ for all } B \geq 0$$

Although this bound looks somewhat simpler to use, this is not directly comparable to Freedman's bound except for some special cases like Lemma 4 and quicksort where the concentration results are similar. It may be noted that the authors [15] analyze the backward execution of the algorithm for these results. This bound becomes weaker if $d(n)$ is not a constant - for some of the later applications $d(n)$ may be larger than A in the worst case. The authors improve the bound for the specific problem of building visibility maps of line segments by using the expected value of $M(n)$. However, there is no generalization given for other problems.

4 Incremental Delaunay Triangulation

We will now consider somewhat more complex scenarios like construction of Delaunay Triangulation and three dimensional convex hull (see Guibas Knuth and Sharir [10]). Broadly speaking these algorithms have two distinct components -

(i) Updating the (partial) structure of the points inserted thus far.
(ii) Updating the point-location data structure of the uninserted points.

For concreteness, we will address the problem of Delaunay Triangulation. The analysis corresponding to updating the point location structure is similar to the analysis of quicksort given above. For the update of structural complexity, it was shown in [10] that the expected cumulative structural change can be bound by $O(n)$, whereas for the latter, the expected work over all the n (random) insertions sequence $O(n \log n)$. We will do a combined analysis since we are interested in obtaining tail estimates on the work including all data structural updates.

In the remaining part of the paper, we will be alluding to the BSM framework and make use of Equation 4 for deriving the tail estimates. To avoid any confusion, we will use stage/level k to refer to the forward algorithm when k objects have been added and do all calculations in this order. Although the martingale has been defined for the backward execution, substituting $n - k$ by k, consistently will not not affect anything except the order of the summations. This will also help us use the random sampling bounds without having to restate them in the flipped order.

We will make extensive use of the following result of [4, 11].

▶ **Theorem 5.** *At any stage i of the RIC of Delaunay triangulation, the i randomly chosen points is a uniform random subset of the n points. So the number of unsampled points within each triangle is bounded by $O(\frac{n}{i} \log n)$ with probability $1 - 1/n^c$ for any constant $c > 1$. Moreover, all the triangles that emerges in the course of edge flips also satisfy the above bounds.*

▶ Remark.

(i) A random sample of size $r + 1$ is constructed by adding a random element to a sample of size r. So the properties of random sampling applies to the intermediate steps as well with high probability. In general, we will use a similar bound on $\max_{\sigma \in \Pi^0(R)} |\ell(\sigma)|$. This is crucial for the application of the generalized version of Freedman's inequality given in Equation 1 where the event B contains all insertion sequences where the bound in Theorem 5 fails during one of more stages. In other words, the complement of B consists of all insertion sequences where the bound holds during all stages.
(ii) All the triangles that show up in the course of edge flips belong to $\Pi^0(R)$. Although some of them are not Delaunay triangles and therefore, only temporary, they can contribute to the running time, depending on the data structure one maintains for the intermediate partitions.

To apply Freedman's bound, we will bound the variance. Unlike the analysis of quicksort, we will consider the work done for all the n points (actually $n - i$ uninserted points in stage i) together. Each edge flip involves four triangles - two old and two new and redistributes the points in the two new triangles. Since each triangle contains $O(\frac{n}{i} \log n)$ points w.h.p, each edge flip can be be done in $O(\frac{n}{i} \log n)$ w.h.p. Since the maximum degree of a Delaunay graph of i points is i, the total number of edge flips in the i-th stage is bounded by i. Therefore we can claim the following.

▶ **Lemma 6.** *The work in stage i of the algorithm, $i \leq n$ can be bound by $O(n \log n)$ w.h.p.*

Let $\Pi_s(R)$ denote the configurations in $\Pi^0(R \cup s)$ adjacent to s (or defined by s). The following claims can be easily derived from some general random-sampling lemmas in [4]

▶ **Lemma 7.** *(i)* $\mathbb{E}[\sum_{\sigma \in \Pi_s(R)} \ell(\sigma)] = O(\frac{n}{r})\mathbb{E}[|\Pi_s(R)|]$
(ii) $\mathbb{E}[\sum_{\sigma \in \Pi_s(R)} \ell^2(\sigma)] = O(\frac{n^2}{r^2})\mathbb{E}[|\Pi_s R)|]$

Bounding Variance. We will need the following result

▶ **Lemma 8.** *For real numbers x_i $1 \leq i \leq r$* $\left(\sum_{i=1}^r x_i\right)^2 \leq r\left(\sum_{i=1}^r x_i^2\right)$

Proof. Using the convexity of the square function, from Jensens inequality it follows that $\frac{\sum_{i=1}^r x_i^2}{r} \geq \left(\frac{\sum_{i=1}^r x_i}{r}\right)^2$. Multiplying both sides by r^2 yields the required result. ◀

Let R^k denote the random subset of the first k sites[2] and let $DT(R^k)$ represent the Delaunay triangulation of R^k which is a planar graph having $2k - h_k - 2$ triangles and $3k - h_k - 3$ edges where h_k is the number of points on the convex hull of R^k. The work done when a site $v \in DT(R^k)$ is picked by the *RIC* , is proportional to the number of unsampled points in the triangles adjoining v. If $l(\sigma)$ is the number of points in a triangle σ, then the work is proportional to $T_k = \sum_{\sigma \in \Delta(v)} l(\sigma)$ where $\Delta(v)$ denotes triangles adjacent to v. Squaring T_k and taking expectation

$$\begin{aligned}
\mathbb{E}[T_k^2] &= \frac{1}{k}\mathbb{E}[\sum_{v \in R^k}(\sum_{\sigma \in \Delta(v)} l(\sigma))^2] \leq \frac{1}{k}\mathbb{E}[\sum_{v \in R^k}\sum_{\sigma \in \Delta(v)} |\Delta(v)| l^2(\sigma)] \text{ from Lemma 8} \\
&= \frac{1}{k}\sum_{v \in R^k} |\Delta(v)|\mathbb{E}[\sum_{\sigma \in \Delta(v)} l^2(\sigma)] \leq \frac{1}{k}\sum_{v \in R^k} \frac{n^2}{k^2}|\Delta(v)|^2 \text{ from Lemma 7} \\
&= O(\frac{n^2}{k}) \text{ as } \sum_v |\Delta(v)|^2 = O(\sum_v \Delta(v)^2) = O(k^2)
\end{aligned}$$

Following Equation 4, this yields $W_n \leq \sum_{k=1}^{k=n} \mathbb{E}[T_k^2] \leq \sum_{k=1}^{k=n} O(\frac{n^2}{k}) = O(n^2 \log n)$. Plugging the bound of $M_n = O(n \log n)$ from Lemma 6 in Freedman's theorem, we obtain the following bound.

▶ **Lemma 9.** *Let $T(n)$ denote the running time of ric based construction of Delaunay Triangulation and let $\lambda = cn \log n$ for a suitable constant c. Then*

$$\Pr[T(n) \geq \alpha(n)\lambda] \leq \exp\left(-\frac{(\alpha(n)cn \log n)^2}{2(n^2 \log n + \alpha(n)c \cdot n^2 \log^2 n/3)}\right) \leq \exp(-\alpha(n))$$

▶ Remark. The above Lemma gives high probability bound for $T(n)$ exceeding $\Omega(n \log^2 n)$ for $\alpha = \Omega(\log n)$. However, this bound is superior to the straightforward Markov's bound applied on the expected work as well as preferable to restarting the original algorithm using independent random bits each time.

[2] We will use this term to distinguish between the input points defining the triangulation and the unadded points

This analysis can be extended to the three-dimensional convex hull algorithm presented in Mulmuley [17]. In [3], the authors obtain similar tail estimates for the for the space complexity (alternately referred to as *conflict history*) of the algorithm of [10]. For fixed dimensional linear programming Seidel [20] proved a similar property and this can be extended to *RIC* algorithms like closest pair [12]. These algorithms typically have the property, that in stage i, with probability $\Omega(\frac{1}{i})$, the algorithm re-builds the data structure. This makes inverse polynomial bound challenging - say for $i = \sqrt{n}$, the *RIC* for Delaunay triangulation could spend $O(n \log(\sqrt{n})$ time to rebuild the associated point location data structure if the i-th point has degree $\Omega(i)$.

5 More generalized RIC: segment intersections

We now consider a more general scenario in RIC (Randomized Incremental Construction). Using a *conflict graph* update model of RIC (see Appendix), we obtain the following expression for expected work.

$$\mathbb{E}[\text{ \#edges created in the conflict graph}] = \sum_{\sigma \in \Pi^0(R \cup s)} l(\sigma) \cdot \Pr\{\sigma \in \Pi^0(R \cup s) - \Pi^0(R)\}$$

From *backward analysis* this probability is the same as deleting a random element from $R \cup s$ which is $\frac{d(\sigma)}{r+1}$ where $r = |R|$. By substituting this we obtain

$$\sum_{\sigma \in \Pi^0(R \cup s)} l(\sigma) \cdot \frac{d(\sigma)}{r+1} = \frac{d(\sigma)}{r+1} \sum_{\sigma \in \Pi^0(R \cup s)} l(\sigma) = O(\frac{d(\sigma)}{r} \cdot \frac{n}{r} \mathbb{E}[\Pi^0(R \cup s)]) \tag{5}$$

Therefore the expected work over the sequence of random insertions is $\sum_{r=1}^{n} O(\frac{d(\sigma)}{r} \cdot \frac{n}{r} \mathbb{E}[\Pi^0(R \cup s)])$.

For the case of line segment intersections, it can be shown that $\mathbb{E}[\Pi^0(R \cup s)] = O(r + \frac{m \cdot r^2}{n^2})$ from which it follows that the expected work is

$$\sum_{r=1}^{n} O(\frac{d(\sigma)}{r} \cdot \frac{n}{r} \cdot O(r + \frac{m \cdot r^2}{n^2}) = \sum_{r=1}^{n} \left(\frac{dn}{r} + \frac{dm}{n} \right) = O(n \log n + m).$$

Here $d(\sigma) \leq 6$ which the maximum number of segments that define a σ (trapezoid in this case).

Tail bounds for this problem has been elusive despite some significant attempts (see [15]). We will show that our previous methodology can be extended to obtain tail estimates on the work done.

Consider an arrangement of n segments with m intersections ($0 \leq m \leq \binom{n}{2}$). In the trapezoidal map $\mathcal{T}$ of the n segments (also known as a vertical visibility diagram), let us denote the set of trapezoids adjacent to segment s_i by T_i. Any trapezoid $\sigma \in \mathcal{T}$ is defined by at most six segments. Since it is a planar map, and there are at most $2n + 2m$ vertices, it follows that $\sum_i |T_i| = O(n + m)$. We would like to obtain a bound on $\sum_i |T_i|^2$. Let us denote by n_i and m_i respectively, the number of segments end-points and intersection points visible to segment s_i. It follows that $|T_i| = O(n_i + m_i)$ and

$$\sum_i |T_i|^2 = O(\sum_i (n_i + m_i)^2) = O(\sum_i n_i^2 + \sum_i n_i \cdot m_i + \sum_i m_i^2)$$

where $m_i \leq O(n)$ from the zone theorem bound. Moreover $\sum_i n_i = O(n)$ and $\sum_i m_i = O(m)$ as each point is visible from the closest segments above and below.

The first expression can be bound by $(\sum_i n_i)^2 = O(n^2)$ and the second expression by $2(\sum_i n_i) \cdot (\sum_i m_i) = O(n \cdot m)$ (Cauchy-Schwartz inequality). The third expression is less than $(m/n) \cdot n^2 = mn$. So, the overall expression can be bound by $O(m \cdot n + n^2)$.

For a uniformly chosen sample R^k of size k, the expected number of intersections in the sample is $\frac{mk^2}{n^2}$, so the variance can be bound by

$$\mathbb{E}[T_k^2 | R^k] = \frac{1}{k}\mathbb{E}[\sum_{s_i \in R^k} (\sum_{\sigma \in T_i} l(\sigma))^2]$$

To simplify calculations, we recall (Theorem 5) that $l(\sigma) \leq O(\frac{n \log n}{k})$ with high probability. So, plugging this in the previous expression, and using the previous bound on $\sum_i |T_i|^2$, we obtain (w.h.p.)

$$\mathbb{E}[T_k^2 | R^k] \leq \frac{1}{k} \cdot O(\frac{n^2 \log^2 n}{k^2}) \cdot \mathbb{E}[k^2 + k \cdot m_k]$$

where m_k is the number of intersections in R^k. Taking expectation over all choices of R^k, we obtain the unconditional expectation as

$$\mathbb{E}[T_k^2] \leq \frac{n^2 \log^2 n}{k^3} \cdot \mathbb{E}[k^2 + k m_k] \leq \frac{n^2 \log^2 n}{k^3} \cdot (k^2 + \frac{mk^3}{n^2})) \leq \frac{n^2 \log^2 n}{k} + m \log^2 n$$

This uses the bound $\mathbb{E}[m_k] = O(k + \frac{m \cdot k^2}{n^2})$. This bound is relevant for the maintenance of conflict graphs.

In contrast, for an algorithm like Mulmuley [18], where only the trapezoids are maintained, the work done[3] can be bound by using $l(\sigma) = 1$ in the expression for $\mathbb{E}[T_k^2]$. This yields $\mathbb{E}[T_k^2] = O(k + m\frac{k^2}{n^2})$. We shall return to this case later.

So

$$W_n \leq \sum_{k=1}^{n} \mathbb{E}[T_k^2] \leq O(n^2 \log^3 n + mn \log^2 n)$$

To obtain high probability bounds using Freedman's inequality, we want to bound this expression by $\frac{\lambda^2}{\log n}$ where $\lambda = c(n \log n + m)$. M_n and $M_n \cdot \lambda$ can be bound by $O(n \log n \cdot \alpha(n))$ and $mn\alpha(n)$ respectively with high probability. This follows from a bound of $O(t\alpha(t))$ on the zone of a segment that intersects t segments in an arrangement of n segments ([18]).

So $\frac{\lambda^2}{W_n + M_n \cdot \lambda}$ can be bound by

$$\frac{\Omega(m^2 + mn \log n + n^2 \log^2 n)}{O(mn \log^2 n + n^2 \log^3 n + mn\alpha(n) \log n + n^2 \alpha(n) \log^2 n} = \frac{\Omega(m^2 + mn \log n + n^2 \log^2 n)}{O(mn \log^2 n + n^2 \log^3 n}.$$

So from Freedman's inequality we obtain a tail bound of $\exp(-\frac{m^2}{mn \log^2 n})$ for $m \geq n \log^2 n$.

▶ **Theorem 10.** *Let $T(n)$ represent the work done in the conflict-graph based segment intersection algorithm, then there exists constant β, such that for $m \geq \beta n \log^2 n$, $\Pr[T(n) \geq m] \leq \exp(-\frac{m}{n \log^2 n})$.*

To the best of our knowledge, no prior concentration bound was known for the conflict-graph based approach for segment intersection given by Clarkson and Shor [4]. The paper by [15] noted that their methods could not be extended to this algorithm.

[3] there is some additional cost for point location that can be bound using Lemma 4

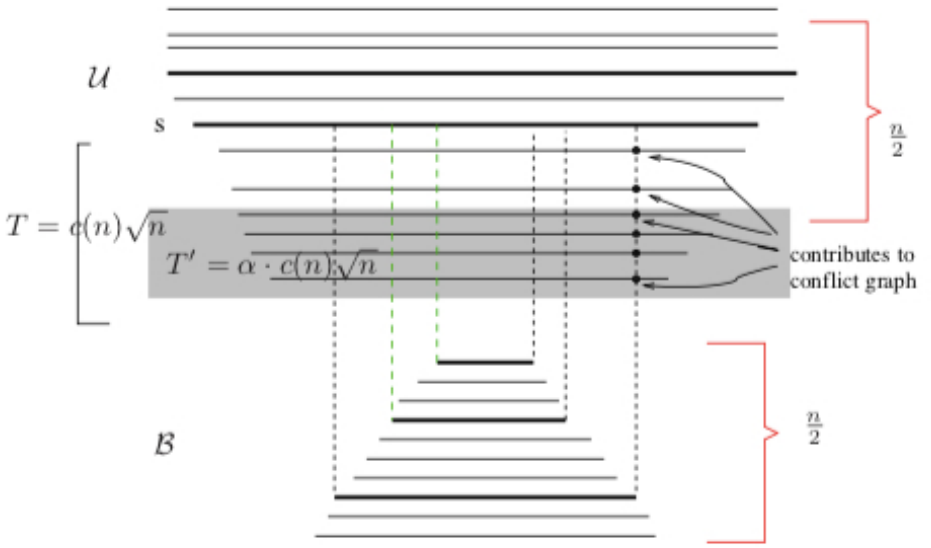


Figure 3 A bad input for segments intersections (trapezoidal maps). The thicker segments correspond to the sampled set.

For the specific case of $m = 0$, the bound can be improved by observing that the zone of a segment can be at most $O(n)$ (instead of $n\alpha(n)$) as there are no intersections. Setting $m = 0$ in the previous bound for W_n, we obtain the following

▶ **Corollary 11.** *For constructing the trapezoidal map of n non-intersecting line segments using RIC , the work done $T(n)$ satisfies $\Pr[T(n) \geq c\beta n \log^2 n] \leq n^{-\beta^2}$ for some constant $c > 1$.*

This shows that we can obtain inverse polynomial concentration bounds around a running time that exceeds the expected running time by a factor of $\Omega(\log n)$.

We now return to the algorithms of [18] and [4] that do not maintain conflict-graphs but only involves segment end-points. As observed before, the quantity W_n can be bound by $\sum_{k=1}^{n} O(k + m\frac{k^2}{n^2}) = O(n^2 + mn)$. We summarize as follows.

▶ **Lemma 12.** *In the segment intersection algorithms of [18, 4] that do not maintain conflict-graphs explicitly, the probability that the work exceeds $c(m + n \log n)$ can be bound by*

$$\exp - \left(\frac{\Omega(m^2 + mn \log n + n^2 \log^2 n)}{O(mn\alpha(n) + n^2\alpha(n) \log n)} \right) \leq \exp(-\frac{\log n}{\alpha(n)}) \text{ since } M = O(n\alpha(n)).$$

For $m \geq n \log n$, the bound improves to $\exp(-\frac{m}{n\alpha(n)})$.

▶ Remark. This bound is better than the results in [15] where the authors show that for some constant $\delta > 0$. $\Pr[T(n) \geq Cm] \leq \exp\left(\frac{-\delta m}{n \log n}\right)$ for $m \geq n \log n \log\log\log n$.

6 Can we improve the tail bounds

Figure 3 shows an input of n horizontal segments divided into two sets $\mathcal{U}, \mathcal{B}$ each of which has $n/2$ segments. In the top pile $\mathcal{U}$ of the $n/2$ horizontal segments, let T denote the lowest $\sqrt{n} \cdot c(n)$ segments for some function $c(n)$ that will determined in the analysis.

We will consider the *RIC* after the first $3\sqrt{n}$ insertions. We want to consider the event that at least $\sqrt{n}$ segments are chosen from $\mathcal{B}$ and no segment is chosen from T. The probability of the former is $1 - 2^{-\sqrt{n}}$ (from Chernoff bounds) whereas the probability of the latter can be easily seen as $(1 - \frac{c(n)}{\sqrt{n}})^{3\sqrt{n}} = \Omega(4^{-3c(n)})$, using $(1-x) \geq \exp -(x + x^2) \geq \frac{1}{4^x}$ $x \in [0, 1/2]$. Since the two events are independent, the probability of their intersection, which will be denoted by the event $\mathcal{E}_1$ is nearly $\Omega(4^{-3c(n)})$ (since $1 - 2^{-\sqrt{n}} \to 1$).

Notice that, if the lowest sampled segment is $s \in \mathcal{U}$ then all the segments below s will intersect the vertical lines through the end-points of the trapezoids defined by the sampled segments in $\mathcal{B}$. Namely, if there are m unsampled segments below s, then the size of the conflict graph is at least $\sqrt{n} \cdot m$. Every time s changes, new edges are created in the conflict graph by the sampled edges in $\mathcal{B}$. Following the choice of the initial $3\sqrt{n}$ segments, let us consider subsequent sampling by the *RIC* where segments from T will be sampled. Within T, let us denote by T' the lowest $\alpha c(n)\sqrt{n}$ segments for some constant $\alpha < 1$ and let T'' denote the remaining segments. Let $\mathcal{E}_2$ denote the event that among the first $\log n$ segments sampled from T, none are from T'. This can be calculated as $(1 - \alpha)^{\log n} = \Omega(4^{-\alpha \log n})$.

From our earlier analysis, the lowest sampled segment in T'' changes about $\theta(\log \log n)$ times with probability $\frac{1}{\log n}$ (Corollary 3) - note that this holds regardless of the event $\mathcal{E}_2$. So, in the second phase of *RIC* , at least $\Omega(\alpha\sqrt{n}c(n) \cdot \sqrt{n} \log \log n)$ edges are created with probability $\Omega(\frac{2^{-\alpha \log n}}{\log n})$.

So, by unconditioning the first phase event $\mathcal{E}_1$, we obtain
$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2] = \Pr[\mathcal{E}_2|\mathcal{E}_1] \cdot \Pr[\mathcal{E}_1] = \Omega(\frac{1}{\log n} n^{-\alpha} \cdot 2^{-6c(n)})$. This yields the following result

▶ **Theorem 13.** *There exists inputs for which the conflict-graph based RIC algorithm for constructing vertical visibility maps (segment intersections with no intersections) encounters* $\Omega(\alpha c(n)\sqrt{n} \log \log n)$ *structural changes with probability* $\Omega(e^{-6c(n)-\alpha})$ *for* $c(n) = o(\sqrt{n})$.

In particular, by choosing $c(n) = \frac{\log n}{13}$, and a small α, the conflict graph based *RIC* algorithm may encounter $\Omega(n \log n \log \log n)$ changes with probability $\Omega(\frac{1}{\sqrt{n}})$ that rules out inverse polynomial bounds for a total work of $O(n \log n)$. Note that $\frac{1}{\sqrt{n}}$ can be easily increased to $\frac{1}{n^\epsilon}$ for any $\epsilon > 0$ by an appropriate choice of $c(n)$.

The reader may compare this result with Corollary 11 to get a sense of the gap between upper and lower bounds of the tail estimates. Coming up with similar examples for the other problems discussed in this paper, like Delaunay Triangulation would be an interesting exercise and shed more light on the behavior of *RIC* .

References

1 P. Chew. The simplest Voronoi diagram algorithm takes linear expected time. In *Manuscript*, 1988.

2 Kenneth L. Clarkson. Applications of Random Sampling in Computational Geometry, II. In *Symposium on Computational Geometry*, pages 1–11. ACM, 1988.

3 Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four Results on Randomized Incremental Constructions. In *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, pages 463–474, 1992.

4 Kenneth L. Clarkson and Peter W. Shor. Application of Random Sampling in Computational Geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

5 J.L. Doob. Regularity properties of certain families of chance variables. *Transactions of the American Mathematical Society*, 47 (3):455—-486, 1940.

6 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

7 W. Feller. *An Introduction to Probability Theory, Vol. 1.* Wiley, New York, NY, third edition, 1968.

8 W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 2.* Wiley, New York, NY, second edition, 1971.

9 David Freedman. On tail probabilities on martingales. In *The Annals of Probability*, volume 3(1), pages 100–118, 1975.

10 Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized Incremental Construction of Delaunay and Voronoi Diagrams. *Algorithmica*, 7(4):381–413, 1992.

11 David Haussler and Emo Welzl. Epsilon-Nets and Simplex Range Queries. In *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry, Yorktown Heights, NY, USA, June 2-4, 1986*, pages 61–71, 1986.

12 S. Khuller and Y. Matias. A Simple Randomized Sieve Algorithm for the Closest-Pair Problem. *Information and Computation*, 118(1):34–37, 1995.

13 C. McDiarmid. Concentration. *Probabilistic Methods for Algorithmic Discrete Mathematics*, 16, Algorithms and Combinatorics:195—-248, 1998.

14 Colin McDiarmid and Ryan Hayward. Large Deviations for Quicksort. *J. Algorithms*, 21(3):476–507, 1996.

15 Kurt Mehlhorn, Micha Sharir, and Emo Welzl. Tail Estimates for the Efficiency of Randomized Incremental Algorithms for Line Segment Intersection. *Comput. Geom.*, 3:235–246, 1993.

16 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

17 K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice-Hall, 1994. URL: `https://books.google.com/books?id=rjgZAQAAIAAJ`.

18 Ketan Mulmuley. A Fast Planar Partition Algorithm, I (Extended Abstract). In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 580–589, 1988.

19 Raimund Seidel. A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons. *Comput. Geom.*, 1:51–64, 1991.

20 Raimund Seidel. Small-Dimensional Linear Programming and Convex Hulls Made Easy. *Discrete & Computational Geometry*, 6:423–434, 1991.

21 Raimund Seidel. Backwards Analysis of Randomized Geometric Algorithms. In *In: Pach J. (eds) New Trends in Discrete and Computational Geometry. Algorithms and Combinatorics*, volume 10. Springer, Berlin, Heidelberg, 1993.

A Appendix

We provide a brief description of the notations and definitions that capture the framework of RIC and its analysis in very general setting.

Given a set S of n elements (like points, segments, lines etc.), a configuration σ is defined by at most d objects where d is $O(1)$. The set of objects is denoted by $d(\sigma)$ and the number of configurations is bounded by n^d if there are no more than $O(1)$ configurations associated each subset of d elements (there can be more than one configuration associated with the same $d(\sigma)$ elements.

Let $\ell(\sigma) = S \cap \sigma - d(\sigma)$ be the elements that intersect with σ. With a slight overloading of notation we will also use $\ell(\sigma)$ to denote the set of the intersecting elements with σ also. Let $\Pi^i(S)$ denote the set of configurations σ with $\ell(\sigma) = i$. We use $\Pi(S) = \cup_i \Pi^i(S)$ to denote all configurations. For any subset $R \subset S$, we use $\Pi(R)$ to denote the configurations defined by elements of R and the *conflict list* of any configuration $d(\sigma) \subset R$ as $\sigma \cap S$, i.e., all the elements and not just the elements in R.

A *conflict graph* represents the relation between the configurations in $\Pi^0(R)$ and the corresponding conflict list, which is a bipartite graph with configurations in $\Pi^0(R)$ on one side and the uninserted elements on the other side. Randomized Incremental construction

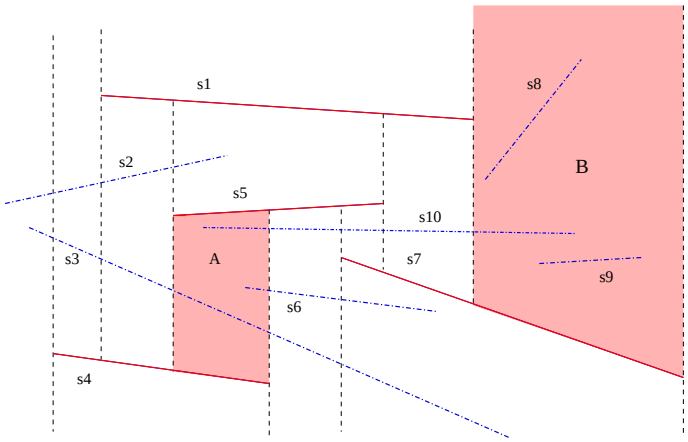


Figure 4 The red segments are sampled segments and blue segments are unsampled. The trapezoids A, B are configurations that belong to $\Pi^0(R)$. Here $d(A) = \{s_4, s_5\}$ $\ell(A) = \{s_3, s_6, s_{10}\}$.

can be thought of as maintaining and update of the conflict graph starting with $R = \phi$ and successively adding a random (uninserted) element $e \in S - R$ into R. This introduces $\sigma \in \Pi^0(R \cup e) - \Pi^0(R)$ requiring appropriate changes in the conflict graph. To illustrate this framework on qiucksort, we define the configurations as intervals defined by a pair of elements $[x_i, x_j]$ where $x_i < x_j$. Initially there is the hypothetical configuration $(-\infty, +\infty)$. As we introduce more pivots, we maintain the ordered set of intervals induced by the elements chosen as pivots. As we introduce a pivot, some interval is split. Eventually we have the sorted set defined by consecutive intervals. When an interval $[x_i, x_j]$ splits because of a pivot element y such that $x_i < y < x_j$, the elements in $\ell([x_i, x_j]) \cap S$ is reassigned to $\ell([x_i, y])$ and $\ell([y, x_j])$ appropriately. The number of comparisons required is roughly $|\ell([x_i, x_j]) \cap S|$ (the cardinality).

The analysis of quicksort in this framework can be done using the technique of *backward analysis* which is very elegant. Let us assign an indicator random variable X_k associated with an element x, such that

$$X_k = \begin{cases} 1 & \text{if } x \text{ is compared for the } k\text{-th pivot} \\ 0 & \text{otherwise} \end{cases}$$

The number of comparisons involving x is given by $\sum_{k=1}^{n} X_k$. Therefore

$$\mathbb{E}[\sum_{k=1}^{n} X_k] = \sum_{k=1}^{n} \mathbb{E}[X_k] = \sum_{k=1}^{n} p_k(x)$$

where $p_k(x)$ is the probability that element x is involved in the partitioning of the k-th pivot insertion.

To compute the probability, we observe that $X_k = 1$ iff the k-th pivot y is one of the two elements that bound the interval containing x after k pivots are chosen randomly. For a fixed choice of k initial pivots, the probability that y is one of the two bounding elements is at most $\frac{2(k-1)!}{k!} = \frac{2}{k}$. The numerator represents the number of permutations with one of the bounding elements being the last pivot. Although this is the probability conditioned on the choice of the first k pivots, clearly unconditioning would also give us the same probability. Therefore the expected number of comparisons involving x is $\sum_{k=1}^{n} \frac{1}{k} = O(\log n)$. Further the total expected number of comparisons is $O(n \log n)$ by summing over all elements.

A $\mathsf{ZPP}^{\mathsf{NP}[1]}$ Lifting Theorem

Thomas Watson
University of Memphis, Memphis, TN, USA
Thomas.Watson@memphis.edu

Abstract

The complexity class $\mathsf{ZPP}^{\mathsf{NP}[1]}$ (corresponding to zero-error randomized algorithms with access to one NP oracle query) is known to have a number of curious properties. We further explore this class in the settings of time complexity, query complexity, and communication complexity.

- For starters, we provide a new characterization: $\mathsf{ZPP}^{\mathsf{NP}[1]}$ *equals* the restriction of $\mathsf{BPP}^{\mathsf{NP}[1]}$ where the algorithm is only allowed to err when it forgoes the opportunity to make an NP oracle query.
- Using the above characterization, we prove a *query-to-communication lifting theorem*, which translates any $\mathsf{ZPP}^{\mathsf{NP}[1]}$ decision tree lower bound for a function f into a $\mathsf{ZPP}^{\mathsf{NP}[1]}$ communication lower bound for a two-party version of f.
- As an application, we use the above lifting theorem to prove that the $\mathsf{ZPP}^{\mathsf{NP}[1]}$ communication lower bound technique introduced by Göös, Pitassi, and Watson (ICALP 2016) is not tight. We also provide a "primal" characterization of this lower bound technique as a complexity class.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Query complexity, communication complexity, lifting

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.59

Related Version A full version of the paper is available at `https://eccc.weizmann.ac.il/report/2017/139/`.

Funding Supported by NSF grant CCF-1657377.

Acknowledgements I thank Mika Göös and Toniann Pitassi for discussions, and anonymous reviewers for thoughtful comments.

1 Introduction

Query-to-communication lifting is a paradigm for proving lower bounds in communication complexity [30, 26, 34] using lower bounds in query complexity (a.k.a. decision tree complexity) [40, 9, 26]. This technique has yielded a wide array of applications, including lower bounds for the Clique vs. Independent Set communication game and the related Alon–Saks–Seymour conjecture in graph theory [15, 6], separations between communication complexity and partition number [20, 3, 16, 4, 22], lower bounds for monotone circuits, monotone span programs, and proof complexity [35, 7, 25, 19, 14, 36, 33], new and unified proofs of quantum communication lower bounds [38] and of separations between randomized and quantum communication complexity [22, 1, 2], lower bounds for LP and SDP relaxations of CSPs [12, 31, 29], separations between communication complexity classes [10, 28, 18, 21, 17, 8], and lower bounds for finding Nash equilibria [37, 5].

The basic format of the technique is a two-step approach in which a relatively simple problem-specific argument is combined with fairly heavy-duty general-purpose machinery for handling communication protocols. More specifically:

Class	Reference
P	[35, 20]
NP	[18, 15]
BPP	[22]
P^{NP}	[17]
$\mathsf{ZPP}^{\mathsf{NP}[1]}$	Theorem 2
SBP	[18]
AWPP	[38]
PostBPP	[18]
PP	[38]

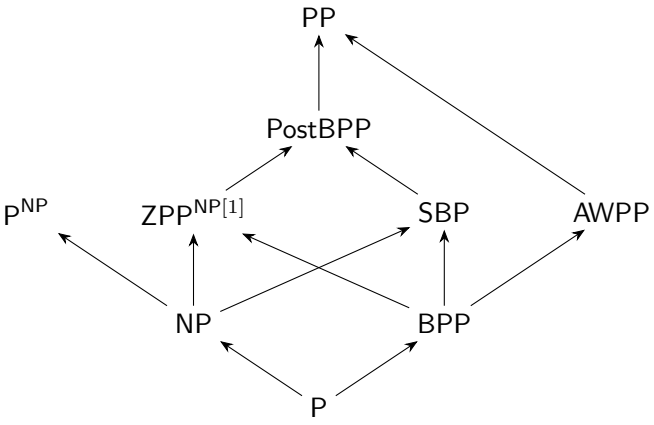


Figure 1 Classes with a known query-to-communication lifting theorem. $\mathcal{C}_1 \to \mathcal{C}_2$ denotes $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

(1) Capture the combinatorial core of the desired communication complexity lower bound by proving an analogous query complexity lower bound.

(2) Apply a *lifting theorem* showing that the query complexity of any boolean function f is essentially the same as the communication complexity of a two-party version of f.

The availability of a lifting theorem greatly eases the burden on the lower bound prover, since query lower bounds are generally much easier to prove than communication lower bounds.

The lifting theorem is with respect to a particular model of computation: deterministic, randomized, nondeterministic, and so on; it is convenient to associate these models with their corresponding classical time-bounded complexity classes: P, BPP, NP, and so on. This idea has led to an ongoing project: prove lifting theorems for the query/communication analogues of various classical complexity classes. Figure 1 shows the main classes for which a lifting theorem is known, along with primary references. Even the less well-known classes sometimes correspond to standard measures in the query/communication settings; e.g., AWPP corresponds to approximate polynomial degree in query complexity and to log of approximate rank in communication complexity. Some notable classes for which a lifting theorem is not known include BQP, UP, and MA. Proving a lifting theorem for AM would be a breakthrough, as it is notoriously open to prove any strong AM-type communication lower bound for an explicit function, but is trivial to do so in the query complexity setting.

Our central contribution is a lifting theorem for the slightly exotic class $\mathsf{ZPP}^{\mathsf{NP}[1]}$, which corresponds to randomized algorithms that can make one call to an NP oracle, output the correct answer with probability $\geq 3/4$, and output $\perp$ with the remaining probability. This model is interesting partly because it has so many curious properties, one of which is that it is robust with respect to the success probability threshold: by [13], the success probability can be efficiently amplified as long as it is $> 1/2$ (which is nontrivial since the standard method for amplification would use multiple independent trials, resulting in multiple NP oracle queries). In terms of relations to other classes, $\mathsf{ZPP}^{\mathsf{NP}[1]}$ contains BPP [11] and is contained in $\mathsf{S_2P}$ [11] and in PostBPP (a.k.a. $\mathsf{BPP}_{\mathsf{path}}$) [21]. If we generalized $\mathsf{ZPP}^{\mathsf{NP}[1]}$ to allow success probability slightly $< 1/2$, or to allow two nonadaptive NP oracle calls, either way the class would contain $\mathsf{AM} \cap \mathsf{coAM}$, and hence proving explicit lower bounds for the communication version would yield breakthrough AM communication lower bounds; in this sense, $\mathsf{ZPP}^{\mathsf{NP}[1]}$ is just shy of the communication lower bound frontier. $\mathsf{ZPP}^{\mathsf{NP}[1]}$ also shows up frequently in the literature on the "two queries problem" [39].

Our starting point is to uncover another curious property of $\mathsf{ZPP}^{\mathsf{NP}[1]}$: we prove it is equivalent (in time, query, and communication complexities) to a new model we dub $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$, which corresponds to randomized algorithms that can make one call to an NP oracle, output the correct answer with probability $\geq 3/4$, and are only allowed to err when they choose not to call the NP oracle. This equivalence plays a crucial role in our proof of the lifting theorem for $\mathsf{ZPP}^{\mathsf{NP}[1]}$.

Once we have the lifting theorem, the natural application domain is to prove new $\mathsf{ZPP}^{\mathsf{NP}[1]}$-type communication lower bounds. [21] developed a technique for proving such lower bounds, and we use our lifting theorem to derive new separations, which imply that the technique from [21] is not tight. This is analogous to the main application from [17], in which a P^{NP} lifting theorem was used to show that the P^{NP}-type communication lower bound technique from [24, 32] is not tight. For context, we note that certain other communication complexity classes have similar lower bound techniques that *are* tight; e.g., the discrepancy bound captures PP communication [27], and the corruption bound captures SBP communication [23]. So for what class is the lower bound technique from [21] tight, if not $\mathsf{ZPP}^{\mathsf{NP}[1]}$? We also answer this question in the full version of this paper. The class did not have a standard name, but it turns out to have a reasonably simple definition.

1.1 Statement of results

We formally define $\mathsf{ZPP}^{\mathsf{NP}[1]}$ and $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$ and their query/communication analogues in Section 2. For any model $\mathcal{C}$ (such as $\mathsf{ZPP}^{\mathsf{NP}[1]}$ or $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$) we use $\mathcal{C}$ for the polynomial time complexity class, $\mathcal{C}^{\mathsf{dt}}$ and $\mathcal{C}^{\mathsf{cc}}$ for the polylog query and communication complexity classes, and $\mathcal{C}^{\mathsf{dt}}(f)$ and $\mathcal{C}^{\mathsf{cc}}(F)$ for the corresponding query and communication complexities of a partial function $f\colon \{0,1\}^n \to \{0,1\}$ and a partial two-party function $F\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (we also consider F's where Alice and Bob have unequal but polynomially-related input lengths). We use $\tilde{\Theta}$ to hide polylog(n) factors. We prove the following characterization in Section 3.

▶ **Theorem 1.**

(i) $\mathsf{ZPP}^{\mathsf{NP}[1]} = \mathsf{CautiousBPP}^{\mathsf{NP}[1]}$.

(ii) $\mathsf{ZPP}^{\mathsf{NP}[1]^{\mathsf{dt}}}(f) = \tilde{\Theta}(\mathsf{CautiousBPP}^{\mathsf{NP}[1]^{\mathsf{dt}}}(f))$ *for all* f.

(iii) $\mathsf{ZPP}^{\mathsf{NP}[1]^{\mathsf{cc}}}(F) = \tilde{\Theta}(\mathsf{CautiousBPP}^{\mathsf{NP}[1]^{\mathsf{cc}}}(F))$ *for all* F.

We now prepare to state the lifting theorem. For $f\colon \{0,1\}^n \to \{0,1\}$ (called the *outer function*) and $g\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ (called the *gadget*), their composition $f \circ g^n\colon \mathcal{X}^n \times \mathcal{Y}^n \to \{0,1\}$ is the two-party function where Alice gets $x = (x_1, \ldots, x_n) \in \mathcal{X}^n$, Bob gets $y = (y_1, \ldots, y_n) \in \mathcal{Y}^n$, and the goal is to evaluate $(f \circ g^n)(x,y) \coloneqq f(g(x_1,y_1), \ldots, g(x_n,y_n))$. Note that any deterministic (P-type) decision tree for f can be turned into a deterministic protocol for $f \circ g^n$ where Alice and Bob communicate to evaluate $g(x_i, y_i)$ whenever the decision tree queries the i^{th} input bit of f. A similar thing can be done in other models besides deterministic. The essence of a lifting theorem is to go in the other direction: convert a protocol for $f \circ g^n$ into a comparable-cost decision tree for f. In other words, if g is sufficiently complicated, then it hides the input bits to f so well that a communication protocol cannot do any better than just running a decision tree for f.

We use the *index* gadget $\textsc{Ind}_m\colon [m] \times \{0,1\}^m \to \{0,1\}$ mapping $(x,y) \mapsto y_x$, where m is a sufficiently large polynomial in n. This gadget has previously been used for the P, BPP, and P^{NP} lifting theorems. (In some cases, lifting theorems with simpler gadgets are known, but for many applications the index gadget is fine.)

▶ **Theorem 2.** *Let $m = m(n) \coloneqq n^C$ for a large enough constant C. For every $f\colon \{0,1\}^n \to \{0,1\}$,*

(i) $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ \textsc{Ind}_m^n) = \tilde{\Theta}(\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{dt}}(f))$,

(ii) $\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ \textsc{Ind}_m^n) = \Theta(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \cdot \log n)$.

Note that part (i) of Theorem 2 is a corollary of part (ii), since by Theorem 1,

$$\begin{aligned}\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ \textsc{Ind}_m^n) &= \tilde{\Theta}(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ \textsc{Ind}_m^n)) \\ &= \tilde{\Theta}(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f)) = \tilde{\Theta}(\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{dt}}(f)).\end{aligned}$$

We are not aware of a way to prove part (i) directly, without going through Theorem 1. To prove part (ii) (in Section 4), we combine tools and techniques from the proofs of lifting theorems for BPP [22], NP [18, 15], and P^{NP} [17], along with some new technical contributions.

Two of the main results in [21] are $\mathsf{MA}^{\mathsf{cc}} \not\subseteq \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$ and $\mathsf{US}^{\mathsf{cc}} \not\subseteq \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$, where MA and US are the classes associated with "Merlin–Arthur games" and "unique witnesses" respectively (more precise definitions are deferred to Section 5). The proofs introduced a certain lower bound technique – let us use $\mathcal{B}^{\mathsf{cc}}(F)$ for the largest bound attainable for F using this technique, and $\mathcal{B}^{\mathsf{cc}}$ for the class of all F's with $\mathcal{B}^{\mathsf{cc}}(F) \leq \mathrm{polylog}(n)$ – and showed that $\mathsf{MA}^{\mathsf{cc}} \not\subseteq \mathcal{B}^{\mathsf{cc}}$, $\mathsf{US}^{\mathsf{cc}} \not\subseteq \mathcal{B}^{\mathsf{cc}}$, and $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}} \subseteq \mathcal{B}^{\mathsf{cc}}$. The definition of $\mathcal{B}^{\mathsf{cc}}$ is not important for now, but we provide it in the full version of this paper, where we show that it can be characterized as a more natural complexity class.

Since $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$ is closed under complement (whereas $\mathcal{B}^{\mathsf{cc}}$ is not), we have $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}} \subseteq \mathcal{B}^{\mathsf{cc}} \cap \mathsf{co}\mathcal{B}^{\mathsf{cc}}$. A natural question is whether the latter is actually an equality, i.e., whether the lower bound technique of [21] for $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$ is tight. Since [21] observed that $\mathsf{MA}^{\mathsf{cc}}, \mathsf{US}^{\mathsf{cc}} \subseteq \mathsf{co}\mathcal{B}^{\mathsf{cc}}$, we have $\mathsf{MA}^{\mathsf{cc}} \cap \mathsf{coMA}^{\mathsf{cc}}$, $\mathsf{US}^{\mathsf{cc}} \cap \mathsf{coUS}^{\mathsf{cc}} \subseteq \mathcal{B}^{\mathsf{cc}} \cap \mathsf{co}\mathcal{B}^{\mathsf{cc}}$, and thus the following result (proven in Section 5 using Theorem 2) answers this question in the negative (in two different ways).

▶ **Theorem 3.**

(i) $\mathsf{MA}^{\mathsf{cc}} \cap \mathsf{coMA}^{\mathsf{cc}} \not\subseteq \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$.

(ii) $\mathsf{US}^{\mathsf{cc}} \cap \mathsf{coUS}^{\mathsf{cc}} \not\subseteq \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$.

2 Definitions

We set up notation and provide the formal definitions of $\mathsf{ZPP}^{\mathsf{NP}[1]}$ and $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$. For the query and communication complexity versions, we follow the convention of using the complexity class names as complexity measures. That is, $\mathcal{C}^{\mathsf{dt}}(f)$ denotes the minimum cost of any correct $\mathcal{C}$-type decision tree for f, and $\mathcal{C}^{\mathsf{dt}}$ also denotes the class of families of partial f's with $\mathcal{C}^{\mathsf{dt}}(f) \leq \mathrm{polylog}(n)$; similarly, $\mathcal{C}^{\mathsf{cc}}(F)$ denotes the minimum cost of any correct $\mathcal{C}$-type communication protocol for F, and $\mathcal{C}^{\mathsf{cc}}$ also denotes the class of families of partial F's with $\mathcal{C}^{\mathsf{cc}}(F) \leq \mathrm{polylog}(n)$ (assuming Alice and Bob have polynomially-related input lengths).

In the query complexity setting, "query" actually has two meanings for us: a decision tree makes queries to individual input bits, then it forms an NP-type (DNF) oracle query.

We think of a randomized algorithm M as taking a uniformly random string $s \in \{0,1\}^r$ (for some number of coins r that depends on the input length); we let $M_s(x)$ denote M running on input x with outcome s. Similarly, we think of a randomized (in our case, $\mathsf{ZPP}^{\mathsf{NP}[1]}$-type or $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type) decision tree T or communication protocol Π as the uniform distribution over a multiset of corresponding non-randomized T_s's or Π_s's indexed by $s \in \{0,1\}^r$; we denote this as $T \sim \{T_s : s \in \{0,1\}^r\}$ or $\Pi \sim \{\Pi_s : s \in \{0,1\}^r\}$.

2.1 $\mathsf{ZPP}^{\mathsf{NP}[1]}$

$\mathsf{ZPP}^{\mathsf{NP}[1]}$ consists of all languages L for which there is a polynomial-time randomized algorithm M (taking input x and coin tosses $s \in \{0,1\}^r$) and a language $L' \in \mathsf{NP}$ such that the following hold.

Syntax: The computation of $M_s(x)$ produces an oracle query q and a function out: $\{0,1\} \to \{0,1,\perp\}$; the output is then $out(L'(q))$.
Correctness: The output is always $L(x)$ or $\perp$, and is $L(x)$ with probability $\geq 3/4$.

We define a $\mathsf{ZPP}^{\mathsf{NP}[1]}$-type decision tree T for f on input x as follows.

Syntax: $T \sim \{T_s : s \in \{0,1\}^r\}$ where each T_s makes queries to the bits of x until it reaches a leaf, which is labeled with a DNF D and a function out: $\{0,1\} \to \{0,1,\perp\}$; the output is then $out(D(x))$.
Correctness: The output is always $f(x)$ or $\perp$, and is $f(x)$ with probability $\geq 3/4$.
Cost: The maximum height of any T_s, plus the maximum width of any DNF appearing at a leaf.

We define a $\mathsf{ZPP}^{\mathsf{NP}[1]}$-type communication protocol Π for F on input (x,y) as follows.

Syntax: $\Pi \sim \{\Pi_s : s \in \{0,1\}^r\}$ where each Π_s communicates until it reaches a leaf, which is labeled with a multiset of rectangles $\{R^w : w \in \{0,1\}^k\}$ (for some k) and a function out: $\{0,1\} \to \{0,1,\perp\}$; the output is then out applied to the indicator of whether $(x,y) \in \bigcup_w R^w$.
Correctness: The output is always $F(x,y)$ or $\perp$, and is $F(x,y)$ with probability $\geq 3/4$.
Cost: The maximum communication cost of any Π_s, plus the maximum k at any leaf.

A priori, the value $3/4$ seems arbitrary since it is not clear whether $\mathsf{ZPP}^{\mathsf{NP}[1]}$ is amenable to amplification of the success probability (naively doing repeated trials would increase the number of NP queries). However, [13] showed that amplification is actually possible, so we may use any constant $> 1/2$ for the success probability in the definition of $\mathsf{ZPP}^{\mathsf{NP}[1]}$ (while affecting the measures $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{dt}}(f)$ and $\mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}(F)$ by only constant factors).

2.2 $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$

$\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$ consists of all languages L for which there is a polynomial-time randomized algorithm M (taking input x and coin tosses $s \in \{0,1\}^r$) and a language $L' \in \mathsf{NP}$ such that the following hold.

Syntax: The computation of $M_s(x)$ either directly outputs a bit (without invoking the oracle) or produces an oracle query q and a nonconstant function out: $\{0,1\} \to \{0,1\}$; in the latter case the output is then $out(L'(q))$.
Correctness: The output is $L(x)$ with probability $\geq 3/4$, and is $L(x)$ for all s such that $M_s(x)$ makes an oracle query.

We define a $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type decision tree T for f on input x as follows.

Syntax: $T \sim \{T_s : s \in \{0,1\}^r\}$ where each T_s makes queries to the bits of x until it reaches a leaf, which is labeled with either an output bit, or a DNF D and a nonconstant function out: $\{0,1\} \to \{0,1\}$; in the latter case the output is then $out(D(x))$.
Correctness: The output is $f(x)$ with probability $\geq 3/4$, and is $f(x)$ for all s such that $T_s(x)$ makes a DNF query.
Cost: The maximum height of any T_s, plus the maximum width of any DNF appearing at a leaf.

We define a $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type communication protocol Π for F on input (x, y) as follows.

Syntax: $\Pi \sim \{\Pi_s : s \in \{0,1\}^r\}$ where each Π_s communicates until it reaches a leaf, which is labeled with either an output bit, or a multiset of rectangles $\{R^w : w \in \{0,1\}^k\}$ (for some k) and a nonconstant function $out\colon \{0,1\} \to \{0,1\}$; in the latter case the output is then out applied to the indicator of whether $(x, y) \in \bigcup_w R^w$.

Correctness: The output is $F(x, y)$ with probability $\geq 3/4$, and is $F(x, y)$ for all s such that $\Pi_s(x, y)$ makes a "union of rectangles" query.

Cost: The maximum communication cost of any Π_s, plus the maximum k at any leaf.

The success probability of any $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type computation can be amplified by taking the majority vote of multiple independent trials – except if at least one trial results in an NP-type oracle query then (to avoid multiple oracle queries) we just use the output of one such trial since we know it will be correct. Thus just like for BPP-type computations, success probability $1/2 + \varepsilon$ can be amplified to $1 - \delta$ with a $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ factor overhead in cost.

3 $\mathsf{ZPP}^{\mathsf{NP}[1]} = \mathsf{CautiousBPP}^{\mathsf{NP}[1]}$

We now prove Theorem 1, starting with part (i). First assume $L \in \mathsf{ZPP}^{\mathsf{NP}[1]}$, witnessed by a randomized algorithm M (taking input x and coin tosses $s \in \{0,1\}^r$) and $L' \in \mathsf{NP}$. To see that $L \in \mathsf{CautiousBPP}^{\mathsf{NP}[1]}$, consider this randomized algorithm with oracle access to L':

1. Sample $s \in \{0,1\}^r$ and run $M_s(x)$ until it produces q and out.
2. If $out(0) = out(1)$ then output this common bit, or an arbitrary bit if $out(0) = out(1) = \perp$.
3. Else if one of $out(0), out(1)$ is $\perp$ then output whichever is not $\perp$.
4. Else invoke the oracle on q and output $out(L'(q))$.

Consider any s for which this algorithm outputs the wrong bit: then it did not make an oracle query (since M never outputs the wrong bit), and $M_s(x)$ would have output $\perp$ (because of either line 2, or line 3 with $out(L'(q)) = \perp$ and $out(1 - L'(q)) \neq L(x)$). Hence this algorithm correctly solves L, with error probability at most that of M.

For the converse direction, we generalize the argument from [11] that $\mathsf{BPP} \subseteq \mathsf{ZPP}^{\mathsf{NP}[1]}$. Assume $L \in \mathsf{CautiousBPP}^{\mathsf{NP}[1]}$, witnessed by a randomized algorithm M (taking input x and coin tosses $s \in \{0,1\}^r$) and $L' \in \mathsf{NP}$. Assume that this has already been amplified so the error probability is $< 1/4r$ (by the remark at the end of Section 2.2). For a fixed input x and $b \in \{0,1\}$, let

$$S_b \coloneqq \{s \in \{0,1\}^r : M_s(x) \text{ outputs } b \text{ without invoking the oracle}\}.$$

To see that $L \in \mathsf{ZPP}^{\mathsf{NP}[1]}$, consider the following randomized algorithm:

1. Sample $s \in \{0,1\}^r$ and run $M_s(x)$ until it produces either an output b (so $s \in S_b$) or q and out.
2. If it produced q and out then ask the NP oracle for the value of $L'(q)$ and output $out(L'(q))$.
3. Else sample independent strings $s^1, \ldots, s^{4r} \in \{0,1\}^r$ and ask the NP oracle whether $\bigcup_i (S_b \oplus s^i) \neq \{0,1\}^r$ (i.e., whether there exists an s' such that for every i, $s' \oplus s^i \notin S_b$); output $\perp$ if so and b if not.

Note that this algorithm never outputs the wrong bit: if $s \in S_b$ for $b = 1 - L(x)$, then $|S_b| < 2^r/4r$ so by a union bound, $\left|\bigcup_i (S_b \oplus s^i)\right| < 4r \cdot (2^r/4r) = 2^r$ and hence the NP oracle returns 1 on line 3 and the algorithm outputs $\perp$. For the success probability, consider two cases. If $|S_0 \cup S_1| \leq 2^r/4$, then line 2 executes (guaranteeing correct output) with probability $\geq 3/4$. Otherwise, since $|S_{1-L(x)}| < 2^r/4r$, we must have $|S_{L(x)}| > 2^r/4 - 2^r/4r > 2^r/5$ (we may

assume r is at least a large enough constant), so by a union bound over all $s' \in \{0,1\}^r$, the probability over $s^1, \ldots, s^{4r}$ that $\bigcup_i (S_{L(x)} \oplus s^i) \neq \{0,1\}^r$ is $< 2^r \cdot (4/5)^{4r} \leq (5/6)^r \leq 1/5$. In this latter case, the probability of outputting $\perp$ is

$$\begin{aligned} &\mathbb{P}[b = 1 - L(x)] + \mathbb{P}\big[\textstyle\bigcup_i (S_b \oplus s^i) \neq \{0,1\}^r \,\big|\, b = L(x)\big] \cdot \mathbb{P}[b = L(x)] \\ &\leq\ 1/4r + (1/5) \cdot |S_{L(x)}|/2^r \ \leq\ 1/4. \end{aligned}$$

In both cases the success probability is $\geq 3/4$.

Parts (ii) and (iii) are proved in the same way as part (i), but we must carefully analyze the cost. We summarize the differences. For the $\mathsf{ZPP}^{\mathsf{NP}[1]} \subseteq \mathsf{CautiousBPP}^{\mathsf{NP}[1]}$ direction, exactly the same argument works but using T_s or Π_s in place of M_s, and making the same DNF query or "union of rectangles" query rather than the same NP oracle query on line 4. This shows $\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \leq \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{dt}}(f)$ and $\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(F) \leq \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}(F)$.

Now consider the $\mathsf{CautiousBPP}^{\mathsf{NP}[1]} \subseteq \mathsf{ZPP}^{\mathsf{NP}[1]}$ direction for parts (ii) and (iii). By standard sparsification of the randomness, we may assume T or Π uses only $O(\log n)$ coin tosses (while affecting the success probability by only $\pm o(1)$). Then as noted at the end of Section 2.2, we may amplify with $O(\log \log n)$ repetitions so r becomes $O(\log n \cdot \log \log n)$ and the error probability becomes $\leq 1/\log^2 n < 1/4r$. As above, we use T_s or Π_s in place of M_s, and make the same DNF query or "union of rectangles" query rather than the same NP oracle query on line 2. For line 3, we note that the predicate $\bigcup_i (S_b \oplus s^i) \neq \{0,1\}^r$, as a function of the input x or (x,y), can be computed by nondeterministically guessing s' and running $T_{s' \oplus s^i}(x)$ or $\Pi_{s' \oplus s^i}(x,y)$ for each $i \in [4r]$; this can be expressed as a DNF of width $4r \cdot$(cost of amplified T), or as a union of 2^k rectangles with $k = r + 4r \cdot$(cost of amplified Π). Thus, the overall cost is $O(r \cdot$(cost of amplified T or Π)$) \leq O($(cost of original T or Π)$\cdot \log n \cdot \log^2 \log n)$. This shows the following, finishing the proof of Theorem 1:

$$\begin{aligned} \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) &\leq O\big(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \cdot \log n \cdot \log^2 \log n\big), \\ \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}(F) &\leq O\big(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(F) \cdot \log n \cdot \log^2 \log n\big). \end{aligned}$$

4 Proof of the Lifting Theorem

We now prove Theorem 2. As noted in Section 1.1, we just need to show part (ii). It is straightforward to see that for all f,

$$\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ \textsc{Ind}_m^n) \ \leq\ O(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \cdot \log n)$$

since we can have the communication protocol run the optimal decision tree for f, communicating $O(\log n)$ bits to evaluate $\textsc{Ind}_m(x_i, y_i)$ whenever this bit is queried, and if a width-w DNF oracle query is formed then we can convert each of its $\leq n^w$ conjunctions into $\leq m^w$ rectangles, resulting in a "union of rectangles" oracle query that contributes $k = O(w \log n)$ to the cost. Thus, the bulk of the proof is to show that for all f,

$$\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \ \leq\ O(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ \textsc{Ind}_m^n)/\log n). \tag{1}$$

In Section 4.1 we provide relevant technical background from the proofs of earlier lifting theorems (mainly the one for BPP [22]). In Section 4.2 we describe how to simulate the communication protocol with a decision tree. In Section 4.3 we prove a key technical lemma.

4.1 Background

Abbreviate $G \coloneqq \textsc{Ind}_m^n$. We consider deterministic communication protocols on G's input domain $[m]^n \times (\{0,1\}^m)^n$, which we view as partitioned into *slices* $G^{-1}(z) = \{(x,y) : G(x,y) = z\}$, one for each $z \in \{0,1\}^n$. We let $|\Pi|$ denote the worst-case number of bits communicated by a deterministic protocol Π. We use boldface letters for random variables.

Let $\mathbb{H}_\infty(\boldsymbol{x}) \coloneqq \min_x \log(1/\mathbb{P}[\boldsymbol{x} = x])$ denote the usual min-entropy of a random variable $\boldsymbol{x}$. Supposing $\boldsymbol{x}$ is distributed over a set X, we define the *deficiency* of $\boldsymbol{x}$ as the nonnegative quantity $\mathbb{D}_\infty(\boldsymbol{x}) \coloneqq \log|X| - \mathbb{H}_\infty(\boldsymbol{x})$. A basic property is that if X is a Cartesian product then marginalizing $\boldsymbol{x}$ to some coordinates cannot increase the deficiency. For a set X we let $\boldsymbol{X}$ denote a random variable uniformly distributed on X.

The following definition and claim originate in the proof of the lifting theorems for NP, SBP, and PostBPP [18, 15]. They describe an invariant that Alice maintains throughout the simulation, and how to restore it (by fixing some coordinates, which will correspond to querying those input bits of f) when it gets violated.

▶ **Definition 4.** *A random variable $\boldsymbol{x} \in [m]^J$ (where $J \subseteq [n]$ is some index set) is called* δ-dense *if for every nonempty $I \subseteq J$, the coordinates $\boldsymbol{x}_I$ (marginally distributed over $[m]^I$) have min-entropy rate at least δ, i.e., $\mathbb{H}_\infty(\boldsymbol{x}_I) \geq \delta \cdot |I| \log m$.*

▷ Claim 5. If $A \subseteq [m]^J$ then there exist an $I \subseteq J$ of size $|I| \leq O(\mathbb{D}_\infty(\boldsymbol{A})/\log n)$ and a nonempty $A' \subseteq A$ such that $\boldsymbol{A}'$ is fixed on I and 0.9-dense on $J \setminus I$.

It is simple to check that all 2^n slices of G's input domain have the same size, and that the uniform distribution over any slice is marginally nearly-uniform on both Alice's input and Bob's input. The following lemma from [22] provides a sufficient condition for similar properties to hold even after we have queried some of the input bits of f.

▶ **Definition 6.** *For a partial assignment $\rho \in \{0,1,*\}^n$, define its* free *positions as* $\operatorname{free}\rho \coloneqq \rho^{-1}(*) \subseteq [n]$*, and its* fixed *positions as* $\operatorname{fix}\rho \coloneqq [n] \setminus \operatorname{free}\rho$*. A rectangle $X \times Y$ is called* ρ-structured *if $\boldsymbol{X}_{\operatorname{free}\rho}$ is 0.9-dense, $\boldsymbol{X}_{\operatorname{fix}\rho}$ is fixed, and each element of $G(X \times Y) \subseteq \{0,1\}^n$ is consistent with ρ.*

▶ **Definition 7.** *A distribution $\mathcal{D}_1$ is said to be* ε-pointwise-close *to a distribution $\mathcal{D}_2$ if for every outcome, the probability under $\mathcal{D}_1$ is within a factor $1 \pm \varepsilon$ of the probability under $\mathcal{D}_2$. The distributions are said to be* ε-close *if the statistical (total variation) distance is $\leq \varepsilon$.*

▶ **Lemma 8** ([22])**.** *Suppose $X \times Y$ is ρ-structured and $\mathbb{D}_\infty(\boldsymbol{Y}) \leq n^3$. Then:*

- **(i)** *For any $z \in \{0,1\}^n$ consistent with ρ, the uniform distribution on $G^{-1}(z) \cap X \times Y$ (which is nonempty) has both of its marginal distributions $o(1)$-close to $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively.*
- **(ii)** *$G(\boldsymbol{X}, \boldsymbol{Y})$ is $o(1)$-pointwise-close to the uniform distribution over the set of all z consistent with ρ.*

Now we come to the main part of the proof of the BPP lifting theorem from [22]. It shows that, given query access to z, we can approximately sample the transcript that would be generated by a communication protocol on a random input from z's slice. In fact, this simulation maintains some invariants, which we need to expose (in the "furthermore" part of the lemma) for use in the subsequent "NP oracle query" phase of our simulation.

▶ **Definition 9.** *A deterministic protocol $\overline{\Pi}$ is said to be a* refinement *of a deterministic protocol Π if they have the same input domain and for every transcript rectangle $X \times Y$ of $\overline{\Pi}$, there exists a transcript rectangle of Π that contains $X \times Y$.*

▶ **Lemma 10** ([22]). *For every deterministic protocol Π on G's input domain with $|\Pi| \leq n \log m$, there exist a refinement $\overline{\Pi}$ and a randomized decision tree T of cost $O(|\Pi|/\log n)$ that on input $z \in \{0,1\}^n$ outputs a transcript of $\overline{\Pi}$ or $\perp$, such that the following two distributions are $o(1)$-close:*

$$\begin{aligned} \boldsymbol{t} &:= \text{output distribution of } T \text{ on input } z, \\ \boldsymbol{t}' &:= \text{transcript generated by } \overline{\Pi} \text{ when run on a random input } (\boldsymbol{x}, \boldsymbol{y}) \sim G^{-1}(z). \end{aligned}$$

Furthermore, for every (non-$\perp$) transcript output by T on input z with positive probability, the associated rectangle $X \times Y$ satisfies:

- **(i)** *$X \times Y$ is ρ-structured, where ρ corresponds to the results of the queries made by T (and is hence consistent with z),*
- **(ii)** $\mathbb{D}_\infty(\boldsymbol{Y}) \leq n^{2.5}$,
- **(iii)** $\mathbb{D}_\infty(\boldsymbol{X}_{\text{free}\,\rho}) \leq O(|\Pi|)$.

4.2 Simulation

▶ **Lemma 11.** *Let $X \times Y$ be a ρ-structured rectangle in G's input domain such that $\mathbb{D}_\infty(\boldsymbol{Y}) \leq n^{2.5}$. Suppose $\left\{R^w \subseteq X \times Y : w \in \{0,1\}^k\right\}$ is a collection of rectangles whose union covers exactly $G^{-1}(f^{-1}(1)) \cap X \times Y$. Then f can be computed by a width-$O((\mathbb{D}_\infty(\boldsymbol{X}_{\text{free}\,\rho}) + k)/\log n)$ DNF on the domain of inputs consistent with ρ.*

Lemma 11 is our key tool for converting the $\mathsf{NP}^{\mathsf{cc}}$ oracle query to an $\mathsf{NP}^{\mathsf{dt}}$ oracle query. The proof, which we give in Section 4.3, combines insights from the lifting theorem proofs for NP [18, 15] and P^{NP} [17] with new calculations. For now we use Lemma 11 to argue (1), thus finishing the proof of Theorem 2.

Let $\Pi \sim \left\{\Pi_s : s \in \{0,1\}^r\right\}$ be a $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type communication protocol for $f \circ G$ (and note WLOG the cost is $\leq n \log m$). Here is a $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type decision tree for f on input z:

1. Sample $s \in \{0,1\}^r$ and (eliding the dependence on s) let $\overline{\Pi}$ and T be the refinement and randomized decision tree from Lemma 10 applied to Π_s.
2. Sample T's coin tosses s' and run $T_{s'}$ on input z until it either outputs $\perp$ (in which case we halt and output an arbitrary bit) or produces a transcript t of $\overline{\Pi}$.
3. Let $X \times Y$ be the rectangle associated with t, and let t^* be the transcript of Π_s whose rectangle contains $X \times Y$.
4. If t^* outputs a bit, then we halt and output the same bit; otherwise let $\left\{R^w : w \in \{0,1\}^k\right\}$ and $out\colon \{0,1\} \to \{0,1\}$ be the rectangles and nonconstant function associated with t^*.
5. Since $X \times Y$ satisfies properties (i), (ii), (iii) from Lemma 10, we may apply Lemma 11 to the collection $\left\{R^w \cap X \times Y : w \in \{0,1\}^k\right\}$ (whose union covers exactly $G^{-1}(f^{-1}(out(1))) \cap X \times Y$ by the correctness of Π), using f if $out(1) = 1$ or $\neg f$ if $out(1) = 0$, to obtain a width-$O((|\Pi_s| + k)/\log n)$ DNF D that computes f or $\neg f$ (respectively) on all inputs consistent with ρ.
6. Output $out(D(z))$.

Since T makes $O(|\Pi_s|/\log n)$ queries and the DNF on line 5 has width $O((|\Pi_s| + k)/\log n)$, the above decision tree indeed has cost $O((\text{cost of } \Pi)/\log n)$. If it reaches line 5 and makes a DNF query, then the output is correct since z is consistent with ρ and hence $out(D(z)) = f(z)$. For the success probability, call t *good* if the corresponding t^* either outputs $f(z)$ directly or makes a "union of rectangles" query, and note that if the above decision tree generates

a good t then the output is correct (by the previous sentence). Hence, letting $\boldsymbol{t}, \boldsymbol{t}'$ be the $o(1)$-close random variables from Lemma 10 applied to $\Pi_{\boldsymbol{s}}$ (with $(\boldsymbol{x}, \boldsymbol{y}) \sim G^{-1}(z)$), we have

$$\begin{aligned}
\mathbb{P}[\text{output is correct}] &\geq \mathbb{E}_{\boldsymbol{s}}\big[\mathbb{P}_{\boldsymbol{s}'}[\boldsymbol{t} \text{ is a good transcript}]\big] \\
&\geq \mathbb{E}_{\boldsymbol{s}}\big[\mathbb{P}_{\boldsymbol{x},\boldsymbol{y}}[\boldsymbol{t}' \text{ is good}] - o(1)\big] \\
&= \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\big[\mathbb{P}_{\boldsymbol{s}}[\boldsymbol{t}' \text{ is good}]\big] - o(1) \\
&= \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\big[\mathbb{P}_{\boldsymbol{s}}[\Pi_{\boldsymbol{s}}(\boldsymbol{x}, \boldsymbol{y}) \text{ outputs } f(z)]\big] - o(1) \\
&\geq \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[3/4] - o(1) \\
&= 3/4 - o(1).
\end{aligned}$$

We conclude that $\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \leq O(\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{cc}}(f \circ G)/\log n)$.[1]

4.3 Forming a DNF

We now prove Lemma 11. Fix any $z \in f^{-1}(1)$ consistent with ρ, and define $J \coloneqq \operatorname{free}\rho$. We need to show that there exists a width-$O((\mathbb{D}_\infty(\boldsymbol{X}_J) + k)/\log n)$ conjunction that accepts z but does not accept any input in $f^{-1}(0)$ consistent with ρ.

For each rectangle $R^w = X^w \times Y^w$ define the set of *weighty rows* as

$$A^w \;\coloneqq\; \big\{x \in X^w \,:\, |Y^w_x| \geq 2^{nm-n^3}\big\} \quad \text{where} \quad Y^w_x \;\coloneqq\; \big\{y \in Y^w \,:\, G(x,y) = z\big\}.$$

▷ Claim 12. There exists a $w \in \{0,1\}^k$ such that $|A^w| \geq |X|/2^{k+1}$.

Proof. Suppose for contradiction this is not the case. Then by Lemma 8.(i) we have

$$\frac{\big|G^{-1}(z) \cap \big(\bigcup_w A^w\big) \times Y\big|}{|G^{-1}(z) \cap X \times Y|} \;\leq\; \frac{|\bigcup_w A^w|}{|X|} + o(1) \;\leq\; \frac{2^k \cdot |X|/2^{k+1}}{|X|} + o(1) \;<\; 3/4. \tag{2}$$

On the other hand, since the R^w's cover $G^{-1}(z) \cap X \times Y$ and since $k \leq n \log m$ WLOG,

$$\big|G^{-1}(z) \cap \big(X \setminus \bigcup_w A^w\big) \times Y\big| \;\leq\; \big|\bigcup_{w,\, x \notin A^w} Y^w_x\big| \;\leq\; 2^k \cdot |X| \cdot 2^{nm-n^3} \;\leq\; |X| \cdot 2^{nm-n^{2.9}},$$

and by Lemma 8.(ii) and $\mathbb{D}_\infty(\boldsymbol{Y}) \leq n^{2.5} \leq n^3$ we have

$$|G^{-1}(z) \cap X \times Y| \;\geq\; |X| \cdot |Y| \cdot (1 - o(1))/2^{|J|} \;\geq\; |X| \cdot 2^{nm-n^{2.5}} \cdot (1 - o(1))/2^n \;\geq\; |X| \cdot 2^{nm-n^{2.6}},$$

and thus

$$\frac{\big|G^{-1}(z) \cap \big(X \setminus \bigcup_w A^w\big) \times Y\big|}{|G^{-1}(z) \cap X \times Y|} \;\leq\; \frac{|X| \cdot 2^{nm-n^{2.9}}}{|X| \cdot 2^{nm-n^{2.6}}} \;=\; 2^{n^{2.6}-n^{2.9}} \;<\; 1/4. \tag{3}$$

Now (2) and (3) form a contradiction. This proves the claim. ◁

[1] Let us summarize the fundamental reason we are unable to make this proof work directly for $\mathsf{ZPP}^{\mathsf{NP}[1]}$ (instead of $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$) without going through Theorem 1. Suppose we reach line 5 with $out(1) = \bot$ and $out(0) \neq \bot$. We would like to form a DNF that accepts those z's consistent with ρ where $G^{-1}(z) \cap X \times Y$ is covered by the union of $\big\{R^w \cap X \times Y \,:\, w \in \{0,1\}^k\big\}$ – and then output $\bot$ if the DNF accepts and output $out(0)$ if it rejects. The issue is that there may be some z's consistent with ρ such that $f(z) = out(0)$ but $G^{-1}(z) \cap X \times Y$ is partially covered by the union – even a fairly small coverage might result in the DNF accepting z. This could cause the overall probability of outputting $\bot$ on z to be much higher in the decision tree than in the communication protocol.

Now fix a $w \in \{0,1\}^k$ such that $|A^w| \geq |X|/2^{k+1}$ and hence $\mathbb{D}_\infty(\boldsymbol{A}^w_J) \leq \mathbb{D}_\infty(\boldsymbol{X}_J) + k + 1$. Applying Claim 5 to A^w_J, we can obtain an $I \subseteq J$ of size $|I| \leq O((\mathbb{D}_\infty(\boldsymbol{X}_J) + k)/\log n)$ and a nonempty $A' \subseteq A^w$ such that $\boldsymbol{A'}$ is fixed on $I \cup \operatorname{fix}\rho$ and 0.9-dense on $J \setminus I$. Consider the conjunction that accepts iff the I coordinates of the input equal z_I; we now argue that this conjunction satisfies the desired properties. It certainly has the right width and accepts z.

Define $\sigma \in \{0,1,*\}^n$ as the partial assignment that extends ρ by fixing the coordinates in I to z_I. Pick any $x' \in A'$ and let $B \coloneqq Y^w_{x'}$. Then $A' \times B$ is σ-structured (note that for all $(x,y) \in A' \times B$, $G(x,y)_I = G(x',y)_I = z_I$ since $x_I = x'_I$) and $\mathbb{D}_\infty(\boldsymbol{B}) \leq n^3$, and thus by Lemma 8.(ii), $G(\boldsymbol{A'}, \boldsymbol{B})$ is $o(1)$-pointwise-close to the uniform distribution over all strings consistent with σ. In particular, for every z' consistent with σ (i.e., for every z' consistent with ρ that is accepted by the conjunction) there exists an $(x,y) \in A' \times B$ such that $G(x,y) = z'$; since $A' \times B \subseteq R^w \subseteq G^{-1}(f^{-1}(1))$, this implies that $f(z') = 1$. In summary, the conjunction does not accept any input in $f^{-1}(0)$ consistent with ρ. This finishes the proof of Lemma 11.

5 Applications

We prove Theorem 3 in this section. Since $\mathsf{MA}^{\mathsf{cc}} \cap \mathsf{coMA}^{\mathsf{cc}}$, $\mathsf{US}^{\mathsf{cc}} \cap \mathsf{coUS}^{\mathsf{cc}} \subseteq \mathcal{B}^{\mathsf{cc}} \cap \mathsf{co}\mathcal{B}^{\mathsf{cc}}$, Theorem 3 cannot be shown using the lower bound technique from [21], so we instead prove the analogous separations in query complexity and apply our lifting theorem. We start by defining the query/communication versions of MA and US.

Merlin–Arthur games (MA) are the model where Merlin nondeterministically sends a message to Arthur (comprised of Alice and Bob in the communication setting), who is randomized and decides whether to accept. On a 1-input, there should exist a witness Merlin can send that makes Arthur accept with probability 1, and on a 0-input, Arthur should reject with probability $\geq 1/2$ no matter what Merlin sends. In the query/communication settings, the cost is Merlin's message length plus Arthur's query/communication cost.

The US model is like ordinary nondeterminism, except that an input is accepted iff there is *exactly one* witness that leads to acceptance (so, rejection means there are either 0 or ≥ 2 accepted witnesses). In query complexity, the cost is the maximum width of any of the witness conjunctions. In communication complexity, the cost is the log of the number of witness rectangles.

5.1 MA ∩ coMA

We now prove Theorem 3.(i). We start with a general technique for proving $\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}$ lower bounds. For a bit b, we say a conjunction is b-monochromatic for a partial function f if it rejects all $(1-b)$-inputs.

▶ **Lemma 13.** *Suppose f has no monochromatic conjunction of width $< k$. Then*

$$\mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f) \geq \min(k, \mathsf{BPP}^{\mathsf{dt}}(f)).$$

Proof. If f has a $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type decision tree of cost $< k$, then this decision tree must never make a DNF query (in which case it is just a BPP-type decision tree, showing that $\mathsf{BPP}^{\mathsf{dt}}(f) \leq \mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}(f)$). To see this, suppose for contradiction some leaf is labeled with a DNF query D and a function *out*, and consider the conjunction that accepts the inputs that lead to that leaf and are accepted by an arbitrarily chosen term of D (which WLOG is consistent with the partial assignment leading to the leaf). Then this conjunction has width $< k$ and is $out(1)$-monochromatic (as any input accepted by it would make the $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$-type decision tree output $out(1)$ after making a DNF query, for some outcome of the coin tosses, and hence could not be an $out(0)$-input). ◀

Let $n = 2\ell^2$, and define the partial function $f\colon \{0,1\}^n \to \{0,1\}$ that interprets its input as a pair of $\ell \times \ell$ boolean matrices (A, B), such that $f(A, B) = 1$ iff A has an all-1 row and every row of B is at most half 1's, and $f(A, B) = 0$ iff B has an all-1 row and every row of A is at most half 1's. Note that $f \in \mathsf{MA}^{\mathsf{dt}} \cap \mathsf{coMA}^{\mathsf{dt}}$ since an MA-type decision tree can guess a row in A and check that a random bit from that row is 1, and a coMA-type decision tree can guess a row in B and check that a random bit from that row is 1. This upper bound lifts to $f \circ \textsc{Ind}_m^n \in \mathsf{MA}^{\mathsf{cc}} \cap \mathsf{coMA}^{\mathsf{cc}}$. We now show that $f \notin \mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}$ which, by Theorems 1 and 2, implies that $f \circ \textsc{Ind}_m^n \notin \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$. This will yield Theorem 3.(i).

By Lemma 13, it suffices to show that (1) f has no monochromatic conjunction of width $\leq \ell/2$, and (2) $\mathsf{BPP}^{\mathsf{dt}}(f) \geq \Omega(\ell)$.

To see (1), consider any conjunction C of width $\leq \ell/2$: Since it does not touch every row of A, and it touches at most half the bits in each row of B, we can construct a 1-input accepted by C by putting all 1's in an untouched row of A, and filling the rest of the matrix entries with 0's (except those whose value is determined by C accepting). Similarly, there must exist a 0-input accepted by C. Thus C is not monochromatic.

For (2), it suffices to show that every cost-$o(\ell)$ deterministic decision tree succeeds with probability $< 3/4$ over the input distribution obtained by filling a uniformly random one of the 2ℓ rows with 1's (and letting all other entries of (A, B) be 0's). If the decision tree accepts after seeing only 0's, then conditioned on a random 0-input it continues to accept (and hence err) with probability $\geq 1 - o(1)$ (since the all-0's path of the decision tree only touches a $o(1)$ fraction of the rows). Similarly, if it rejects after seeing only 0's, then conditioned on a random 1-input it continues to reject (and hence err) with probability $\geq 1 - o(1)$. In either case, it errs with probability $\geq 1/2 - o(1)$ over an unconditioned random input.

5.2 $\mathsf{US} \cap \mathsf{coUS}$

We now prove Theorem 3.(ii). Let weight$(\cdot)$ refer to Hamming weight. For even n, define the partial function $f\colon \{0,1\}^n \to \{0,1\}$ that interprets its input as $(a, b) \in \{0,1\}^{n/2} \times \{0,1\}^{n/2}$, such that $f(a, b) = 1$ iff weight$(a) = 1$ and weight$(b) \in \{0, 2\}$, and $f(a, b) = 0$ iff weight$(b) = 1$ and weight$(a) \in \{0, 2\}$. Note that $f \in \mathsf{US}^{\mathsf{dt}} \cap \mathsf{coUS}^{\mathsf{dt}}$ since a US-type decision tree can guess the location of a 1 in a, and a coUS-type decision tree can guess the location of a 1 in b. This upper bound lifts to $f \circ \textsc{Ind}_m^n \in \mathsf{US}^{\mathsf{cc}} \cap \mathsf{coUS}^{\mathsf{cc}}$. We now show that $f \notin \mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}$ which, by Theorems 1 and 2, implies that $f \circ \textsc{Ind}_m^n \notin \mathsf{ZPP}^{\mathsf{NP}[1]\mathsf{cc}}$. This will yield Theorem 3.(ii).

Note that Lemma 13 cannot help us here, since this f does have small monochromatic conjunctions (e.g., a conjunction with two positive literals from a is 0-monochromatic), so we devise a different technique. In fact, we show something stronger than $f \notin \mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}$. Define $\mathsf{BPP}^{\mathsf{NP}[1]}$ in the natural way (two-sided error, and allowed to err after an NP oracle query is made), and notice that the class may depend on the exact choice of success probability (since the standard method of amplification involves multiple independent trials, which would increase the number of NP oracle queries). Let us use $\mathsf{BPP}_p^{\mathsf{NP}[1]}$ to indicate that the success probability must be $\geq p$ on each input. As $\mathsf{CautiousBPP}^{\mathsf{NP}[1]}$ can be efficiently amplified (see the end of Section 2.2), the following lemma implies that $f \notin \mathsf{CautiousBPP}^{\mathsf{NP}[1]\mathsf{dt}}$.

▶ **Lemma 14.** *For every constant $\varepsilon > 0$, $\mathsf{BPP}_{3/4+\varepsilon}^{\mathsf{NP}[1]\mathsf{dt}}(f) \geq \Omega(n)$.*

Proof. It suffices to show that every cost-$o(n)$ $\mathsf{P}^{\mathsf{NP}[1]}$-type decision tree succeeds with probability $\leq 3/4 + o(1)$ over the uniform distribution on valid inputs to f. Let v be the leaf reached after seeing only 0's, and say v is labeled with DNF D and function $out\colon \{0,1\} \to \{0,1\}$. Assume $out(1) = 1$ (the case $out(1) = 0$ is argued similarly). Con-

sider the joint random variables $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a}', \boldsymbol{b}'$ where $\boldsymbol{a}$ has a unique 1 in a random position, $\boldsymbol{b}$ is all 0's, $\boldsymbol{a}'$ is obtained from $\boldsymbol{a}$ by flipping a random 0 to 1, and $\boldsymbol{b}'$ is obtained from $\boldsymbol{b}$ by flipping a random 0 to 1. Note that $(\boldsymbol{a}, \boldsymbol{b})$ is the input distribution conditioned on weight$(a) = 1$ and weight$(b) = 0$, and $(\boldsymbol{a}', \boldsymbol{b}')$ is the input distribution conditioned on weight$(a) = 2$ and weight$(b) = 1$. We have $\mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ reaches } v] \geq 1 - o(1)$ and thus $\mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ reaches } v \text{ and is accepted}] \geq \mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ is accepted}] - o(1)$. Also, conditioned on any outcome of $(\boldsymbol{a}, \boldsymbol{b})$ that reaches v and is accepted, with probability $\geq 1 - o(1)$ the two flipped bits are not among those read along the path to v and not among those read by an arbitrarily chosen term of D that accepts $(\boldsymbol{a}, \boldsymbol{b})$, in which case $(\boldsymbol{a}', \boldsymbol{b}')$ also reaches v and is accepted. Thus, $\mathbb{P}\big[(\boldsymbol{a}', \boldsymbol{b}') \text{ reaches } v \text{ and is accepted} \,\big|\, (\boldsymbol{a}, \boldsymbol{b}) \text{ reaches } v \text{ and is accepted}\big] \geq 1 - o(1)$. Combining these, we get

$$\begin{aligned}
& \mathbb{P}[(\boldsymbol{a}', \boldsymbol{b}') \text{ is accepted}] \\
\geq\ & \mathbb{P}\big[(\boldsymbol{a}', \boldsymbol{b}') \text{ and } (\boldsymbol{a}, \boldsymbol{b}) \text{ both reach } v \text{ and are accepted}\big] \\
=\ & \mathbb{P}\big[(\boldsymbol{a}', \boldsymbol{b}') \text{ reaches } v \text{ and is accepted} \,\big|\, (\boldsymbol{a}, \boldsymbol{b}) \text{ reaches } v \text{ and is accepted}\big] \\
& \cdot \mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ reaches } v \text{ and is accepted}] \\
\geq\ & (1 - o(1)) \cdot (\mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ is accepted}] - o(1)) \\
=\ & \mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ is accepted}] - o(1).
\end{aligned}$$

Thus, under the uniform distribution on valid inputs to f,

$$\begin{aligned}
\mathbb{P}[\text{err}] \ \geq\ & \mathbb{P}\big[\text{err} \,\big|\, \text{weight}(a) = 1 \text{ and weight}(b) = 0\big]/4 \\
& + \mathbb{P}\big[\text{err} \,\big|\, \text{weight}(a) = 2 \text{ and weight}(b) = 1\big]/4 \\
=\ & \big(\mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ is rejected}] + \mathbb{P}[(\boldsymbol{a}', \boldsymbol{b}') \text{ is accepted}]\big)/4 \\
=\ & \big(1 - (\mathbb{P}[(\boldsymbol{a}, \boldsymbol{b}) \text{ is accepted}] - \mathbb{P}[(\boldsymbol{a}', \boldsymbol{b}') \text{ is accepted}])\big)/4 \ \geq\ (1 - o(1))/4. \qquad \blacktriangleleft
\end{aligned}$$

We complement Lemma 14 by noting that $\mathsf{BPP}^{\mathsf{NP}[1]\mathsf{dt}}_{3/4}(f) \leq 2$: With probability 1/4 each:

- accept iff weight$(a) \leq 1$,
- accept iff weight$(a) \geq 1$,
- reject iff weight$(b) \leq 1$,
- reject iff weight$(b) \geq 1$.

Hence $\mathsf{BPP}^{\mathsf{NP}[1]\mathsf{dt}}_{3/4} \not\subseteq \mathsf{BPP}^{\mathsf{NP}[1]\mathsf{dt}}_{3/4+\varepsilon}$, which implies that $\mathsf{BPP}^{\mathsf{NP}[1]}_{3/4} \not\subseteq \mathsf{BPP}^{\mathsf{NP}[1]}_{3/4+\varepsilon}$ in a relativized world. Thus, unlike $\mathsf{ZPP}^{\mathsf{NP}[1]}$, $\mathsf{BPP}^{\mathsf{NP}[1]}$ is not generally amenable to efficient amplification; this phenomenon has subsequently been fully explored in [41].

References

1. Scott Aaronson and Andris Ambainis. Forrelation: A Problem That Optimally Separates Quantum from Classical Computing. In *Proceedings of the 47th Symposium on Theory of Computing (STOC)*, pages 307–316. ACM, 2015. doi:10.1145/2746539.2746547.
2. Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in Query Complexity Using Cheat Sheets. In *Proceedings of the 48th Symposium on Theory of Computing (STOC)*, pages 863–876. ACM, 2016. doi:10.1145/2897518.2897644.
3. Andris Ambainis, Martins Kokainis, and Robin Kothari. Nearly Optimal Separations Between Communication (Or Query) Complexity And Partitions. In *Proceedings of the 31st Computational Complexity Conference (CCC)*, pages 4:1–4:14. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.CCC.2016.4.

4 Anurag Anshu, Aleksandrs Belovs, Shalev Ben-David, Mika Göös, Rahul Jain, Robin Kothari, Troy Lee, and Miklos Santha. Separations in Communication Complexity Using Cheat Sheets And Information Complexity. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 555–564. IEEE, 2016. doi:10.1109/FOCS.2016.66.

5 Yakov Babichenko and Aviad Rubinstein. Communication Complexity of Approximate Nash Equilibria. In *Proceedings of the 49th Symposium on Theory of Computing (STOC)*, pages 878–889. ACM, 2017. doi:10.1145/3055399.3055407.

6 Shalev Ben-David, Pooya Hatami, and Avishay Tal. Low-Sensitivity Functions from Unambiguous Certificates. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 28:1–28:23. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.ITCS.2017.28.

7 Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the Relative Complexity of Resolution Refinements and Cutting Planes Proof Systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000. doi:10.1137/S0097539799352474.

8 Adam Bouland, Lijie Chen, Dhiraj Holden, Justin Thaler, and Prashant Vasudevan. On The Power of Statistical Zero Knowledge. In *Proceedings of the 58th Symposium on Foundations of Computer Science (FOCS)*, pages 708–719. IEEE, 2017. doi:10.1109/FOCS.2017.71.

9 Harry Buhrman and Ronald de Wolf. Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

10 Harry Buhrman, Nikolai Vereshchagin, and Ronald de Wolf. On Computation and Communication with Small Bias. In *Proceedings of the 22nd Conference on Computational Complexity (CCC)*, pages 24–32. IEEE, 2007. doi:10.1109/CCC.2007.18.

11 Jin-Yi Cai and Venkatesan Chakaravarthy. On Zero Error Algorithms Having Oracle Access to One Query. *Journal of Combinatorial Optimization*, 11(2):189–202, 2006. doi:10.1007/s10878-006-7130-0.

12 Siu On Chan, James Lee, Prasad Raghavendra, and David Steurer. Approximate Constraint Satisfaction Requires Large LP Relaxations. *Journal of the ACM*, 63(4):34:1–34:22, 2016. doi:10.1145/2811255.

13 Richard Chang and Suresh Purini. Amplifying ZPP$^{\mathrm{SAT}[1]}$ and the Two Queries Problem. In *Proceedings of the 23rd Conference on Computational Complexity (CCC)*, pages 41–52. IEEE, 2008. doi:10.1109/CCC.2008.32.

14 Susanna de Rezende, Jakob Nordström, and Marc Vinyals. How Limited Interaction Hinders Real Communication (and What It Means for Proof and Circuit Complexity). In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 295–304. IEEE, 2016. doi:10.1109/FOCS.2016.40.

15 Mika Göös. Lower Bounds for Clique vs. Independent Set. In *Proceedings of the 56th Symposium on Foundations of Computer Science (FOCS)*, pages 1066–1076. IEEE, 2015. doi:10.1109/FOCS.2015.69.

16 Mika Göös, T.S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized Communication vs. Partition Number. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 52:1–52:15. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.ICALP.2017.52.

17 Mika Göös, Pritish Kamath, Toniann Pitassi, and Thomas Watson. Query-to-Communication Lifting for P$^{\mathrm{NP}}$. In *Proceedings of the 32nd Computational Complexity Conference (CCC)*, pages 12:1–12:16. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.CCC.2017.12.

18 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles Are Nonnegative Juntas. *SIAM Journal on Computing*, 45(5):1835–1869, 2016. doi:10.1137/15M103145X.

19 Mika Göös and Toniann Pitassi. Communication Lower Bounds via Critical Block Sensitivity. In *Proceedings of the 46th Symposium on Theory of Computing (STOC)*, pages 847–856. ACM, 2014. doi:10.1145/2591796.2591838.

20 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic Communication vs. Partition Number. In *Proceedings of the 56th Symposium on Foundations of Computer Science (FOCS)*, pages 1077–1088. IEEE, 2015. doi:10.1109/FOCS.2015.70.

21 Mika Göös, Toniann Pitassi, and Thomas Watson. The Landscape of Communication Complexity Classes. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 86:1–86:15. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.ICALP.2016.86.

22 Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-Communication Lifting for BPP. In *Proceedings of the 58th Symposium on Foundations of Computer Science (FOCS)*, pages 132–143. IEEE, 2017. doi:10.1109/FOCS.2017.21.

23 Mika Göös and Thomas Watson. Communication Complexity of Set-Disjointness for All Probabilities. *Theory of Computing*, 12(1):1–23, 2016. Special issue for selected papers from APPROX–RANDOM 2014. doi:10.4086/toc.2016.v012a009.

24 Russell Impagliazzo and Ryan Williams. Communication Complexity with Synchronized Clocks. In *Proceedings of the 25th Conference on Computational Complexity (CCC)*, pages 259–269. IEEE, 2010. doi:10.1109/CCC.2010.32.

25 Jan Johannsen. Depth Lower Bounds for Monotone Semi-Unbounded Fan-In Circuits. *RAIRO - Theoretical Informatics and Applications*, 35:277–286, 2001. doi:10.1051/ita:2001120.

26 Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012.

27 Hartmut Klauck. Lower Bounds for Quantum Communication Complexity. *SIAM Journal on Computing*, 37(1):20–46, 2007. doi:10.1137/S0097539702405620.

28 Hartmut Klauck. On Arthur Merlin Games in Communication Complexity. In *Proceedings of the 26th Conference on Computational Complexity (CCC)*, pages 189–199. IEEE, 2011. doi:10.1109/CCC.2011.33.

29 Pravesh Kothari, Raghu Meka, and Prasad Raghavendra. Approximating Rectangles by Juntas and Weakly-Exponential Lower Bounds for LP Relaxations of CSPs. In *Proceedings of the 49th Symposium on Theory of Computing (STOC)*, pages 590–603. ACM, 2017. doi:10.1145/3055399.3055438.

30 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

31 James Lee, Prasad Raghavendra, and David Steurer. Lower Bounds on the Size of Semidefinite Programming Relaxations. In *Proceedings of the 47th Symposium on Theory of Computing (STOC)*, pages 567–576. ACM, 2015. doi:10.1145/2746539.2746599.

32 Periklis Papakonstantinou, Dominik Scheder, and Hao Song. Overlays and Limited Memory Communication. In *Proceedings of the 29th Conference on Computational Complexity (CCC)*, pages 298–308. IEEE, 2014. doi:10.1109/CCC.2014.37.

33 Toniann Pitassi and Robert Robere. Strongly Exponential Lower Bounds for Monotone Computation. In *Proceedings of the 49th Symposium on Theory of Computing (STOC)*, pages 1246–1255. ACM, 2017. doi:10.1145/3055399.3055478.

34 Anup Rao and Amir Yehudayoff. *Communication Complexity*. In preparation, 2017.

35 Ran Raz and Pierre McKenzie. Separation of the Monotone NC Hierarchy. *Combinatorica*, 19(3):403–435, 1999. doi:10.1007/s004930050062.

36 Robert Robere, Toniann Pitassi, Benjamin Rossman, and Stephen Cook. Exponential Lower Bounds for Monotone Span Programs. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 406–415. IEEE, 2016. doi:10.1109/FOCS.2016.51.

37 Tim Roughgarden and Omri Weinstein. On the Communication Complexity of Approximate Fixed Points. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 229–238. IEEE, 2016. doi:10.1109/FOCS.2016.32.

38 Alexander Sherstov. The Pattern Matrix Method. *SIAM Journal on Computing*, 40(6):1969–2000, 2011. doi:10.1137/080733644.

39 Rahul Tripathi. The 1-Versus-2 Queries Problem Revisited. *Theory of Computing Systems*, 46(2):193–221, 2010. doi:10.1007/s00224-008-9126-x.

40 Nikolai Vereshchagin. Relativizability in Complexity Theory. In *Provability, Complexity, Grammars*, volume 192 of *AMS Translations, Series 2*, pages 87–172. American Mathematical Society, 1999.

41 Thomas Watson. Amplification with One NP Oracle Query. Technical Report TR18-058, Electronic Colloquium on Computational Complexity (ECCC), 2018. URL: https://eccc.weizmann.ac.il/report/2018/058.